The logic of infons

Yuri Gurevich
Microsoft Research Redmond
gurevich@microsoft.com

Itay Neeman
Department of Mathematics, UCLA
ineeman@math.ucla.edu

Abstract

Infons are pieces of information. In our work on the Distributed Knowledge Authorization Language (DKAL), we discovered that the logic of infons is a conservative extension of intuitionistic logic by means of connectives p `said` *and p* `put` *where p ranges over principals. We investigate infon logic and a* primal *fragment of it. In both cases, we develop model theory, prove soundness and completeness, and analyze the computational complexity of the ground multiple derivability (GMD) problem (which of the given ground queries follow form the given ground hypotheses). Our most involved technical result is a linear time algorithm for the GMD problem for the primal infon logic given a constant bound on the quotation depth of the hypotheses. In applications quotation-depth is small. Our result gives rise to a linear time algorithm for the GMD problem for SecPAL, a precursor of DKAL that expresses many important access control scenarios.*

Contents

1 Introduction

Infons are pieces of information, e.g. John is permitted to write into File 13 of the given computer system. We distinguish infons from propositions and do not assign truth values to infons. Instead we ask whether the infons in question are known to relevant parties a.k.a. principals. For example, does the owner of File 13 know that John is permitted to write into the file? Does the administrator of the system know? Does John know? The notion of infons is a basic notion in DKAL, Distributed Knowledge Authorization Language, that we have been developing and applying [9, 10, 11].

Originally we emphasized the algebra of infons. That was built on the intuition that infons are partially ordered: $a \leq b$ if the information in a is a part of that in b. At the bottom of that partial order are uninformative infons carrying no information whatsoever. And there is a natural union operation $a + b$ on infons. You know $a + b$ if you know a and you know b. Infon $a + b$ is the least upper bound of a and b. But one has to be careful with the intuitive information order because of the omniscience problem well known in epistemic logic. The partial order is intractable. Indeed, valid logic statements are uninformative. Accordingly we used a rule-based feasible approximation of the intuitive order.

The right tool to deal with infons is logic. The addition operation $x + y$ can be thought as conjunction. And we introduced implication [11]. If you know a and you know an implication $a \to b$ then you know b. For example, the copyright owner of a book may stipulate that a shop can sell the book provided that they have a valid license from the American Publishers Association. Implication allowed us to simplify DKAL. Here is one important example of that. Trust, in the form "principal p is trusted on saying infon x," used to be a primitive notion in DKAL. One of the so-called house rules in [9] is this: if you know that p is trusted on saying x and if you know that p said x then you know x. In [11] we define the notion p is trusted on saying infon x as follows: (p said x) $\to x$. The house rule now follows from pure logic: if you know (p said x) $\to x$ and if you know p

said x then you know x.

The algebraic intuition is still useful. In terms of the information order, $a \to b$ is the least solution x of the inequality $a + x \geq b$.

To our surprise, we discovered that infon logic with conjunction (a.k.a. addition) and implication is precisely (disjunction-free) intuitionistic logic. In addition to these two binary connectives $+$ and $\to$, infon logic has two unary connectives p `said` and p `put` for any principal p. The number of such unary connectives is unbounded as the number of principals is unbounded.

The *ground multiple derivability problem* $\mathrm{GMD}(L)$ for a logic L is to compute, given ground L-formulas $x_1, \ldots, x_m$ (the hypotheses) and $y_1, \ldots, y_n$ (the queries), which of the queries are derivable from the hypotheses. In the case $n = 1$, one speaks of the *derivability problem*.

A priori, the derivability problem for intuitionistic logic may seem to be intractable. Even the termination of the proof search does not seem to be guaranteed. But Kripke semantics for intuitionistic logic [13] clarifies the matter. The problem is solvable in polynomial space. It follows that termination is guaranteed. The problem is in fact polynomial space complete [17].

We prove that infon logic is similar in that respect. The derivability problem for infon logic is polynomial space complete in the worst case. This does not make infon logic useless. It works in great many practical cases. Typical cases are far from the worst ones.

Further, we noticed a narrow, almost impoverished fragment of infon logic — we call it *primal infon logic* — that suffices nonetheless for many purposes. For example, the ground multiple derivability problem for the authorization language SecPAL [5] reduces to that of primal infon logic. SecPAL is the precursor of DKAL and expresses many important access control scenarios.

Primal infon logic builds on a fragment of intuitionistic logic that we call weak intuitionistic logic. Weak intuitionistic logic is weak indeed and has not attracted much attention in logic literature. The only references of relevance that we know are [2] and [3]. One of the derivation rules of weak infon logic, namely $\dfrac{\Gamma \vdash y}{\Gamma \vdash (x \to y)}$, seems silly. Why do you need the implication if you already have the conclusion? But in the context of access control, primal infon logic is meaningful. And the particular inference rule makes good sense as well. A principal may know the conclusion y, but may be willing to share only a weaker part $x \to y$ of his knowledge with another principal. (Besides the implication may be the premise of another infon, but this is not specific to access control.)

We stratify the ground multiple derivability problem for primal logic according to the maximal quotation depth of hypotheses. In practice quotation depth is small. This is reflected in recent logic-based authorization languages (except for DKAL). They use construct `says` but do not nest it. This applies in particular to SecPAL, the richest of them.

We were able to prove that the ground multiple derivability problem for primal infon logic can be solved very efficiently, given a fixed bound on the quotation depth of hypotheses. For any such bound the problem is solvable in linear time. Our computation model is the usual random access machine, as in [7].

Finally let us notice that only one inference rule of full infon logic is absent (or rather weakened) in primal logic, namely the rule $\dfrac{\Gamma, x \vdash y}{\Gamma \vdash x \to y}$. One can use that rule sparingly, bridging incrementally between primal and full infon logics.

This paper is self-contained. In particular we do not presume that the reader is familiar with DKAL. We start, in §2, with a sequent calculus for infon logic. Then we weaken the calculus to define primal infon logic. In §3, for either logic, we define Kripke-type semantics, prove the soundness and completeness, and prove the subformula property that is so important for computational complexity. In §4, with an eye on the efficient solution of the ground multiple derivability problem, we introduce an alternative calculus for primal infon logic and prove that it is equivalent to the original calculus. In §5.1 we prove that the ground multiple derivability problem for weak intuitionistic logic is linear time. That result is generalized in the final §5.2 to any bounded-quotation-depth fragment of primal infon logic.

Related literature

The literature on intuitionistic logic is extensive. Our main sources were the old textbook of Kleene [12], the seminal paper of Kripke [13] and especially a recent textbook of Mints [15]. Arnon Avron independently discovered weak intuitionistic logic; see the forthcoming papers [2, 3]. He told us about that when we reported this work of ours recently at Tel Aviv University.

The literature on epistemic logic is extensive as well. Initially we hoped to find all we need in the literature. In particular, we consulted the books [8] and [14]. But our case seems to be different. The most common axiom $K_i\varphi \to \varphi$ of epistemic logic fails in our case. Principals may know wrong things and may lie. In DKAL, if principal A says x to a principal B (and if principal B is willing to listen) then B learns only that A `said` x.

The literature on algorithms was very helpful, especially the Cai-Paige paper [6].

Acknowledgments

Conversations with Nikolaj Bjørner, Andreas Blass and Grigori Mints have been most useful.

2 Infon logic and its primal fragment

In DKAL, infons are expressed by means of terms, and formulas have the form "principal knows infon". But this paper is devoted solely to the logic of infons. Accordingly we will treat infon expressions as formulas. In particular the Boolean constant `True` will be treated as an uninformative infon. (In DKAL, we cast `True` into an uninformative infon `True asInfon` [11].)

Language Primitive formulas have the form $A(t_1, \ldots, t_k)$ where A is an attribute name and each t_i is a term. For the purpose of this paper the structure of primitive formulas is of no importance except that the complexity of primitive formulas should be taken into account when we design linear time algorithms later on. There is also a primitive formula `True`.

There is a vocabulary of function and attribute names, and there is an infinite sequence of (the names of) *principals*. Composite formulas are built from primitive formulas by the following means.

- Conjunction. Taking into account the intended application, we denote the conjunction by $+$.
- Implication $\rightarrow$.
- Two unary connectives p `said` and p `put` for every principal p. If x is a formula then p `said` x and p `put` x are formulas as well.

In the sequel, formulas are by default in that language.

2.1 Infon logic

Infon logic was introduced in [11] but the present exposition is self-contained. Here we present infon logic as a sequent calculus which is essentially an extension of the intuitionistic propositional system NJp [15, §2.2]. A sequent has the form $\Gamma \vdash x$ where x is a formula and Γ is a set of formulas written as a sequence.

Axioms

$$\texttt{(True)} \qquad \vdash \texttt{True}$$

$$\texttt{(x2x)} \qquad x \vdash x$$

Inference rules

$$\texttt{(Premise Inflation)} \qquad \frac{\Gamma \vdash y}{\Gamma, x \vdash y}$$

$$\texttt{(+E)} \qquad \frac{\Gamma \vdash x + y}{\Gamma \vdash x} \qquad \frac{\Gamma \vdash x + y}{\Gamma \vdash y}$$

$$\texttt{(+I)} \qquad \frac{\Gamma \vdash x \quad \Gamma \vdash y}{\Gamma \vdash x + y}$$

$$\texttt{(}\rightarrow\texttt{E)} \qquad \frac{\Gamma \vdash x \quad \Gamma \vdash x \rightarrow y}{\Gamma \vdash y}$$

$$\texttt{(}\rightarrow\texttt{I)} \qquad \frac{\Gamma, x \vdash y}{\Gamma \vdash x \rightarrow y}$$

$$\texttt{(S)} \qquad \frac{\Delta \vdash y}{q \texttt{ said } \Delta \vdash q \texttt{ said } y}$$

$$\texttt{(P)} \qquad \frac{\Delta \vdash y}{q \texttt{ told } \Delta \vdash q \texttt{ put } y}$$

From the Liberal Datalog point of view [4], the axioms and inference rules form a program that computes a binary relation $\vdash$ of type `Set(Infon)` $\times$ `Infon`.

Explanation

Here E and I allude to elimination and introduction respectively. If $\Delta = \{x_1, \ldots, x_n\}$ then q `said` Δ is the set $\{(q \texttt{ said } x_i) : i = 1, \ldots, n\}$, and q `told` Δ is any of the 2^n sets $\{(q \texttt{ told}_i\ x_i) : i = 1, \ldots, n\}$ where each $\texttt{told}_i \in \{\texttt{said}, \texttt{put}\}$. Rule (P) can be reformulated thus:

$$\texttt{(P')} \qquad \frac{(\Delta_1 \cup \Delta_2) \vdash y}{(q \texttt{ said } \Delta_1) \cup (q \texttt{ put } \Delta_2) \ \vdash \ q \texttt{ put } y}$$

Corollary 2.1 (Transitivity)**.** *If* $\Gamma \vdash x$ *and* $\Gamma, x \vdash y$ *then* $\Gamma \vdash y$.

Proof. By ($\rightarrow$I), we have $\Gamma \vdash (x \rightarrow y)$. It remains to apply ($\rightarrow$E). □

2.2 Primal infon logic

Primal infon logic can be given by a sequent calculus that is obtained from the calculus above by replacing ($\rightarrow$I) with the combination of a weaker inference rule ($\rightarrow$IW) and a rule (Trans) reflecting Corollary 2.1.

$$\texttt{(}\rightarrow\texttt{IW)} \qquad \frac{\Gamma \vdash y}{\Gamma \vdash (x \rightarrow y)}$$

$$\texttt{(Trans)} \qquad \frac{\Gamma \vdash x, \quad \Gamma, x \vdash y}{\Gamma \vdash y}$$

In primal infon logic one derives only from the given hypotheses Γ (and from subsets of Γ). This does not apply to full infon because of the rule ($\rightarrow$I).

Lemma 2.2. *In either calculus,*
if $\quad \Gamma \vdash x_1, \ldots, \Gamma \vdash x_n,\ \ \Gamma \cup \{x, \ldots, x_n\} \vdash y$
then $\quad \Gamma \vdash y$.

Proof. Induction on n. If $n = 0$, the lemma is trivial. Suppose that $n > 0$ and the lemma has been proved for $n - 1$. Let $\Gamma' = \Gamma \cup \{x_1, \ldots, x_{n-1}\}$. Use (Premise Inflation) to derive $\Gamma' \vdash x_n$ from the penultimate premise. The last premise is $\Gamma', x_n \vdash y$. Now use (Trans) and then the induction hypothesis. □

Remark 2.1. One may want to replace rule (P) with a simpler rule $\dfrac{\Delta \vdash y}{q\ \texttt{put}\ \Delta \vdash q\ \texttt{put}\ y}$ plus an axiom $q\ \texttt{said}\ x \vdash q\ \texttt{put}\ x$. We did not do that because of the subformula property; see Theorem 3.1. For example, in our system, there is a proof of sequent

$$s = [q\ \texttt{said}\ r\ \texttt{said}\ x \ \vdash\ q\ \texttt{put}\ r\ \texttt{put}\ x].$$

that uses only subformulas of s. Any proof of s in the modified system involves additional formulas.

3 Kripke structures

3.1 Kripke structures for full infon logic

A Kripke structure for full infon logic is a non-empty quasi-order $(W, \leq)$ of worlds with two binary relations P_q and S_q for every principal q such that the following requirements are satisfied.

K1. $P_q \subseteq S_q$.

K2. If $u \leq w$ and wP_qv then uP_qv, and the same for S_q.

By induction, any formula x is assigned a cone (that is an upward closed set) $C(x)$ of worlds. If $u \in C(x)$, we say that x holds in u or that u models x, and we write $u \models x$. If x is primitive then $C(x)$ is an arbitrary cone. Further:

K3. $C(x + y) = C(x) \cap C(y)$.

K4. $C(x \to y) = \{u : C(x) \cap \{v : v \geq u\} \subseteq C(y)\}$.

K5. $C(q\ \texttt{put}\ x) = \{u :\ \{v :\ uP_qv\} \subseteq C(x)\ \}$.

K6. $C(q\ \texttt{said}\ x) = \{u :\ \{v :\ uS_qv\} \subseteq C(x)\ \}$.

K7. $C(\texttt{True}) = W$.

Clauses K3, K4 and K7 are standard in the Kripke semantics for intuitionistic logic and go back to [13].

It is easy to check, by induction on formula z, that every $C(z)$ is indeed a cone. We consider here only the case when $z = (q\ \texttt{put}\ x)$. It suffices to show that $\{v : wP_qv\} \subseteq \{v : uP_qv\}$ given that $u \leq w$. But this follows from K2.

Further let $C(\Gamma) = \bigcap_{x \in \Gamma} C(x)$; in particular $C(\emptyset) = W$. A Kripke structure K models a sequent $(\Gamma \vdash x)$ if $C(\Gamma) \subseteq C(x)$ in K. A sequent s is *valid* if every Kripke structure models s.

Remark 3.1. Alternatively one may want to require that a Kripke structure K comes with a distinguished world h, that could be called a *hub*, and define x to hold in K if it holds in the hub. Then it is natural to define that K models a sequent $(\Gamma \vdash x)$ if $K \models \Gamma$ implies $K \models x$. It is easy to see that this leads to the same notion of validity.

3.2 Kripke strictures for weak infon logic

The definition is similar to the one above except that requirement K4 is replaced with the following looser requirement.

K4W. $C(x \to y)$ is an arbitrary cone subject to constraint $C(y) \subseteq C(x \to y) \subseteq \{u :\ C(x) \cap \{v : v \geq u\} \subseteq C(y)\}$.

3.3 Soundness and completeness

Theorem 3.1. *In the case of either infon logic, full or primal, the following claims are equivalent for any sequent s.*

1. *s is provable.*
2. *s is valid.*
3. *Every finite Kripke structure models s.*
4. *There is a proof of s that uses only subformulas of s.*

Proof. We deal with both logics at once. We prove that $(1) \implies (2) \implies (3) \implies (4) \implies (1)$. Implications $(4) \implies (1)$, and $(2) \implies (3)$ are obvious.

$(1) \implies (2)$. Let K be an arbitrary Kripke structure. By induction on the given derivation of sequent s we show that $K \models s$. If s is an (x2x) axiom, the desired $C(x) \subseteq C(x)$ is obvious. If s is the (True) axiom, use K7. Note also that $C(q\ \texttt{said}\ x) \subseteq C(q\texttt{put}\ x)$. By K5 and K6, it suffices to show that, for every u, $\{v : uP_qv\} \subseteq \{v : uS_qv\}$. This is exactly K1.

Suppose that s is not an axiom. Let u be a world in the cone of the premise of s. We need to show that u models the conclusion of s. Consider the last step in the given derivation of s. Several cases arise. The case of (Premise Inflation) is obvious. The cases (+E) and (+I) are obvious as well; just use K3.

($\to$E). By the induction hypothesis (IH), u models x as well as $x \to y$. By K4W (use the second inclusion of the constraint), u models y.

($\to$I). This case is relevant for the full but not primal infon logic. By K4, it suffices to check that $v \models x$ implies $v \models y$ for all $v \geq u$. Suppose $v \models x$. By IH, $v \models y$.

(→IW). This case is relevant for primal infon logic. By IH, u models y. By K4W (use the first inclusion of the constraint), u models $x \to y$.

(Trans). This case is relevant for primal infon logic. Recall that $u \in C(\Gamma)$. By IH applied to the first premise, $u \in C(x)$. Now we apply IH to the second premise: $u \in C(y)$.

(P) and (S). These two cases are similar. We consider only (P). By the choice of u, we have $u \in C(q\ \texttt{put}\ \Delta_1) \cap C(q\ \texttt{said}\ \Delta_2)$ where $\Delta_1 \cup \Delta_2 = \Delta$. We noted above already that we always have $C(q\ \texttt{said}\ x) \subseteq C(q\ \texttt{put}\ x)$. Accordingly $u \in C(q\ \texttt{put}\ \Delta)$ which, by K5, is equivalent $v \in C(\Delta)$ for all v with uP_qv. For any such v, by IH, $v \in C(y)$. By K5, $u \in C(q\ \texttt{put}\ y)$.

(3) $\implies$ (4). We assume that (4) fails and construct a counter-model K for sequent s. Call a formula *local* if it is a subformula of s. A *local theory* is any set u of local formulas that contains every local formula x such that sequent $u \vdash x$ is provable using only local formulas.

The quasi-order of K is the set of local theories ordered by inclusion. Set uP_qv true if and only if for every formula $(q\ \texttt{put}\ x)$ or $(q\ \texttt{said}\ x)$ in u, we have $x \in v$. Set uS_qv true if and only if for every formula $q\ \texttt{said}\ x$ in u, we have $x \in v$. Requirement K1 follows. Requirement K2 is satisfied as well. We check only the P version of K2. Suppose $u \leq w$ and wP_qv. If u contains $q\ \texttt{put}\ x$ then w contains $q\ \texttt{put}\ x$ and therefore v contains x. The case of S_q is similar.

Now we define cones $C(z)$ for formulas z. If z is a primitive formula then $C(z) = \{u :\ z \in u\}$, so that $u \models z$ iff $z \in u$. It remains to defines the cones $C(z)$ for compound formulas z. In the case of full infon logic, we could have used only clauses K3–K7 for the purpose. But, for the uniformity, we choose a different route. If z is local define $C(z) = \{u : z \in u\}$. The remaining formulas z are composed from primitive formulas and local formulas by means of connectives $+, \to, q\ \texttt{put}, q\ \texttt{said}$; use clauses K3–K7 to define $C(z)$ in all these cases. In the case of full infon logic, we check that requirements K3–K7 are satisfied for the local formulas (the case of non-local formulas is clear from the definition), so that K is a legitimate Kripke structure. In the case of primal infon logic, we do the same except that K4 is replaced with K4W.

K3. By (+E) and (+I),

$$u \in C(x+y) \iff (x+y) \in u \iff$$
$$x \in u \ \wedge\ y \in u \iff u \in C(x) \cap C(y).$$

K4. First we prove that $C(x \to y) \subseteq \{u : C(x) \cap \{v : v \geq u\} \subseteq C(y)\}$. This part is relevant to both infon logics. Suppose that u contains $x \to y$. If a local theory $v \geq u$ contains x then, by (→E) with $\Gamma = v$, we have that v contains y.

Second we prove that $\{u : C(x) \cap \{v : v \geq u\} \subseteq C(y)\} \subseteq C(x \to y)$. This part is relevant only to full infon logic. Suppose that $C(x) \cap \{v : v \geq u\} \subseteq C(y)$. We claim that sequent $u, x \vdash y$ is provable. Otherwise, there is a local theory v that includes u, contains x but does not contain y which contradicts the choice of u. By (→I) with $\Gamma = u$, we have that u contains $x \to y$.

K4W. This case is relevant for primal infon logic. The right inclusion has been already proven. To prove the left inclusion, suppose that $u \in C(y)$, so that $u \vdash y$ is provable. By (→IW), $u \vdash (x \to y)$ is provable and so $u \in C(x \to y)$.

K5. First suppose that $u \in C(q\ \texttt{put}\ x)$, that is u contains $(q\ \texttt{put}\ x)$, and let v be any world with uP_qv. By the definition of uP_qv, local theory v contains x, that is $v \in C(x)$. Second suppose that $\{v : uP_qv\} \subseteq C(x)$, that is every local theory v with uP_qv contains x. Let Δ be the set of formulas y such that u contains $(q\ \texttt{put}\ y)$, and let Δ^* be the least local theory that includes Δ. By the definition of P_q in K, we have $uP_q\Delta^*$. Then Δ^* contains x. By Lemma 2.2, sequent $\Delta \vdash x$ is provable. By (P), $u \vdash (q\ \texttt{put}\ x)$ is provable.

The case of K6 is similar to that of K5, and the case of K7 is obvious. Thus K is a legitimate Kripke structure. Finally let s be $\Gamma \vdash x$, and let Γ^* be the least local theory that includes Γ. By the assumption that (4) fails, Γ^* does not contain x. Thus K is a counter-model for s.

□

4 Primal infon logic: Hilbert-type calculus

The rest of this paper is devoted to primal infon logic. We introduce a Hilbert-type calculus for the logic. Let $\texttt{told}$ with or without a subscript range over $\{\texttt{said}, \texttt{put}\}$, and let $\texttt{pref}$ with or without a subscript range over strings of the form

$$q_1\ \texttt{told}_1\ q_2\ \texttt{told}_2\ \ldots q_k\ \texttt{told}_k$$

where k may be zero. We say that $\texttt{pref}_1$ is weaker than $\texttt{pref}_2$ and write $\texttt{pref}_1 \leq \texttt{pref}_2$ if $\texttt{pref}_1$ is the result of replacing some (possibly none) occurrences of $\texttt{said}$ in $\texttt{pref}_2$ with $\texttt{put}$.

Axioms: $\texttt{pref True}$

Inference rules:

(Pref S2P) $$\frac{\texttt{pref}_2\, x}{\texttt{pref}_1\, x} \quad \text{where} \quad \texttt{pref}_1 \leq \texttt{pref}_2$$

(Pref+E) $$\frac{\texttt{pref}\,(x+y)}{\texttt{pref}\,x} \qquad \frac{\texttt{pref}\,(x+y)}{\texttt{pref}\,y}$$

(Pref+I) $$\frac{\texttt{pref}\,x \qquad \texttt{pref}\,y}{\texttt{pref}\,(x+y)}$$

(Pref→E) $$\frac{\texttt{pref}\,x \qquad \texttt{pref}\,(x \to y)}{\texttt{pref}\,y}$$

(Pref→I) $$\frac{\texttt{pref}\,y}{\texttt{pref}\,(x \to y)}$$

A derivation of a formula φ from hypotheses Γ in the Hilbert-type calculus is a chain $x_1, \ldots, x_n$ with $x_n = \varphi$ such that every x_i is a hypothesis, an axiom or the result of applying one of the rules to earlier members.

Given a formula z, we define the *components* of z as follows: (i) z is a component, and (ii) if $\texttt{pref}\,(x + y)$ is a component or if $\texttt{pref}\,(x \to y)$ is a component, then $\texttt{pref}\,x$ and $\texttt{pref}\,y$ are components.

Lemma 4.1. *If x is a component of z then p $\texttt{told}$ x is a component of p $\texttt{told}$ z.*

Proof. Induction on the definition of components. □

Components of a sequent $s = [\Gamma \vdash \varphi]$ are the components of its formulas. A formula $\texttt{pref}_1 x$ is dominated by a formula $\texttt{pref}_2 x$ if $\texttt{pref}_1 \leq \texttt{pref}_2$. A formula $\texttt{pref}_1\, x$ is *local* to sequent s if it is dominated by a component of s. A derivation $x_1, \ldots, x_n$ of φ from Γ in the Hilbert-type calculus is *local* if every formula x_i is local to sequent $\Gamma \vdash \varphi$.

Theorem 4.2. *Let C1 be the sequent calculus for primal infon logic, C2 be the Hilbert-type calculus, and s be a sequent $\Gamma \vdash \varphi$. The following are equivalent.*

1. *s is derivable in C1.*
2. *There is a derivation of s in C1 that uses only subformulas of s.*
3. *φ is derivable from Γ in C2.*
4. *There is a local derivation of φ from Γ in C2..*

Proof. (1) and (2) are equivalent by Theorem 3.1. (4) obviously implies (3). It suffices to prove that (3) implies (1) and (2) implies (4).

(3) implies (1). By induction on the given derivation of φ from Γ in C2, we prove $\Gamma \vdash \varphi$ in C1. If φ belongs to Γ, use axiom (x2x) and then repeatedly apply (Premise Inflation). If $\varphi = (\texttt{pref True})$, start with the axiom $\texttt{True}$ and repeatedly apply rules (S) or (P) to obtain the sequent $\emptyset \vdash (\texttt{pref True})$. Then repeatedly apply (Premise Inflation) to obtain $\Gamma \vdash \texttt{pref True}$. If φ is not an axiom, several cases arise according to the last step of the derivation of φ. We consider only the case when rule (Pref→E) was applied at the last step; other cases are similar. By the induction hypothesis, $\Gamma \vdash \texttt{pref}\ x$ and $\Gamma \vdash \texttt{pref}(x \to y)$ are provable. By Lemma 2.2, it suffices to prove the sequent

$$\Gamma,\ \texttt{pref}\,x,\ \texttt{pref}\,(x \to y)\ \vdash\ \texttt{pref}\,y.$$

Start with an obviously provable sequent $x, (x \to y) \vdash y$, repeatedly apply rules (S) or (P) to obtain

$$\texttt{pref}\,x,\ \texttt{pref}\,(x \to y)\ \vdash\ \texttt{pref}\,y,$$

and then repeatedly apply (Premise Inflation).

(2) implies (4). By induction on the given proof of sequent $s = [\Gamma \vdash \varphi]$ in C1 that uses only subformulas of s, we construct a local derivation of φ from Γ in C2. The cases when s is an axiom are obvious. Otherwise several cases arise according to the last step of the derivation of s. We consider here only two cases.

(→E) Suppose that rule (→E) was applied at the last step. By the induction hypotheses, we have local derivations of x and of $x \to y$ from Γ. It remains to apply rule (Pref→) with the empty $\texttt{pref}$.

(P) Suppose that rule (P′) was applied at the last step, so that $\Gamma = (q\ \texttt{said}\ \Delta_1) \cup (q\ \texttt{put}\ \Delta_2)$. By the induction hypothesis, there is a local derivation D of y from $\Delta_1 \cup \Delta_2$. Think of Δ_1, Δ_2 as sequences. Without loss of generality, $D = \langle \Delta_1, \Delta_2, T\rangle$ so that first Δ_1 formulas are listed, then Δ_2 formulas are listed, and then the remaining *tail formulas* z are listed. Each tail formula $z = \texttt{pref}_1\, z_0$ that is not an axiom is obtained via an inference rule R_z from one or two earlier formulas. And there is a component $\texttt{pref}_2\, z_0$ of sequent $[\Delta_1 \cup \Delta_2 \vdash y]$ with $\texttt{pref}_1 \leq \texttt{pref}_2$.

Construct $D' = \langle q\ \texttt{said}\ \Delta_1,\ q\ \texttt{put}\ \Delta_1,\ q\ \texttt{put}\ \Delta_2,\ q\ \texttt{put}\ T\rangle = \langle q\ \texttt{said}\ \Delta_1,\ q\ \texttt{put}\ D\rangle$ so that first the $(q\ \texttt{said}\ \Delta_1)$ formulas are listed and then the $(q\ \texttt{put}\ D)$ formulas are listed. D' is the desired local derivation of sequent $s = [(q\ \texttt{said}\ \Delta_1) \cup (q\ \texttt{put}\ \Delta_2) \vdash q\ \texttt{put}\ y]$. First we check that D' is a derivation. Rule (PrefS2P) is repeatedly used to obtain $q\ \texttt{put}\ \Delta_1$ from $q\ \texttt{said}\ \Delta_1$; and any formula $q\ \texttt{put}\ z$ with z in T that is not an axiom is obtained via rule R_z from one or two earlier formulas. Next we check that every D' formula is local to s. This is obvious for the $(q\ \texttt{said}\ \Delta_1)$. It remains to show that any formula $q\ \texttt{put}\ z$ with $z = \texttt{pref}_1 z_0 \in D$ is local to s. Since D is local, $\texttt{pref}_1 z_0$ of D is dominated by a component $\texttt{pref}_2 z_0$ of $\Delta_1 \cup \Delta_2 \vdash y$. By Lemma 4.1, $q\ \texttt{put}\ \texttt{pref}_2\, z_0$ is a component of sequent $[q\ \texttt{put}\ (\Delta_1 \cup \Delta_2) \vdash q\ \texttt{put}\ y]$ and therefore local to s. Hence $q\ \texttt{put}\ \texttt{pref}_1 z_0$ is local to s. □

5 The linear time theorem

In the rest of the paper we work with the Hilbert-type calculus for primal logic. We define the *quotation depth* of formulas. Note the unusual last clause.

$\mathrm{QD}(x) = 0$ if x is primitive.
$\mathrm{QD}(p\ \texttt{told}\ x) = 1 + \mathrm{QD}(x)$.
$\mathrm{QD}(x + y) = \max\{\mathrm{QD}(x), \mathrm{QD}(y)\}$.
$\mathrm{QD}(x \to y) = \mathrm{QD}(y)$

In applications, the quotation depth is small. For example, the authorization language SecPAL [5], the predecessor of DKAL, has no nested sayings at all; principals make pronouncements but the pronouncements contain no quotations.

The *ground multiple derivability problem* for a logic L is to compute, given ground formulas $x_1, \ldots, x_m$ (the hypotheses) and $y_1, \ldots, y_n$ (the queries), which of the queries are derivable from the hypotheses. We presume that the syntax of formulas is such that an occurrence of a subformula y in a formula x is uniquely defined by the position of the first symbol of y in x.

Theorem 5.1. *For every natural number d, there is a linear time algorithm for the case of the ground multiple derivability problem for primal infon logic where the quotation depth of the hypotheses is $\leq d$.*

Let L_d be the fragment of primal infon logic with formulas of quotation depth $\leq d$. A derivation of an L_d query from L_d hypotheses in primal infon logic may contain formulas outside of L_d. We show that, in order to prove Theorem 5.1 for a given d, it suffices to prove that $\mathrm{GMD}(L_d)$ is solvable in linear time. Call a formula x *regular* if it has no occurrences of `True`. Call formulas x, y equivalent if $x \vdash y$ and $y \vdash x$ in primal infon logic.

Lemma 5.2. *1. For any formula z there is an equivalent formula z' that is either an axiom or a regular formula.*
2. There is a linear time algorithm that computes z'.

Proof. 1. Easy induction on z. We consider only the case $z = x \to a$ in the induction step where a is an axiom. In that case z' may be a. Indeed $z \vdash a$ because a is an axiom, and $a \vdash z$ by (Pref→I).
2. Construct the parse tree for z and execute the algorithm implicit in 1. □

Lemma 5.3. *If $x_1, \ldots, x_n$ is a local derivation of a regular formula from regular hypotheses of quotation depth $\leq d$ then every x_i is regular and $\mathrm{QD}(x_i) \leq d$.*

Proof. The first claim follows from the definition of local formulas. To prove the second claim, note that the quotation depth of the conclusion of any inference rule is bounded by the quotation depth of the premise(s). □

Corollary 5.4. *In order to prove Theorem 5.1 for a given d, it suffices to prove that $\mathrm{GMD}(L_d)$ is solvable in linear time.*

We prove that every $\mathrm{GMD}(L_d)$ is solvable in linear time. Case $d = 0$ is taken up in §5.1, and case $d > 0$ is taken up in §5.2.

5.1 The case of weak intuitionistic logic

We prove that $\mathrm{GMD}(L_0)$ is solvable in linear time. L_0 is a fragment of propositional intuitionistic logic; we call it weak intuitionistic logic. In the case of L_0, local formulas are subformulas of $\Gamma \cup Q$. The calculus of § 4 simplifies in the case of L_0 as follows.

Axiom: `True`

Inference rules

$$\frac{x + y}{x} \qquad \frac{x + y}{y} \qquad \text{(+e)}$$

$$\frac{x \quad y}{x + y} \qquad \text{(+i)}$$

$$\frac{x \quad x \to y}{y} \qquad (\to\text{e})$$

$$\frac{y}{x \to y} \qquad (\to\text{i})$$

Given hypotheses Γ and queries Q, we need to decide which of the queries follow from Γ. Formulas local to $\Gamma \cup Q$ will be called simply local. By Theorem 4.2, we may restrict attention to derivations where all formulas are local. The idea is to compute all local consequences of Γ. This is obviously sufficient.

By Lemmas 5.2 and 5.3, we may assume that the hypotheses Γ and queries Q are regular. Let n be the length of the input sequence Γ, Q. If y is subformula of a hypothesis or query, the *key* $K(y)$ of y is the position of the first symbol of the first occurrence of y in the input sequence.

Parse tree Run a parser on the input string producing a parse tree. The subtrees of the hypotheses and the queries hang directly under the root. Each node of the parse tree has a label that is or represents (according to the lexical analyzer) a connective, attribute name or function name; constants are treated as nullary function names. The label length is $O(\log(n))$ due to the lexical analysis phase of parsing. Extra tags mark hypotheses and query; such a tag is not a part of the official label.

By induction define the *locutions* $L(u)$ of nodes u. If u is a node with children $u_1, \ldots, u_k$, then

$$L(u) = \mathrm{Label}(u)(L(u_1), \ldots, L(u_k)).$$

The locutions are being introduced for use in our analysis of the algorithm; the algorithm will not have the time to produce all the locutions.

Each node u is associated with a particular occurrence of $L(u)$ in the input string. The initial position of $L(u)$ in the input string is the *key* $K(u)$ of u. Nodes u and v are *homonyms* if $L(u) = L(v)$. A node with the least key in its homonymy class is a *homonymy original*. A homonymy original u represents the locution $L(u)$; its key is the key of the locution. We presume that the nodes u come with *homonymy pointers* $H(u)$ initially set to nil.

Homonymy originals Run the Cai-Paige algorithm [6] on the parse tree. The algorithm partitions the (keys of) nodes u into buckets B_ℓ according to the height ℓ of u, that is the height (or depth) of the subtree rooted at u. Furthermore, every bucket is ordered according to the lexicographic order of locutions $L(u)$. Note that two homonyms have the same height and thus belong to the same bucket. Homonyms are ordered according their keys. The Cai-Paige algorithm sets every homonymy pointer $H(u)$ to the homonymy original of u. This wonderful algorithm runs in linear time.

In the rest of the proof, we are interested primarily in the formula nodes u, where $L(u)$ is a formula. By default, a node will be a formula node.

Preprocessing Create a table T of records indexed by the homonymy originals u such that $L(u)$ is a formula. A record $T(u)$ has five fields. One of them is the *status field* $S(u)$ with values in the set $\{1, 2, 3\}$. The status field is dynamic; its value may change as our algorithm runs. All other fields are static; once created their values remain immutable.

The status field $S(u)$ determines the current status of the formula $L(u)$. Formulas of status $1, 2, 3$ will be called *raw, pending, processed* respectively. A raw formula has not been derived. A pending formula has been derived but remains a subject of some processing. A processed formula has been derived and processed. Initially every $S(u) = 1$. Traverse the parse tree setting the status $S(u)$ to 2 if $L(u)$ is an axiom or else u is tagged as a hypothesis.

The static fields of the record $T(u)$ are $(+, \text{left})$, $(+, \text{right})$, $(\rightarrow, \text{left})$ and $(\rightarrow, \text{right})$. Each entry in these fields is a sequence of (the keys of) formula nodes. To compute those sequences, traverse the parse tree in the depth-first manner paying attention only to formula nodes v. If v is the left child of a node v' with label "+" then append $H(v')$ to the $(+, \text{left})$ sequence of $H(v)$ and move on. The cases where v is the right child of v' or the label of v' is $\rightarrow$ or both are treated similarly.

Processing Walk through the table T and process every pending formula $L(u)$ in turn. When the processing of $L(u)$ is finished do the following.

- Set $S(u) = 3$ indicating that $L(u)$ is processed.
- If u is tagged as a query then output $K(u)$.

The processing consists of firing, one after another, the inference rules applicable to $L(u)$.

Rule (+e) requires that $L(u)$ has the form $x + y$. If any of the formulas x, y is raw, make it pending. More exactly, let u_1, u_2 be the left and right children of u. If $S(H(u_i)) = 1$, set $S(H(u_i)) = 2$.

Rule (+i) may be applied in two ways depending on whether we view $L(u)$ as the left or right premise of the rule. The two cases — call them the left and right cases — are similar; we describe here only the left case where $L(u)$ is the left summand x of a formula $x + y$. For any such y that has been derived but $x + y$ is still raw, make $x + y$ pending. More exactly, for every v in the $(+, \text{left})$ sequence of $T(u)$ do the following. Note that $L(v) = L(u) + L(w)$ where w is the right child of v. If $S(H(w)) > 1$ and $S(v) = 1$, set $S(v) = 2$.

Rule ($\rightarrow$e) also may be applied in two ways depending on whether we view $L(u)$ as the left or right premise of the rule. This time around the two cases — call them the left and right cases — are quite different. The left case is similar to the left case in the application of rule (+i) above. For every v in the $(\rightarrow, \text{left})$ sequence of $T(u)$ do the following. Note that $L(v) = L(u) \rightarrow L(w)$ where w is the right child of v. If $S(v) > 1$ and $S(H(w)) = 1$, set $S(H(w)) = 2$. The right case requires that $L(u)$ has the form $x \rightarrow y$. If x has been derived and y is raw, make y pending. More exactly, let u_1, u_2 be the left and right children of u. If $S(H(u_1)) > 1$ and $S(H(u_2)) = 1$, set $S(H(u_2)) = 2$.

Rule ($\rightarrow$i) has only one premise $y = L(u)$. Any raw formula of the form $x \rightarrow y$ becomes pending. More exactly, for every v in the $(\rightarrow, \text{right})$ sequence of $T(u)$ do the following. Note that $L(v) = L(w) \rightarrow L(u)$ where w is the homonymy original of the left child of v. If $S(v) = 1$, set $S(v) = 2$.

This completes the algorithm.

Proof of correctness Note that every pending formula becomes processed. Obviously only provable formulas become pending. To prove the correctness of the algorithm, it suffices to prove that every provable formula $L(v)$ becomes pending. We do that by induction on the proof length of $L(v)$. If $L(v)$ is an axiom or hypothesis, it becomes pending when the table is formed. Otherwise several cases arise depending on the rule used at last step of the given proof of $L(v)$. All cases are pretty obvious. We consider just one of those cases.

Case (+i) where $L(v) = x+y$ and x, y have been derived earlier. By symmetry we may assume without loss of generality that x became pending first. Formula y is $L(u)$ for some u. But then v occurs in the (+,right) sequence of $T(u)$. Accordingly $L(v)$ becomes pending as a result of applying the right case of rule (+i) in the processing of $L(u)$.

Time complexity It remains to check that the algorithm is indeed linear time. Obviously the parse-tree and table stages are linear time. The only question is whether the processing stage is linear time. Instead we claim something

else. Note that the processing of a pending formula $L(u)$ is done in a fixed finite number of phases. At each phase, we make some number $k(u)$ of attempts to apply a particular inference rule viewing $L(u)$ as one of the premises. Each attempt takes bounded time. The number of attempts is not bounded. It suffices to prove, however, that $\sum_u k(u) = O(n)$.

The proof is similar for all phases. Here we consider only the phase when we attempt to apply rule (+e) with $L(u)$ as the left premise x. In this phase the number $k(u)$ is the length of the (+,left) sequence of u. But the (+,left) sequences for distinct records are disjoint. And so $\sum_u k(u) \leq n$.

5.2 General case

Given a positive integer d, we construct a linear time algorithm for $\mathrm{GMD}(L_d)$ by modifying the algorithm for $\mathrm{GMD}(L_0)$. As in §4, told with or without a subscript ranges over $\{\mathtt{said}, \mathtt{put}\}$, and pref with or without a subscript ranges over quotation prefixes $q_1\ \mathtt{told}_1\ \ldots\ q_k\ \mathtt{told}_k$ where k is the length of pref. We say that a quotation prefix is relevant if its length is $\leq d$.

Parsing Parse the input as in §5.1 except that now we have additional label p told. There is a problem, however. In §5.1, every local formula was the locution $L(u)$ of some node u of the parse tree. Now it is not necessarily the case. To remedy the situation, we graft extra nodes into the parse tree. This is not the optimal remedy but it simplifies the exposition.

By induction on nodes u, define $\mathtt{pref}(u)$. If u is the root, then $\mathtt{pref}(u)$ empty. Suppose that v is the parent of u. If the label of v is of the form $p\ \mathtt{told}$ then $\mathtt{pref}(u)$ is $\mathtt{pref}(v)$ appended with the label of v; otherwise $\mathtt{pref}(u) = \mathtt{pref}(v)$. Let $C(u)$ be the formula $\mathtt{pref}(u)\, L(u)$. It is easy to check that every $C(u)$ is a component and that every component is $C(u)$ for some u. Call node u relevant if $\mathtt{pref}(u)$ is relevant.

The *fan* $F(\mathtt{pref})$ of a prefix pref is an upward growing tree[1] of prefixes. It contains all prefixes $\mathtt{pref}' \leq \mathtt{pref}$. Further, it contains a prefix $q_1\ \mathtt{told}_1\ \ldots\ q_i\ \mathtt{told}_i$ of length i if it contains a prefix $q_1\ \mathtt{told}_1\ \ldots\ q_i\ \mathtt{told}_i\ q_{i+1}\ \mathtt{told}_{i+1}$ of length $i+1$ which is a parent of $q_1\ \mathtt{told}_1\ \ldots\ q_i\ \mathtt{told}_i$. Every node in $F(\mathtt{pref})$ has at most two parents and at most one child. If $d = 2$, then $F(\mathtt{pref})$ contains at most 7 nodes.

Traverse the relevant nodes of the parse tree in the depth-first manner and graft a fresh copy $F(u)$ of $F(\mathtt{pref}(u))$ at every formula node u. There is the one-to-one correspondence $\xi : F(u) \longrightarrow F(\mathtt{pref}(u))$. If $v \in F(u)$ and $\xi(v)$ is a prefix $q_1\ \mathtt{told}_1\ \ldots\ q_i\ \mathtt{told}_i$ of length $i > 0$ then the label of v is $q_i\ \mathtt{told}_i$ and the key of v is the pair

$$(\mathrm{Key}(u),\ \mathtt{told}_1\ \ldots\ \mathtt{told}_i)$$

The keys are ordered lexicographically. We do not distinguish between $\mathrm{Key}(u)$ and the pair $(\mathrm{Key}(u), s)$ where string s is empty. Every node u of the resulting *parse structure* has at most three parents, and the nodes $\leq u$ form a tree.

Remark 5.1. For every relevant original formula node u, the formula $\mathtt{pref}(u)\, L(u)$ is a component of $\Gamma \cup Q$. If $\mathtt{pref} \leq \mathtt{pref}(u)$ then $\mathtt{pref}\, L(u)$ is local. Every local formula is obtained this way. Thus the parse structure has a node for every local formula (and for some non-local formulas).

Homonymy originals As in §5.1, run the Cai-Paige algorithm on the parse structure and compute homonymy pointers $H(u)$.

Preprocessing Let T_1 be the table T constructed in §5.1. T_1 is a one-dimensional table of records $T_1(u)$ indexed by formula nodes u such that u is a homonymy original. Now, in addition to T_1, we construct a sparse two-dimensional table T_2. The rows of T_2 are indexed by original formula nodes u, and the columns are indexed by relevant prefixes π. If there is a node v with $L(v) = \pi\, L(u)$, then $T_2(u, \pi) = \{H(v)\}$; otherwise $T_2(u, \pi) = \emptyset$. A traversal around the parse structure suffices to fill in table T_2.

Remark 5.2. We graft nodes and then put only some of them into table T_2. This is not the most efficient way to do things. One can forgo grafting, construct table T_2 directly, and refer to the table instead of to grafted nodes in the following processing. We thought that grafting would simplify the exposition.

Remark 5.3. It is more efficient to combine preprocessing with parsing.

Processing As in §5.1, we walk through the table T and process every pending formula $L(u)$ in turn. The processing consists of firing, one after another, the inference rules applicable to $L(u)$. The case of rule

$$\frac{\mathtt{pref}_2\, x}{\mathtt{pref}_1\, x}$$

is new. Suppose that $L(u) = \mathtt{pref}_2\, x$ and let $\mathtt{pref}_1 \leq \mathtt{pref}_2$. The descendant u_0 of u with locution x has an ancestor v with locution $\mathtt{pref}_1\, x$. If $H(v)$ is raw, make it pending.

As far as the remaining rules are concerned, the situation is similar to that in §5.1. For example, consider the application of the rule

$$\frac{\mathtt{pref}\ (x + y)}{\mathtt{pref}\ x}$$

[1] Normally our trees go downward so that the root is at the top.

to a formula $L(u) = \texttt{pref}(x + y)$. Find the descendant u_0 of u with $L(u_0) = x + y$. The left child of u_0 has an ancestor v with locution $\texttt{pref}\, x$. If $H(v)$ is raw, make it pending.

For a more interesting example, consider the application of rule

$$\frac{\texttt{pref}\, x \qquad \texttt{pref}\, y}{\texttt{pref}\, (x + y)}$$

to a formula $L(u) = \texttt{pref}\, x$. Find the descendant u_0 of u with $L(u) = x$. For each v in the (+, left) field of record $T_1(H(u_0))$ do the following.

Check the entry $T_2(v, \texttt{pref})$. If it is empty, do nothing. Otherwise let w be the node in the entry. The locution of v is $x + y$, and the locution of w is $\texttt{pref}\, (x + y)$. Let w_0 be the descendent of w with locution $x + y$. The right child of w_0 has an ancestor w' with locution $L(w') = \texttt{pref}\, y$. If the status of $H(w')$ is > 1, so that $\texttt{pref}\, y$ has been proved, but the status of w is 1, so that $\texttt{pref}\, (x + y)$ has not been proved, then set the status of w to 2; otherwise do nothing.

Correctness and time complexity The proof of the correctness of the algorithm and the analysis of time complexity of §5.1 survive with minor changes.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, 1974.

[2] Arnon Avron, "Tonk — A Full Mathematical Solution," To appear in Ruth Manor's Festschrift.

[3] Arnon Avron and Ori Lahav, "Canonical Constructive Systems," submitted to TABLEAUX 2009.

[4] Andreas Blass and Yuri Gurevich, "Two Forms of One Useful Logic: Existential Fixed Point Logic and Liberal Datalog," *Bulletin of the European Association for Theoretical Computer Science* 95 (June 2008), 164–182.

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

[6] Jiazhen Cai and Robert Paige, "Using multiset discrimination to solve language processing problems without hashing," *Theoretical Computer Science* 145:(1-2), 189–228, July 1995.

[7] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, *Algorithms,* MIT Press, 1990.

[8] Ronald Fagin, Joseph Y. Halpern, Yoram Moses and Moshe Y. Vardi, *Reasoning about Knowledge,* MIT Press, 1995.

[9] Yuri Gurevich and Itay Neeman, "DKAL: Distributed-Knowledge Authorization Language," 21st IEEE Computer Security Foundations Symposium (CSF 2008), 149–162.

[10] Yuri Gurevich and Arnab Roy, "Operational Semantics for DKAL: Application and Analysis," Microsoft Research Tech Report MSR-TR-2008-184, December 2008.

[11] Yuri Gurevich and Itay Neeman, "DKAL 2 — A Simplified and Improved Authorization Language," Microsoft Research Tech. Report, Feb. 2009.

[12] Stephen Cole Kleene, "Introduction to Metamathematics," D. Van Nostrand Company 1952.

[13] Saul Kripke, "Semantical Analysis of Intuitionistic Logic I," in *Formal Systems and Recursive Functions*, eds. J. W. Addison, L. Henkin and A. Tarski, North-Holland 1965, 92–130.

[14] John-Jules Ch. Meyer and Wiebe van der Hoek, *Epistemic Logic for AI and Computer Science*, Cambridge University Press 1995.

[15] Grigori Mints, *A Short Introduction to Intuitionistic Logic,* Kluwer Academic / Plenum Publishers 2000.

[16] Robert Paige and Robert E. Tarjan, "Three partition refinement algorithms," *SIAM Journal of Computing* 16:6, 973–989, December 1987.

[17] Richard Statman, "Intuitionistic Propositional Logic is Polynomial-Space Complete," *Theoretical Computer Science* 9:1 (July 1979), 67–72.