Annals of Pure and Applied Logic 165 (2014) 164–198

Contents lists available at SciVerse ScienceDirect

Annals of Pure and Applied Logic

www.elsevier.com/locate/apal

Intuitionistic completeness of first-order logic

Robert Constable *, Mark Bickford

Cornell University, Department of Computer Science, Ithaca, NY 14853, United States

ARTICLE INFO

Article history:
Available online 19 August 2013

MSC:
03
06
18
68

Keywords:
BHK semantics
Completeness
Constructive type theory
Evidence semantics
Intersection type
Intuitionistic logic

ABSTRACT

We constructively prove completeness for *intuitionistic first-order logic, iFOL*, showing that a formula is provable in *iFOL* if and only if it is *uniformly valid* in intuitionistic evidence semantics as defined in intuitionistic type theory extended with an intersection operator.

Our completeness proof provides an effective procedure that converts any uniform evidence into a formal *iFOL* proof. Uniform evidence can involve arbitrary concepts from type theory such as ordinals, topological structures, algebras and so forth. We have implemented that procedure in the Nuprl proof assistant.

Our result demonstrates the value of uniform validity as a semantic notion for studying logical theories, and it provides new techniques for showing that formulas are not intuitionistically provable. Here we demonstrate its value for minimal and intuitionistic first-order logic.

1. Introduction

We introduce the concept of *uniform validity* for minimal and intuitionistic first-order logic, *mFOL* and *iFOL*, and show that it opens a new approach to completeness questions for constructive logics. We use the idea to prove constructive completeness theorems with respect to *uniform evidence semantics* for both minimal and intuitionistic first-order logic.

1.1. Main results

Our main results are these:

One. A formula of intuitionistic first-order logic is provable if and only if its Friedman embedding into minimal logic is *uniformly valid* under minimal logic evidence semantics.

* Corresponding author.
E-mail address: rc@cs.cornell.edu (R. Constable).

0168-0072/$ – see front matter
http://dx.doi.org/10.1016/j.apal.2013.07.009

We use the term *evidence semantics* for a particular version of the propositions-as-types principle (also called the Curry–Howard isomorphism or proofs-as-programs identification) as it is expressed in constructive type theories. Furthermore, we show that the Friedman transformation can be explained as a proof theoretic means of expressing *intuitionistic evidence semantics* in which *False* is the type of an *abort operator* or the empty type. From this viewpoint, our main result can be stated as:

Two. A formula of intuitionistic first-order logic is provable if and only if it is *uniformly valid* under *intuitionistic evidence semantics*.

We believe that a plausible case can be made that evidence semantics captures the informal notion referred to as the *Brouwer Heyting Kolmogorov* (*BHK*) *semantics* for minimal logics. Furthermore, when this semantics is extended by Veldman's observations about how to treat *False*, it can be used to show completeness for *iFOL*. If this belief becomes widely shared, then our result could be stated this way:

Three. A formula of *iFOL* is provable if and only if it is *uniformly valid* under its intended *Brouwer Heyting Kolmogorov Veldman* (*BHKV*) semantics in which *False* is a type which can either be empty or the type of an *abort operator* in computations and constructions.

Evidence semantics for minimal logic as expressed in intuitionistic type theory [57,20,58,63,4,28,78,70,71] appears to be a precise formal version of the BHK semantics for minimal logic. For minimal logic, evidence semantics also corresponds to other well studied notions of semantics for constructive logics such as the propositions-as-types principle and the Curry–Howard isomorphism. More on this just below.

It is not so clear that intuitionistic evidence semantics corresponds to BHK semantics for intuitionistic logic nor to the propositions-as-types interpretation of Martin-Löf's *Intuitionistic Type Theory* (*ITT*) [57,58]. For example in propositions-as-types semantics, *False* is interpreted as the empty type *Void*. In intuitionistic evidence semantics we propose here that *False* can be interpreted either as the type *Void* or as the type of an *abort* operator, {*abort*} and thus is non-empty. We don't always know which case applies. Also, the *abort* object is not regarded as canonical evidence and is thus never an input value in a typed program.

It is possible that the accepted semantics for intuitionistic logics will eventually involve the computational interpretation of *False* and negation that we propose here – our modern interpretation of remarks in Brouwer's writings.[1] This outcome is plausible since a closely related idea for treating *False* has been advocated by Veldman and de Swart [83,75] in the context of Kripke models.

Brouwer's development of intuitionistic mathematics [16,38] involved a redefinition of the basic logical operators. His account of implication, disjunction, and quantification are easily understood, and they were formalized in logical calculi by Heyting [35–37] and Kolmogorov [48,47]. This semantics is now the standard semantics for minimal logic, the core of intuitionistic logic. We call this *minimal logic evidence semantics*. Brouwer investigated several interpretations of negation, and Van Atten [3] page 25 claims that Brouwer never used *ex falso quodlibet* in his intuitionistic work. In van Stigt's account of negation [74], he discusses several approaches that Brouwer considered including one reported in Section 5.8 in which he says that falsity means that the "construction does not go through" – in Dutch "gaat niet." The idea developed here is very close to this meaning, and it gives rise to *intuitionistic evidence semantics*.

We don't want the issue of BHK semantics to be distracting and the issue of *False* to dominate our account, so we state and prove first a more conventional formulation of our results, version **One** above. The first version uses Friedman's elegant embedding of *iFOL* into *mFOL* and avoids a semantics for *False*. Leivant's account [53] of Friedman's embedding for Heyting Arithmetic is especially simple in that it applies

[1] Brouwer proposed several interpretations of negation, see [74]. According to Van Atten [3], Brouwer never used the *ex falso quodlibet* rule in his intuitionistic writings.

only to atomic formulas. His version makes the idea for our first result quite clear, although one needs a slight variant for our pure *iFOL*.

Our second result depends on a different semantics for *False* which we describe after dealing with the minimal logic core. The third version stated above is not treated in more detail in this article, depending as it does on a philosophical and scholarly analysis of the history of intuitionism. One could claim that this third version would provide the kind of result that Beth sought 66 years ago. In due course it might become interpreted in that way.

1.2. Evidence semantics

Evidence semantics for minimal logic is a version of the propositions-as-types semantics (a.k.a. Curry–Howard *isomorphism*, proofs-as-programs *identification*, type theoretic *realizability*) widely used in computer science as a practical semantics for programming languages and their associated programming logics, see [73].[2]

Evidence semantics for constructive logics, in all of its versions, is quite different from Beth and Kripke semantics, for which there have been intuitionistic completeness theorems for many years [83,75]. Our uniform evidence semantics is a new logical notion that is a natural extension of the propositions-as-types semantics and which we believe will also be practically useful. Evidence semantics with an abort operator is an addition we propose in this article for turning the Friedman translation into a semantic notion suitable for first-order logic.

Some scholars believe that the propositions-as-types semantics is very close to semantic ideas formulated by Brouwer, Heyting and Kolmogorov, called BHK semantics.[3] On the other hand, some logicians think of this as an informal notion, not precise enough for a semantic theorem. In contrast, Kleene's precise notions of *recursive realizability* could in principle be related exactly to evidence semantics and to the propositions-as-types semantics by developing the notion of *type theoretic realizability*.

We do not consider the relationship between BHK, evidence semantics, and realizability in detail, but we feel comfortable using the term *type theoretic realizability* as another precise concept in our discussion, clearly related to evidence semantics but treating *False* as an empty type. Moreover, we do not try to precisely characterize the connection between evidence semantics, realizability semantics and BHK semantics because apparently additional historical work remains to be done to understand more exactly what Brouwer meant in his writings.[4]

We do not rely on Church's Thesis for any of these results, and according to Kleene [46,79], our use of the Fan Theorem precludes it.[5]

1.3. Intuitionistic model theory

This article contributes to an *intuitionistic model theory* as proposed by Beth in 1947 [10] and advanced by Per Martin-Löf [57,58]. Beth's ideas led to *Beth models* and *Kripke models* whose computational meaning is not as clear as in the realizability tradition, even given Veldman and de Swart's intuitionistic completeness

[2] This widely used semantics in computer science is based on the work of a large number of logicians, computer scientists, and mathematicians. Several background articles will be explicitly referenced including [44,25,14,52,56,18,40,57,19,28,73].

[3] See Troelstra [81] where he states on page 12 "The standard informal interpretation of logical operators in intuitionistic logic is the so-called proof-interpretation or Brouwer–Heyting–Kolmogorov interpretation (BHK-interpretation for short)."

[4] We think there is a good case equating BHK with propositions-as-types and to type theoretic realizability, but others disagree, even about Kleene's motivations. In relating our work to Kleene's 1945 notion of realizability, we came to see that Kleene altered his motivation for recursive realizability from seeing it as exploring Hilbert's notion of incomplete communication to exploring Brouwer's informal computational semantics. Perhaps not everyone agrees. By 1965 Kleene definitely interpreted his earlier work as an investigation of Brouwer's ideas.

[5] The Computational Type Theory which Nuprl implements was designed in 1984 to use an *open-ended* notion of effective computability from the start [20].

theorems for Kripke models [83,75]. We work closer to a realizability tradition expressed in type theory but directly inspired by Kleene [44,46]; this motivation is explicit in [18]. Other researchers were informed by observations in Curry and Feys [25], and Howard's unpublished notes from 1968 [40] inspired Martin-Löf's semantics for his *Intuitionistic Type Theory* (*ITT*) [57,58,63] and his more recent *Intensional-ITT* [15,59,63].

The Martin-Löf approach was extended and implemented by the PRL Group using *Constructive Type Theory* (*CTT*) automated by the Nuprl proof assistant as reported in the book *Implementing Mathematics with the Nuprl Proof Development System* [20]. A similar approach was taken by the Coq Group using the *Calculus of Constructions* (*CoC*) [21] automated by the Coq proof assistant, and later modified to the *Calculus of Inductive Constructions* [22] which is much closer to Martin-Löf's theory [63] and is described in [9]. This was also done by the Gothenberg Group reported in the book *Programming in Martin-Löf's Type Theory* [63,15], and then by the Minlog Group as reported in Proof Theory at Work: Program Development in the Minlog System [8]. The propositions-as-types approach was used in *Logical Frameworks* such as Edinburgh LCF [34] and Twelf [64] and MetaPRL [39]. These logics and systems are studied in numerous doctoral dissertations and articles many of which are cited in [1]. This is the tradition and context framing and motivating our completeness results.

All of these logics are implemented by *proof assistants* such as Agda, Coq, MetaPRL, Minlog, Nuprl, and Twelf among others. These implemented logics play a role in the development of *correct-by-construction software* [30,18] as well as in logic and mathematics.

1.4. Computational semantics

The semantic tradition we rely on is grounded in precise knowledge of an underlying computation system and its efficient implementation in programming languages. The computation system comes first and is essential to the semantics for the logic because computation rules define the *canonical forms of evidence* for typing judgments. There is an extensive literature on this subject that is tied very strongly to logic and comes closest to our results (posted in the arXiv in 2011 from technical reports in 2010). A recent article by Ilik [43] illustrates the strong connections to completeness issues.

Our computational semantics of evidence terms follows the method of *structured operational semantics* of Plotkin [67,68] and depends on the seminal work of Douglas Howe on computational equality [41,42]. The few basic results about programming language semantics we mention can be found in textbooks on the subject [78,33,62,65]. Many results from this theory are now being formalized in proof assistants and applied directly to building better languages and systems [66].

1.5. Logical background

It is widely agreed that Tarski's semantics [77] for classical first-order formulas faithfully captures their intuitive *truth-value based interpretation*. Gödel's classical completeness theorem for first-order logic is nowadays presented with respect to this Tarskian semantics, showing that a FOL formula is provable if and only if it has the value "true" in all Tarski structures. This has become a fundamental result in logic which is widely taught to undergraduates using many excellent textbook proofs, such as Smullyan's enduring *First-Order Logic* [72].[6]

According to Troelstra [81], the BHK semantics for *iFOL* is the "intended semantics", faithful to an intuitionistic conception of knowledge. In retrospect we can see a strong connection between BHK semantics and propositions-as-types/evidence semantics. In contrast to the classical situation, there has been no

[6] For a direct connection of our methods to other results in Smullyan's book see [13].

intuitionistic completeness proof with respect to this semantics. We mention some of the excellent attempts later in this section.

Brouwer's semantics may still be more widely known for rejecting the *Law of Excluded Middle* (LEM) than for its connection to computation. Using uniform evidence, we will see a unique computational property of LEM that distinguishes it from all of the other standard first-order axioms. Namely, LEM is a basic classical law of logic that is not uniformly valid. Indeed it is the only non-uniformly valid law in most classical axiomatizations. The non-uniformity property also holds for the law of double negation elimination (DNE), $\sim\sim P \Rightarrow P$, favored by Kolmogorov as the distinguishing feature of classical logic. Indeed, non-uniformity holds for any law that implies LEM.

Brouwer relied on an informal model of mental constructions in terms of which he could interpret most concepts and theorems of logic and mathematics, including many concepts from set theory (see [74]). Brouwer's approach avoided a precise account of computation that Church, Turing, and now legions of computer scientists use to give meaning to mathematical statements whose meaning is grounded in computations executed by modern digital computers. Brouwer's intuitive understanding of logical operators as interpreted in the formal systems of Heyting [35] and Kolmogorov [48] has come to be known among logicians as *Brouwer Heyting Kolmogorov* (*BHK*) *semantics*. In 1945 Kleene [44,79] invented his *realizability* semantics for intuitionistic number theory in order to connect Brouwer's informal notion of computability to the precise theory of general recursive functions. Kleene used indexes of general recursive functions as *realizers*, and by 1952 [45] he viewed realizability as a formal account of BHK semantics under the assumption of Church's Thesis.

By 1982 Martin-Löf [57,58] building on the work of Kleene [46] and Tait [76] and Howard [40] refined the informal BHK approach and raised it to the level of a *semantic method* for constructive logics grounded in operational semantics [68]. Already in 1970 Martin-Löf proposed using Brouwer's analysis of bar induction as the meaning of Π^1_1 statements and developed a constructive version of completeness for classical first-order logic [55] based on a constructive topological model of Borel sets in the Cantor Space.

1.6. Previous completeness theorems

Over the last fifty years there have been numerous deep and evocative efforts to formulate completeness theorems for the intuitionistic propositional calculus and for intuitionistic first-order logic modeled after Gödel's Theorem [75,26,49,55,83,61]. Some efforts led to apparently more technically tractable semantic alternatives to BHK such as *Beth models* [11,83,75], *Kripke models* [51], topological models [23,32,77,69,55], intuitionistic model theoretic validity [80], and provability logic [2]. Dummett [26] discusses completeness issues extensively.

The value of developing precise mathematical semantics for intuitionistic mathematics in the spirit of Tarski's work dates at least from Beth 1947 [10] with technical progress by 1957 [11]. While completeness questions did not drive this exploration of various forms of constructive semantics, nevertheless, each new semantics was tested by the challenge of finding a completeness proof in addition to explaining with increasing precision the differences between classical and constructive notions of knowledge.

So the completeness issue *a la BHK* has been identified as important for sixty six years. A very significant early attempt to base completeness on BHK is the (nonconstructive) work of Läuchli [52,54] who stressed the notion of *uniformity* as important. None of these efforts provides a constructive completeness theorem faithful to BHK semantics (a.k.a. Brouwer realizability) either for the intuitionistic propositional calculus (IPC) or for the full predicate calculus.

The closest correspondingly faithful constructive completeness theorem for *intuitionistic validity* is by Friedman in 1975 (presented in [80]), and the closest classical proof for the Brouwer Heyting Kolmogorov (propositions-as-types/proofs-as-terms/proofs-as-programs) semantics for intuitionistic first-order logic may be from 1998 by Artemov using *provability logic* [2]. Results suggest how delicate completeness theorems

are since constructive completeness with respect to full intuitionistic validity contradicts *Church's Thesis* [49,80] and implies *Markov's Principle* as well [60,61].

1.7. Constructive type theory with an intersection operator

We first informally discuss *evidence semantics* for minimal logic.[7] Using evidence semantics, we introduce the idea of *uniform validity*, a concept central to our results and one that is also classically meaningful.

This concept provides an effective tool for constructive semantics because we can establish uniform validity by *exhibiting even one polymorphic object* among a possibly unbounded number of them that we might find. For example, the propositional formula $A \Rightarrow A$ is uniformly valid exactly because there is an object in the intersection of the family of all evidence types for this formula indexed by each possible choice of proposition A among the *type of propositions*, $\mathbb{P}$.

We write this intersection as $\forall[A : \mathbb{P}].A \Rightarrow A$ or as $\bigcap A : \mathbb{P}.A \Rightarrow A$.[8] In this case, given the extensional equality of functions, the polymorphic identity function $\lambda(x.x)$ is the one and only object in the intersection. So the witness for uniform validity, like the witness for provability, can be provided by a single object. Truth tables provide a single (expensive) piece of evidence for classical propositional logic. There are single witnesses for the validity of all uniformly valid first-order formulas. For example, it will be clear after we provide the evidence semantics that the polymorphic term $\lambda(h.\lambda(x.\lambda(p.h(\langle x, p\rangle))))$ establishes the uniform minimal (logic) validity of

$$\sim \exists x.P(x) \Rightarrow \forall x.\big(\sim P(x)\big)$$

hence the uniform intuitionistic and classical validity as well.

Another important observation about uniform validity is that *the formulas of first-order logic that are provable intuitionistically and minimally are uniformly valid.* It is also noteworthy that *the law of excluded middle is not uniformly valid in either constructive or classical evidence semantics.*

The meaning of *False* also raises the semantic issue that leads us to first consider minimal logic and the Friedman embedding of *iFOL* into *mFOL*. Consider the intuitionistically valid assertion $False \Rightarrow A$ for any proposition A. One type theory witness for uniform validity is $\lambda(x.x)$, and other witnesses include any constant function, say $\lambda(x.17)$. If we designate a *diverging* term such as *div*, then $\lambda(x.div)$ is also evidence because the claim being made is that if x belongs to the evidence type for *False*, then x or 17 or *div* belongs to the evidence type for A. So according to the informal semantics of intuitionistic logic, the claim $False \Rightarrow A$ is "vacuously true" since no element can be evidence for *False* whose standard evidence is the empty type.

From the constant function with an arbitrary evidence term *evd*, $\lambda(x.evd)$, we cannot reconstruct the proof of $False \Rightarrow A$. The term *evd* might be entirely misleading. The evidence in the constructive metatheory, *CTT*, for $False \Rightarrow A$ provided by the *CTT* proof is $\lambda(x.any(x))$. This evidence suggests a way to provide an alternative semantics for *False*, and thus for *iFOL*, that avoids the issue just discussed and avoids the need for minimal logic in our account. On the other hand, the minimal logic approach is extremely simple, and it is well known and well studied. So we use that method first and then point out how to avoid it.

We are using the new semantics of *False* and *iFOL* in our Nuprl proof, and we will account for it much more fully in a future article about the formalization of our proof in *CTT*.

In minimal logic, there is no atomic propositional constant *False*. Instead the *arbitrary* propositional constant $\bot$ is used, and its interpretation allows non-empty types as well as empty ones. For the same

[7] We can extend this semantics to classical logic if oracle computations are allowed to justify the law of excluded middle, $P \vee \sim P$, with an operator $magic(P)$ [19]. We make some observations about classical logic based on this *classical evidence semantics*.

[8] We work in a *predicative* metatheory, therefore the type of all propositions is stratified into orders or levels, written $\mathbb{P}_i$. For these results we can ignore the level of the type or just write $\mathbb{P}$.

reason, avoiding vacuous hypotheses, we require that all domains of discourse for minimal logic can be non-empty.

1.8. Comparison to intuitionistic validity

Results of McCarty [60,61] demonstrate that unless one changes in significant ways what one means by completeness (or by validity) or otherwise limits the collection of formulas at issue, then a completeness theorem will be impossible to prove intuitionistically. We have opted to change the notion of validity, not by preconceived choice, but by a discovery.

We discovered that provability is captured exactly by *uniform validity*, an intuitively smaller collection of formulas than those constructively valid. Nevertheless, uniform validity is extremely useful in practice when thinking about purely logical formulas precisely because it corresponds exactly to proof and yet is an entirely semantic notion based on evidence semantics, the semantics that enables strong connections to computer science.

1.9. Counterexamples

Soundness with respect to uniform validity provides a simple method of showing that formulas are not provable by showing that they are not uniformly valid. For example, it is trivial to show that $P \vee \sim P$ is not uniformly valid, e.g. to show $\sim \forall[P : Prop].P \vee \sim P$. Suppose there were a uniform realizer, d. It would have to be an element of the disjoint union type, thus either $inr(\star)$ or $inl(\star)$. If it is an inr term, then pick P to be a true proposition, say $True$, and otherwise pick it to be $False$. These choices show that there can be no such uniform d. Once we have this easy result, we can show that $\sim\sim P \Rightarrow P$ is also not uniformly valid, e.g. we show $\sim \forall[P : Prop]. \sim\sim P \Rightarrow P$. We do this by assuming that $\forall[P : Prop]. \sim \sim P \Rightarrow P$ and using the fact that for any P we can prove $\sim\sim (P \vee \sim P)$, thus if we could prove the uniform statement, we could also prove $\forall[P : Prop].P \vee \sim P$ which we just showed is not uniformly true. By the same technique we can show that Pierce's law is not uniformly valid, e.g. $\sim \forall[P, Q : Prop].(((P \Rightarrow Q) \Rightarrow P) \Rightarrow P)$. All the formulas that intuitionistically imply $P \vee \sim P$ are provably false by this method.

We can also show that first-order Markov's Principle (MP) is not uniformly valid; that is $(\forall x.(P(x) \vee \sim P(x))\& \sim \forall x.\sim P(x)) \Rightarrow \exists y.P(y)$ is not uniformly valid. This is because we can choose a two element domain D with elements a and b and consider two predicates, P_1 and P_2 which have opposite values on a and b. The existential quantifier must pick one of a or b, but it will be an incorrect choice for one of the predicates.

2. Proof rules and proof expressions

2.1. Proof expressions

We assign denotational meaning to proofs, according to the "proofs-as-terms" principle (PAT). The rules include constraints on the subexpressions of a proof. This is especially natural in *refinement style logics* studied by Bates [6] and Griffin [31] and used in *CTT* and *tableaux systems* [12,72,27].

For each rule we provide a name that is the *outermost operator* of a proof expression with slots to be filled in as the refinement style proof is developed. The partial proofs are organized as a tree generated in two passes. The first pass is top down, driven by the user creating terms with slots to be filled in on the algorithmic bottom up pass once the downward pass is complete. Here is a simple proof of the intuitionistic tautology $A \Rightarrow (B \Rightarrow A)$.

$$\vdash A \Rightarrow (B \Rightarrow A) \text{ by } \lambda\big(x.slot_1(x)\big)$$

$$x : A \vdash (B \Rightarrow A) \ by \ slot_1(x)$$

In the next step, $slot_1(x)$ is replaced at the leaf of the tree by $\lambda(y.slot_2(x, y))$ to give:

$$\begin{aligned} &\vdash A \Rightarrow (B \Rightarrow A) \text{ by } \lambda\big(x.slot_1(x)\big) \\ &\quad x : A \vdash (B \Rightarrow A) \text{ by } \lambda\big(y.slot_2(x, y)\big) \text{ for } slot_1(x) \\ &\quad x : A, y : B \vdash A \ by \ slot_2(x, y) \end{aligned}$$

When the proof is complete, we see the slots filled in at each inference step as in:

$$\begin{aligned} &\vdash A \Rightarrow (B \Rightarrow A) \text{ by } \lambda\big(x.\lambda(y.x)\big) \\ &\quad x : A \vdash (B \Rightarrow A) \text{ by } \lambda(y.x) \\ &\quad x : A, y : B \vdash A \ by \ x \end{aligned}$$

We present the rules in a *top down style*, as in tableaux, and as in the style of the *Edinburgh LCF* tactic system [29] and the Nuprl Proof Development System [20]. Here we show the *construction rules* first, often called the *introduction rules* because they introduce the canonical proof terms or called the *right-hand side* rules in the sequent calculus because they apply to terms on the right-hand side of the turnstile. So typical names seen in the literature are these: for & we say *AndIntro* or *AndR*; for $\Rightarrow$ we say *ImpIntro* or *ImpR*; for $\vee$ we say *OrIn-r* or *OrIn-l*, and for $\forall x$ we say *AllIntro* or *AllR*, and for $\exists$ we say *ExistsIntro* or *ExistsR*.

For each of these construction rules, the constructor needs subterms which build the component pieces of evidence. Thus for *AndR* the full term will have slots for the two pieces of evidence needed, the form will be $AndR(slot1, slot2)$ where the slots are filled in as the proof tree is expanded. When the object to be filled in depends on a new hypothesis to be added to the left-hand side of the turnstile, the rule name supplies a unique label for the new hypothesis, so we see a rule name like $ImpR(x.slot(x))$ or $AllR(x.slot(x))$. In the case of the rule for $\exists$, there is a subtlety. The rule name provides two slots, but the second depends on the object built for the first, so we see rule names such as $ExistsR(a; slot(a))$.

For each connective and operator we also have rules for their occurrence on the left of the turnstile. These are the *rules for decomposing* or using or *eliminating* a connective or operator. They tell us how to use the evidence that was built with the corresponding construction rules, and the formula being decomposed is always named by a label in the list of hypotheses, so there is a variable associated with each rule application. Here are typical names: for & we say $AndElim(x)$ or $AndL(x)$. However, there must be more to this rule name because typically new formulas are added to the hypothesis list, one for each of the conjuncts, so we need to provide labels for these formulas. Thus the form of elimination for & is actually $AndL(x; l, r.slot(l, r))$ where l stands for the left conjunct and r for the right one.

Rule names such as *AndR*, *AndL*, OrR_l, OrR_r, *OrL*, and so forth are suggestive in terms of the details of the proof system, but they are not suggestive of the structure of the evidence, the semantics. We will use rule names that define the computational forms of evidence. The *evaluation rules* for these proof terms are given as in *ITT* or *CTT*, for instance in the book *Implementing Mathematics* [20] or in the Nuprl Reference Manual [50].

So instead of $AndR(a; b)$ where a and b are the subterms built by a completed proof by progressively filling in open slots, we use $pair(a; b)$ or even more succinctly $\langle a, b\rangle$, and for the corresponding decomposition rule we use $spread(x; l, r.t(l, r))$ where the binding variables l, r have a scope that is the subterm $t(l, r)$. This term is a compromise between using more familiar operators for decomposing a pair p such as $first(p)$ and $second(p)$ or $p.1$ and $p.2$ with the usual meanings, e.g., $first(\langle a, b\rangle) = \langle a, b\rangle.1 = a$. The reason to use *spread* is that we need to indicate how the subformulas of $A\&B$ will be named in the hypothesis list.

The decomposition rules for $A \Rightarrow B$ and $\forall x.B(x)$ are the most difficult to motivate and use intuitively. Since the evidence for $A \Rightarrow B$ is a function $\lambda(x.b(x))$, a reader might expect to see a decomposition rule name such as $apply(f;a)$ or abbreviated to $ap(f;a)$. However, a Gentzen sequent-style proof rule for decomposing an implication has this form:

$H, f : A \Rightarrow B, H' \vdash G$ by *ImpL* on f

1. $H, f : A \Rightarrow B,\ H' \vdash A$
2. $H, f : A \Rightarrow B,\ v : B, H' \vdash G$

As the proof proceeds, the two subgoals 1 and 2 with conclusions A and G respectively will be refined, say with proof terms a and $g(f, v)$ respectively. We need to indicate that the value v is $ap(f;a)$, but at the point where the rule is applied, we only have slots for these subterms and a name v for the new hypothesis B. So the rule form is $apseq(f; slot_a; v.slot_g(v))$ where we know that v will be assigned the value $ap(f; slot_a)$ to "sequence" the two subgoals properly. So *apseq* is a sequencing operator as well as an application, and when the subterms are created, we can evaluate the term further as we show below. We thus express the rule as follows.

$H, f : A \Rightarrow B,\ H' \vdash G$ *by* $apseq(f; slot_a; v.slot_g(v))$

$H, f : A \Rightarrow B,\ v : B, H' \vdash G$ *by* $slot_g(v)$
$H, f : A \Rightarrow B,\ H' \vdash A$ *by* $slot_a$

We evaluate the term $apseq(f;a;v.g(v))$ to $g(ap(f;a))$ or more succinctly to $g(f(a))$. This simplification can only be done on the final bottom up pass of creating a closed proof expression, one with no slots.

Semantic consistency. With this introduction, the following rules should make sense. They define what we will call the *pure proof expressions*. It should also be clear that we can easily prove by induction on the structure of proofs that there is computational evidence for every provable formula and that the evidence is polymorphic (uniform). This provides a simple semantic consistency proof for *iFOL* and easy demonstrations that specific formulas such as $P \vee \sim P$ are not provable.

2.2. First-order refinement style proof rules over domain of discourse D

Minimal Logic
Construction rules

- **And Construction**

  $H \vdash A\&B$ *by* $pair(slot_a; slot_b)$
  $H \vdash A$ *by* $slot_a$
  $H \vdash B$ *by* $slot_b$

- **Exists Construction**

  $H \vdash \exists x.B(x)$ *by* $pair(d; slot_b(d))$
  $H \vdash d \in D$ *by* $obj(d)$
  $H \vdash B(d)$ *by* $slot_b(d)$

- **Implication Construction**

  $H \vdash A \Rightarrow B$ *by* $\lambda(x.slot_b(x))$ *new* x
  $H, x : A \vdash B$ *by* $slot_b(x)$

- **All Construction**

  $H \vdash \forall x.B(x)$ *by* $\lambda(x.slot_b(x))$ *new* x
  $H, x : D \vdash B(x)$ *by* $slot_b(x)$

- **Or Construction**

  $H \vdash A \vee B$ *by* $inl(slot_l)$
  $H \vdash A$ *by* $slot_l$

  $H \vdash A \vee B$ *by* $inr(slot_r)$
  $H \vdash B$ *by* $slot_r$

Decomposition rules

- **And Decomposition**

  $H, x : A\&B,\ H' \vdash G$ *by* $spread(x; l, r.slot_g(l, r))$ *new* l, r
  $H,\ l : A,\ r : B,\ H' \vdash G$ *by* $slot_g(l, r)$

- **Exists Decomposition**

  $H, x : \exists y.B(y),\ H' \vdash G$ *by* $spread(x; d, r.slot_g(d, r))$ *new* d, r
  $H,\ d : D,\ r : B(d),\ H' \vdash G$ *by* $slot_g(d, r)$

- **Implication Decomposition**

  $H, f : A \Rightarrow B,\ H' \vdash G$ *by* $apseq(f; slot_a; v.slot_g[ap(f; slot_a)/v])$ *new* v[9]
  $H, f : A \Rightarrow B,\ H' \vdash A$ *by* $slot_a$
  $H, f : A \Rightarrow B,\ H',\ v : B \vdash G$ *by* $slot_g(v)$

- **All Decomposition**

  $H, f : \forall x.B(x),\ H' \vdash G$ *by* $apseq(f; d; v.slot_g[ap(f; d)/v])$
  $H, f : \forall x.B(x),\ H' \vdash d \in D$ *by* $obj(d)$
  $H, f : \forall x.B(x),\ H', v : B(d) \vdash G$ *by* $slot_g(v)$[10]

- **Or Decomposition**

  $H, y : A \vee B,\ H' \vdash G$ *by* $decide(y; l.leftslot(l);\ r.rightslot(r))$
  $H, l : A, H' \vdash G$ *by* $leftslot(l)$
  $H, r : B, H' \vdash G$ *by* $rightslot(r)$

- **Hypothesis**

  $H, d : D,\ H' \vdash d \in D$ *by* $obj(d)$

  $H, x : A,\ H' \vdash A$ *by* $hyp(x)$

  We usually abbreviate the justifications to *by* d and *by* x respectively.

[9] This notation shows that $ap(f; slot_a)$ is substituted for v in $g(v)$. In the CTT logic we stipulate in the rule that $v = ap(f; slot_a)$ *in* B.
[10] In the CTT logic, we use equality to stipulate that $v = ap(f; d)$ *in* $B(v)$ just before the hypothesis $v : B(d)$.

Intuitionistic Rules

- **False Decomposition**

  $H, f : \mathit{False}, H' \vdash G$ *by* $\mathit{any}(f)$

  This is the rule that distinguishes intuitionistic from minimal logic, called *ex falso quodlibet*. We use the constant *False* for intuitionistic formulas and $\bot$ for minimal ones to distinguish the logics. In practice, we would use only one constant, say $\bot$, and simply add the above rule with $\bot$ for *False* to axiomatize *iFOL*. However, for our results it's especially important to be clear about the difference, so we use both notations.

Note that we use the term d to denote objects in the domain of discourse D. In the classical evidence semantics, we assume that D is non-empty by postulating the existence of some d_0 in it. Also note that in the rule for *False* Decomposition, it is important to use the $\mathit{any}(f)$ term which allows us to thread the explanation for how *False* was derived into the justification for G.

Classical Rules

- **Non-empty Domain of Discourse**

  $H \vdash d_0 \in D$ *by* $\mathit{obj}(d_0)$

- **Law of Excluded Middle (LEM)**

  Define $\sim A$ as $(A \Rightarrow \mathit{False})$
  $H \vdash (A \vee \sim A)$ *by* $\mathbf{magic}(A)$

  Note that this is the only rule that mentions a formula in the rule name.

2.3. Computation rules

Each of the rule forms when completely filled in becomes a term in an applied lambda calculus [24,17,5], and there are *computation rules* that define how to reduce these terms in one step. These rules are given in detail in several papers about Computational Type Theory and Intuitionistic Type Theory, so we do not repeat them here. One of the most detailed accounts is in the book *Implementing Mathematics with the Nuprl Proof Development System* [20,50] and in ITT82 [57]. They are discussed in textbooks on programming languages such as *Types and Programming Languages* [65] and *Type Theory and Functional Programming* [78].

Some parts of the computation theory are needed here, such as the notion that all the terms used in the rules can be reduced to *head normal form*. Defining that reduction requires identifying the *principal argument places* in each term. We give this definition in the next section.

The reduction rules are simple. For $\mathit{ap}(f; a)$, first reduce f, if it becomes a function term, $\lambda(x.b)$, then reduce the function term to $b[a/x]$, that is, substitute the argument a for the variable x in the body of the function b and continue computing. If it does not reduce to a function, then no further reductions are possible and the computation aborts. It is possible that such a reduction will abort or continue indefinitely. But the terms arising from proofs will always reduce to normal form. This fact is discussed in the references.

To reduce $\mathit{spread}(p; x, y.g)$, reduce the principal argument p. If it does not reduce to $\mathit{pair}(a; b)$, then there are no further reductions, otherwise, reduce $g[a/x, b/y]$.

To reduce $decide(d; l.left; r.right)$, reduce the principal argument d until it becomes either $inl(a)$ or $inr(b)$ or aborts or fails to terminate.[11] In the first case, continue by reducing $left[a/l]$ and in the other case, continue by reducing $right[b/r]$.

It is important to see that none of the first-order proof terms is recursive, and it is not possible to hypothesize such terms without adding new computation forms. It is thus easy to see that all evidence terms terminate on all inputs from all models. We state this below as a theorem about valid evidence structures.

Fact Every uniform evidence term for minimal, intuitionistic, and classical logic denotes canonical evidence, and the functional terms terminate on any inputs from any model.

Additional notations. It is useful to generalize the semantic operators to n-ary versions. For example, we will write λ terms of the form $\lambda(x_1, \ldots, x_n.b)$ and a corresponding n-ary application, $f(x_1, \ldots, x_n)$. We allow n-ary conjunctions and n-tuples which we decompose using $spread_n(p; x_1, \ldots, x_n.b)$. More rarely we use n-ary disjunction and the decider, $decide_n(d; case_1.b_1; \ldots; case_n.b_n)$. It is clear how to extend the computation rules and how to define these operators in terms of the primitive ones.

It is also useful to define *True* to be the type $\bot \Rightarrow \bot$ with element $id = \lambda(x.x)$. Note that $\lambda(x.spread(pair(x; x); x_1, x_2.x_1))$ is computationally equivalent to id [41], as is $\lambda(x.decide(inr(x); l.x; r.x))$.[12]

3. Main theorems

3.1. Evidence semantics

We start by defining first-order languages and then define *realizability evidence semantics* in a form that captures BHK semantics.[13] We note that if we use effective computation with respect to classical oracles, the semantics applies to classical logic as well in some sense. This observation is not important for our main results.

Definition 1. A first order language $\mathcal{L}$ is a symbol D and a finite set of relation symbols $\{R_i | i \in I\}$ with, I finite, given arities $\{n_i | i \in I\}$. First order formulas, $\mathcal{F}(\mathcal{L})$, over $\mathcal{L}$ are defined as usual. The variables in a formula (which range over D) are taken from a fixed set $Var = \{d_i | i \in \mathbb{N}\}$. Negation $\neg\psi$ can be defined to be $\psi \Rightarrow False$. The first order formulas of minimal logic, $\mathcal{MF}(\mathcal{L})$, are the formulas in $\mathcal{F}(\mathcal{L})$ that do not use either negation or *False*.[14]

In type theory, the propositions, $\mathbb{P}$, are identified with types.[15] A non-empty type is a true proposition and members of the type are the evidence for the truth of the proposition. An empty type provides no evidence and represents a false proposition.

Definition 2. A structure M for $\mathcal{L}$ is a mapping that assigns to D a type $M(D)$ and assigns to each R_i a term of type $M(D)^{n_i} \to \mathbb{P}$. We write $\mathcal{S}(\mathcal{L})$ for the type[16] of structures for $\mathcal{L}$. If $M \in \mathcal{S}(\mathcal{L})$ and $x \in M(D)$ then $M[d := x]$ is an extended structure that maps the variable d to the term x.

[11] The computation systems of *CTT* and *ITT* include diverging terms such as $fix(\lambda(x.x))$. We sometimes let *div* denote such terms.

[12] We could also use the term $\lambda(x.decide(inr(x); l.div; r.r))$ and normalization would reduce it to $\lambda(x.x)$.

[13] We use the term *evidence semantics* when we want to avoid confusion with Kleene's specific notion of realizability [44] from 1945 about which there is a very large literature and which is based on general recursive functions and an implicit use of Church's Thesis [79].

[14] The usual definition of minimal logic includes a designated constant $\bot$ and defines weak negation as $\psi \Rightarrow \bot$. We merely view $\bot$ as one of the atomic relation symbols R_i with arity $n_i = 0$.

[15] Recall that we work in a *predicative* metatheory, therefore the type of all propositions is stratified into orders or levels, written $\mathbb{P}_i$. For these results we can ignore the level of the type or just write $\mathbb{P}$.

[16] Since we work in type theory we always use types rather than sets.

Definition 3. Given $M \in \mathcal{S}(\mathcal{L})$ that has been extended to map the variables $V_0 \subseteq Var$ into $M(D)$, we extend the mapping M to all formulas in $\mathcal{F}(\mathcal{L})$ with free variables in V_0 by:

$$\begin{aligned}
M(\mathit{False}) &= \mathit{Void}\\
M\big(R_i(v_1,\ldots,v_{n_i})\big) &= M(R_i)\big(M(v_1),\ldots,M(v_{n_i})\big)\\
M(\psi_1 \& \psi_2) &= M(\psi_1) \times M(\psi_2)\\
M(\psi_1 \vee \psi_2) &= M(\psi_1) + M(\psi_2)\\
M(\psi_1 \Rightarrow \psi_2) &= M(\psi_1) \to M(\psi_2)\\
M(\neg\psi) &= M(\psi \Rightarrow \mathit{False})\\
M(\forall v.\,\psi) &= x : D \to \big(M[v := x]\big)(\psi)\\
M(\exists v.\,\psi) &= x : D \times \big(M[v := x]\big)(\psi)
\end{aligned}$$

Thus, any $M \in \mathcal{S}(\mathcal{L})$ assigns a type $M(\psi)$ to a sentence (a formula with no free variables) $\psi \in \mathcal{F}(\mathcal{L})$. $M(\psi)$ is synonymous with the proposition $M \models \psi$, and the members of type $M(\psi)$ are the evidence for $M \models \psi$.

We can extend this semantics to *classical FOL* by using effective functions computable with respect to an oracle, and for FOL we know that a formula is valid in evidence semantics iff it is valid in Tarski semantics [19].

Definition 4. A sentence $\psi \in \mathcal{F}(\mathcal{L})$ is valid if

$$\forall M \in \mathcal{S}(\mathcal{L}).\, M \models \psi$$

Evidence for the validity of ψ is a function of type $M : \mathcal{S}(\mathcal{L}) \to M(\psi)$ that computes, for each $M \in \mathcal{S}(\mathcal{L})$, evidence for $M \models \psi$.

A sentence $\psi \in \mathcal{F}(\mathcal{L})$ is uniformly valid if there is at least one term that is a member of all the types $M(\psi)$ for $M \in \mathcal{S}(\mathcal{L})$. Such a term is a member of the intersection type

$$\bigcap_{M \in \mathcal{S}(\mathcal{L})} M(\psi)$$

We write an intersection type $\bigcap_{x \in T} P(x)$ as a proposition using the notation $\forall[x : T].\, P(x)$. The square brackets indicate that evidence for the proposition $\forall[x : T].\, P(x)$ is uniform and does not depend on the choice of x.

To summarize:

$$\begin{aligned}
\psi \text{ is valid } &\equiv \forall M \in \mathcal{S}(\mathcal{L}).\, M \models \psi\\
\psi \text{ is uniformly valid } (\psi) &\equiv \forall\big[M : \mathcal{S}(\mathcal{L})\big].\, M \models \psi
\end{aligned}$$

3.2. Completeness

We write $\vdash_{IL} \psi$ to say that there is a proof of ψ in intuitionistic logic and $\vdash_{ML} \psi$ to say that there is a proof of ψ in minimal logic. From a proof in intuitionistic logic of any proposition we can construct evidence for the proposition. Automated proof assistants like Agda, Coq, Isabelle, MetaPRL, Minlog, and Nuprl can construct the evidence automatically. We observe, and can easily prove, that the evidence constructed from an intuitionistic proof of a first order formula $\psi \in \mathcal{F}(\mathcal{L})$ is actually evidence that ψ is uniformly valid. Our

main theorem states that for formulas of minimal logic the converse is also true: a uniformly valid formula is provable.

Here is some notation we use to apply Friedman's transformation and a simpler version of it due to Leivant [53]. We write ψ^A to designate the transformation of ψ that replaces all occurrences of *False* by $\bot$ and then at every atomic formula P except for $\bot$ we replace P by $P \vee \bot$. Leivant [53] in Lemma 5.1 gives a very simple proof that this transformation is an embedding of *iFOL* into *mFOL*, that means $\Gamma \vdash_{IL} \psi \Rightarrow \Gamma^A \vdash_{ML} \psi^A$. So the formula ψ^A has no occurrences of *False* and no negation. It is a formula of *mFOL*. It is immediately true that $\Gamma \vdash_{IL} \psi \Leftrightarrow \Gamma^A \vdash_{ML} \psi^A$ as well. When we write ψ^{False} we mean that *False* is substituted for $\bot$ in ψ^A.

Theorem 1. *For any* $\psi \in \mathcal{MF}(\mathcal{L})$,

$$\big(\forall[M : \mathcal{S}(\mathcal{L})].\, M \models \psi\big) \quad \Leftrightarrow \quad \vdash_{ML} \psi$$

Using Friedman's A-transformation [53], we can derive from Theorem 1 a corresponding completeness theorem for intuitionistic logic.

Corollary 1. *For any* $\psi \in \mathcal{F}(\mathcal{L})$,

$$\big(\forall[M : \mathcal{S}(\mathcal{L})].\, M \models \psi^A\big) \quad \Leftrightarrow \quad \vdash_{IL} \psi$$

Proof. By Theorem 1 it is enough to show

$$\vdash_{ML} \psi^A \quad \Leftrightarrow \quad \vdash_{IL} \psi$$

($\Rightarrow$) If $\vdash_{ML} \psi^A$ then also $\vdash_{IL} \psi^A$ for any interpretation of A including **False**. It is easy to prove, by induction on the structure of ψ that $\vdash_{IL} \psi^{\mathbf{False}} \Leftrightarrow \psi$.

($\Leftarrow$) This is Friedman's Theorem. □

We will prove Theorem 1 by defining an effective procedure that builds a tree of *evidence structures* (defined below) starting with an initial evidence structure formed from the uniform evidence term. We show that any evidence structure is either trivial (and therefore a leaf of the ultimate minimal logic proof) or else can be transformed into a finite number (either one or two) of derived evidence structures, and the transformation tells us what rule of minimal logic to use at that step of the proof.

Theorem 1 will then follow from the fact that this effective procedure must terminate and yield a finite proof tree. The termination of the procedure for an arbitrary term $evd \in \bigcap_{M \in \mathcal{S}(\mathcal{L})} M(\psi)$ is a strong claim. The evidence need not be a fully-typed term with all of its subterms typed, so there can be sections of "dead code" in the evidence that are not typable and may not be normalizable. Nevertheless the fact that the evidence is uniformly in the type $M(\psi)$ implies that the "dead code" is irrelevant and our procedure will terminate, but the proof of this fact (which follows in classical logic from König's lemma) in intuitionistic mathematics seems to require Brouwer's Fan Theorem.

If we assume that the uniform evidence term is fully normalized, then we can make a direct inductive argument for termination of our proof procedure. Since the evidence constructed from a proof in minimal logic is fully normalizable, this results in an alternate version of completeness that we state as follows:

Theorem 2. *Any* $\psi \in \mathcal{MF}(\mathcal{L})$ *is provable in minimal logic* ($\vdash_{ML} \psi$) *if and only if there is a fully normalized term evd in the type* $\bigcap_{M \in \mathcal{S}(\mathcal{L})} M(\psi)$.

We work only in intuitionistic logic, so we must avoid the use of excluded middle for propositions that are not decidable and in particular we cannot assume the proposition $evd \in M(\psi)$ is decidable. As in *ITT*,

CTT does not have decidable type checking. Because of this, we will need the concept that evidence term *evd* is *consistent with* the type $M(\psi)$. One notion of consistency that is sufficient for our proof is that there is no structure M for which $evd \notin M(\psi)$. However, the resulting proof is logically more complex than the one we give below where consistency is based on interpreting the types in *finitary* structures. We turn to this topic in the next section. Using finitary structures we can prove two key lemmas, one on Satisfiability of evidence structures and one on their Validity. Before we do this, we prove a special case of the theorem that is easy to follow and shows the need for these key lemmas.

3.3. Simple example

We first informally prove a special case of **Theorem 1** in which evidence for a formula of minimal propositional logic is normal and pure. In this case the termination argument for the proof generating procedure is by a simple induction on the depth of the normal evidence term. Moreover, this special case illustrates well the general method and one of the delicate points concerning evidence structures. It also motivates the definition of evidence structures and the rules for generating them given in Section 6. Even this special case is novel, although it could be taught to undergraduates.

Theorem 3 *(Completeness of Minimal Propositional Logic with respect to normal pure evidence). We can find (from the following proof) an effective procedure Prf such that given any a logical formula* $F(\bar{P})$ *in minimal propositional logic with* normal uniform pure *evidence evd,* $Prf(nil, F, evd)$ *builds a minimal logic proof, pf, of the formula,* $F(\bar{P})$*, that is,* $\vdash_{min} F(\bar{P})$ *by pf.*

Proof. The proof procedure *Prf* we build here is part of a more general procedure that we build later as an explicit extension of this one. *Prf* is defined by the rules given in Section 6, however we can also think of it as implicitly defined by this proof.

We proceed by induction on the depth t of the evidence term *evd*. We are given $nil \models F(\bar{P}), evd$, $depth(evd) = t$. From this we build the minimal logic proof inductively by progressively refining evidence structures. Thus to prove $\bar{h} : \bar{H}(\bar{P}) \models F(\bar{P}), evd(\bar{h})$ with satisfiable hypotheses, it suffices to know that the procedure is correct for any uniformly valid $\bar{h'} : \bar{H'}(\bar{P}) \models F'(\bar{P}), evd'(\bar{h'})$ with satisfiable hypotheses and evidence term depth t' where $t' < t$. We define these terms precisely in the next three sections.

First decide whether $evd(\bar{h})$ is canonical or non-canonical. This is a decidable property of any normal evidence term based on its outer operator. This property is fundamental to the computation system in computational and intuitionistic type theories.

Consider first the canonical cases, these can be further classified by the outermost constructors **C1–4**. Then we will consider the non-canonical ones, **D1–3**. We show that the key subterms can be classified by the destructor cases of the proof rules.

Constructor Cases C1 to C4.

1. **C1-implies:** *evd* is $\lambda(x.f_2(x))$ and F is $F_1 \Rightarrow F_2$. Generate a "derived evidence structure" by adding the hypothesis $x : F_1$ and changing the goal to get $\bar{H}, x : F_1 \models F_2, f_2(x)$. It is easy to see intuitively that in the derived evidence structure, $f_2(x)$ is still valid in a sense we will make precise later. Moreover, the hypotheses are clearly satisfiable.
   Now execute $Prf_3([\bar{H}, x : F_1], F_2, f_2(x))$ where $[H, x : F_1]$ is the new hypothesis list. The procedure is correct on this argument because the evidence term depth is smaller by one. We see intuitively that $f_2(x)$ has the type F_2 in the context of the expanded hypothesis list. We will prove this later in the article as a general fact about evidence structures.
2. **C2-and:** *evd* is $pair(f_1; f_2)$ and F is $F_1 \& F_2$. Consider evidence structures $\bar{H} \models F_i, f_i$ for $i = 1, 2$. These are clearly valid and the hypotheses remain satisfiable.

By the induction hypothesis, executing $Prf_3(\bar{H}, F_1, f_1)$ and $Prf_3(\bar{H}, F_2, f_2)$ will produce subproofs for these sequents.

The rule $pair(slot_1; slot_2)$ indicates how the subproofs fit together. We know by the induction hypothesis that the procedure succeeds since t decreases by one in each case.

3. **C3-or:** *evd* is $inl(f_l)$ or $inr(f_r)$ and F is a disjunction, say $F_l \vee F_r$. In this case we examine the evidence structures for $\bar{H} \models F_l, f_l$ and $\bar{H} \models F_r, f_r$. If *evd* is $inl(f_l)$, then because the evidence structure is valid, it must be that f_l is evidence for F_l, and we execute the proof procedure *Prf* on the first evidence structure with its satisfiable hypotheses. Otherwise execute the proof procedure on the other structure. It is easy to see that the evidence term depth decreases by one on the recursive calls to *Prf*.
4. **C4-atomic:** *evd* is a variable v and F is atomic. The only normal evidence of this form arises when the variable labels the hypothesis $v : F$. In this *base case* the evidence term has depth 0. The procedure returns $hyp(v)$.

Next are the cases and the steps we take for *non-canonical evidence* for F, i.e. the destructors. We note that there cannot be a term of the form $op(v)$ where v occurs among the hypotheses and the operator is a non-logical operator because the evidence term must be for a logical formula, and the only variables declared in the context have logical type of a subformula of F.

It remains to consider the following non-canonical elements of evidence types given the structure $\bar{H} \models F, evd$ where *evd* is not reducible further and is normal and pure. The method is to analyze the term *evd* by following the evaluation path through the *principal argument positions*, as defined above in the computation rules, until we reach a variable; that variable prevents further reduction, and it is typed in the hypothesis list $\bar{H}$, e.g. it might be d in the term $spread(decide(d; l.t_l; r.t_r); x, y.e')$. We then continue reduction by expanding the evidence term symbolically in the way specified below. The depth of the evidence term decreases when we do this, and we can see it clearly because the evidence term is normal and pure.

So consider the *outermost subterm* of *evd* encountered on the evaluation path, say $g'(x)$, denote its term context by $evd(\ldots g'(x) \ldots)$, with principal argument x typed in $\bar{H}$. We use a mnemonic naming of the principal variable x to suggest its structure, e.g. f when it is a function, p when it is a pair, c when it is a case split, an element of a disjoint union.

Destructor Cases D1 to D3.

1. **D1-implies:** $g'(f)$ is $ap(f; f_1)$ and f is a unique label for a hypothesis $f : F_1 \Rightarrow F_2$. We build from the given evidence structure two generated evidence structures, called *input* and *output*. Then we execute the procedure *Prf* on each of them. Termination is guaranteed by induction, provided we can show that the derived evidence structures are valid and satisfiable. It is easy to see that they are satisfiable, but validity requires a deeper analysis. It will involve the use of a class of *finitary models* that we define in Section 4.

   Consider first the **input** evidence structure $\bar{H}, f : F_1 \Rightarrow F_2, \bar{H}' \models F_1, f_1$ whose evidence term f_1 is a subterm of g' and whose context is $\bar{H}, f : F_1 \Rightarrow F_2, \bar{H}' \models F_1, f_1$.

   According to the induction hypothesis, $Prf_3([\bar{H}, f : F_1 \Rightarrow F_2, \bar{H}'], F_1, f_1)$ will construct a proof, say pf_{in}. The induction hypothesis applies since t decreases by one and the lemmas provide validity and satisfiability. Establishing validity is one of the crucial points of the proof, and we need to invoke properties of special finitary models we construct that guarantee f_1 has the type F_1. Essentially the argument is that we can build a model of $F_1 \Rightarrow F_2$ such that the functions in it will diverge on inputs that do not belong to F_1. This model will be one of those in the intersection of all models, hence we know that f_1 must be in the input type.

   Consider next the generated **output** evidence structure $\bar{H}, f : F_1, v : F_2, \bar{H}' \models F, evd$. Replace all occurrences of f in *evd* by the symbolic constant function $\lambda(x.v)$ for a fresh variable v. Then reduce

$evd'(\ldots v \ldots)$ to evd'. All explicit occurrences of $ap(f; f_1)$ in evd are thus replaced by v, and these are the only occurrences that will arise. It is noteworthy that we can remove the hypothesis about f since we will not need it again. The depth of evd' is less than t. Note that this depends on using the simple symbolic constant function $\lambda(x.v)$ for v a fresh variable.

We cite the lemmas to know that this generated evidence structure is valid with satisfiable hypotheses, and we note that the substitution for f enforces an implicit constraint that $v = ap(f; f_1)$ in all evidence structures generated from this one. Under this constraint, the generated structure has the same meaning as the given one. The hypotheses are clearly satisfiable.

The induction hypothesis guarantees termination of *Prf* whose term depth decreases by one, to $t - 1$.

We now add the rule $apseq(f; slot_{in}; v.slot_{out})$. Since we know by the induction hypothesis that the application of *Prf* will succeed on $evd(\ldots v \ldots)$, to produce pf_{out}, we return that value for $slot_{out}$. For $slot_{in}$ we use pf_{in}.

2. **D2-and:** $g'(p)$ is $spread(p; l, r.t(l, r))$. We know that $p : A\&B$ appears in the hypothesis list which we modify to $\bar{H}'$ by replacing $p : A\&B$ with $l : A, r : B$. We create the evidence structure $\bar{H}' \models F, evd'$ where evd' is the subterm obtained by replacing $spread(p; l, r.t(l, r))$ in evd by $spread(pair(l; r); l, r.t(\bar{x}, l, r))$ and then reducing $g(\ldots spread(pair(l; r); l, r.t(\bar{x}, l, r)) \ldots)$ as much as possible to g'' which has lower term depth $t' < t$.

   Note, these reductions might increase the size of the evidence term, but we see that the logical depth of the input to *Prf* has decreased because we remove the destructor and substitute a variable of depth one for the binding variable of the same depth.

   We also need to *update the constraints of the model* in the generated evidence structures. If there is a constraint $p = exp$ in the given structure, then we add the constraint $exp = pair(l; r)$ in the generated structures.

   We add the rule $spread(p; l, r.slot_{and})$, and g'' will go into the slot. Since $A\&B$ is satisfiable, so are the new hypotheses, $l : A$ and $r : B$.
3. **D3-or:** $g'(d)$ is $decide(d; l.t_a(\bar{x}, l); r.t_b(\bar{x}, r))$ and d appears in the hypothesis list as $d : A \vee B$. Now we generate two valid satisfiable evidence substructures $\bar{H}_1, a : A, \bar{H}_2{}' \models F, evd(\ldots g'(inl(a)) \ldots)$ and $\bar{H}_1, b : B, \bar{H}_2{}' \models F, evd(\ldots g'(inr(b)) \ldots)$, and we reduce the new evidence terms as much as possible. This symbolic computation might increase the size of the evidence terms; however, the logical depth of the typing context has decreased in each term by the Reducibility Lemma (essentially because we remove the destructor and substitute a variable for the binding variable).

   If there is a constraint $d = exp$ in the given evidence structure, then we add the constraint $d = inl(a)$ in the first case and $d = inr(b)$ in the other. □

The same kind of argument works for *mFOL* with normal evidence terms. However to handle the universal quantifiers, we need to maintain a finite function. We look briefly at this special case to motivate the issues with these functions.

Theorem 4 *(Completeness of Minimal First-Order Logic wrt normal pure evidence). Given a valid evidence structure, $\bar{h} : \bar{H}(D, \bar{R}) \models G(D, \bar{R}), evd(\bar{h})$, with fully normal pure evidence $evd(\bar{h})$ over the domain D and atomic relations $\bar{R}$ for a logical formula $G(D, \bar{R})$ in first-order minimal logic, we can build a minimal logic proof, $pf(\bar{h})$, of the goal formula $G(D, \bar{R})$, from hypotheses $\bar{H}$, that is, $\bar{h} : \bar{H} \vdash^1_{min} G(D, \bar{R})$ by $pf(\bar{h})$. From the proof we can extract a proof building procedure Prf_4 that extends Prf and builds the proof $pf(\bar{h})$.*

Proof. The evidence term, $evd(\bar{h})$ is given to be normalized and pure. In this argument we must deal with two new issues. The functions witnessing universal quantifiers may be evaluated on multiple arguments and the symbolic values they return are elaborated as we proceed. We must update functions available in the model, extending them as the procedure continues. They appear in the model as a list of applications of the

form $fx = if x =_\alpha d_i$ *then pattern*. The model is updated by adding new clauses to these function definitions as well as elaborating the patterns as the evidence structure is analyzed.

When we evaluate the functions in the evidence term, we substitute patterns for applications such as $ap(f; d_i)$, the depth of the resulting term might increase, so we must prioritize the redexes and use a more complex induction that weights some redexes higher than others. The reduction of *spread*, *decide*, *ap*, and ap_{val} count more than the reduction of the constructors. We use induction on the ordinal $(d \times \omega^2) + (p \times \omega) + c$ where d is the total number of *spread*, *decide*, *ap* terms in *evd*, and p is the total number of *pairs*, and c is the total number of *inl*, *inr*, λ, *vpairs*.

To deal with the fact that elements of the domain can be described as proof terms as we have seen before, it is important to evaluate these arguments first before "applying" the function; that is, we need a *call by value* application, both for evaluating functions and evaluating pairs that witness existentials. In the existential case, we need to evaluate the first component of the pair to a domain element d before we evaluate the second to a proof term. We must substitute the domain element into the goal formula. It is important that the domain D be a *value domain*, one whose elements have fully normal canonical forms.

We handle the propositional operators as in Theorem 3, thus in this proof we only need to analyze the goals with quantified formulas. First we deal with the constructors.

Constructor Cases C1 and C2.

1. **C1** *evd* is a $\lambda(x.g'(x))$ and G is $\forall x.B(x)$. We add a new hypothesis, $x : D$, and cite the rule $\lambda(x.slot)$ where slot is filled in by $b(x)$. The generated evidence structure for the subgoal is: $\bar{H}, x : D \models B, b(x)$. The ordinal depth of the evidence term decreases in the substructure, so by induction, we can build a proof for this case, and $Prf([\bar{h}, x : D], B(D, \bar{R})(x), b([\bar{h}, x]))$ builds it. We can also assign the type $B(x)$ to $b(x)$ in this context.
2. **C2** *evd* is $vpair(e; b)$ and G is $\exists x.B(x)$. First we evaluate the witness, e, building the new evidence structure $\bar{H} \models D, e$. The term e must evaluate to a variable d with type D since only declarations of this form can establish that a value belongs to D. Then we generate $\bar{H} \models B(d), b$, where the value d is used in place of e.

   The procedure succeeds by the induction hypothesis since the ordinal depth of the evidence is smaller and the lemmas guarantee validity and satisfiability. Note that d and b are both subterms of *evd*.

   It is quite easy to see that the generated evidence structures are valid by the definition of evidence for existentially quantified formulas which requires that the components have exactly the property needed for validity of the generated evidence structures.

   We use the rule $pair(slot_1; slot_2)$ to indicate how the subproofs fit together.

   For information only, we notice that the number of logical operators in the hypothesis list, *hn* does not increase and the logical depth of the goal, *gn* decreases.

Destructor Cases D1 and D2. It remains to consider the following non-canonical elements of evidence types given the structure $\bar{H} \models G, g$ where g is fully normal and pure. The method is to analyze the term g by following the evaluation path through the *principal argument positions* as defined above for the computation rules until we reach a variable as a principal argument. Call this the *principal variable*.

Locating the principal variable is elaborated in the Reducibility Lemma. Finding it is simple for pure normal evidence because the structure of the evidence term is explicit. The principal variable prevents further reduction, and it is typed in the hypothesis list $\bar{H}$, e.g. it might be d in the term $spread(decide(d; l.t_l; r.t_r); x, y.e')$. We then continue the symbolic computation by expanding the evidence term in the way described below and used in Theorem 3. (Intuitively, since the proof generation follows the symbolic computation in this case, it must terminate.)

To treat the case of using a hypothesis of the form $f : \forall x.B(x)$, we need to add constraints to the hypothesis list of the form $v = ap(f; d)$, just as in the D-implies case of Theorem 3. However in this proof we need to keep track of the conditions that are imposed on these functions and to update the symbolic patterns we substitute for all occurrences of f in the evidence term. We need to be aware that two applications of f to the same input d must yield the same result. We also want to keep track of the symbolic value of the output of the function as the hypotheses are refined. In this case we will be producing values for v that are *functions defined by cases* which we did not encounter in the case of implication.

1. **D-all** given $g'(f)$ is $ap(f; d_i)$, note that f references a universal statement, $f : \forall x.B(x)$. Generate two evidence structures, an *input* one and an *output* one and a *constraint* for the output structure, expressed in the hypothesis but not counted in the logical depth.
   The *input structure* is $H, f : \forall x.A(x), H' \models D, d_i$. The term d_i is atomic and must appear among the hypotheses as $d_i : D$, so the justification is atomic and the evidence structure is easily seen to be both valid and satisfiable as we proved in the lemmas. This is a *base case* of the procedure. We generate the *output structure*:

$$H, f : \forall x.B(x), H', v_i = ap(f; d_i), v_i : B(d) \models G, g\big(\ldots g'(f)\ldots\big)$$

   We proceed as in the propositional case by substituting a function term $\lambda(x.body(x))$ for f in g where the body is a conditional form *if* $x =_\alpha d_1$ *then* v_1; *if* $x =_\alpha d_2$ *then* v_2; .... These values v_i are elaborated as the evidence structures are refined, but at this point they are simple variables, and we might be substituting other symbolic λ terms for them in the constraints.
   If there is already a function constant that has been created for f, then we only need to add a clause to the already created symbolic function and include in the function body the case, *if* $x =_\alpha d_i$ *then* v_i. Otherwise, we create a new function, say f_0; then include in the function body the case, *if* $x =_\alpha d_i$ *then* v_i unless the term d_i has already appeared in the function body. Now substitute this function f_0 in g for f, $g[f_0/f]$, and reduce the term, say to g'. We will see later that this step lowers the ordinal depth of the evidence term, replacing $ap(f; d_i)$ by v_i during reduction, and the constraint guarantees that the generated evidence structure is valid with fully normal pure evidence. Thus the proof procedure *Prf* terminates in this case.
   It is important that we substitute the symbolic constant function only into the $ap(f; d_i)$ redex where it will be immediately reduced. We substitute into redexes as they emerge.
2. **D-exists** given $g'(p)$ is $spread(p; y, z.g''(y, z))$ where p references $\exists y.B(y)$, we generate the evidence structure $H, y : D, z : B(y), H' \models G, g''(y, z)$ which has an evidence term of smaller ordinal depth than t. If there is a constraint $v = ap(f; d)$ among the hypotheses, we extend it to $v = pair(y; z)$. We made the same extension in the case that p references a simple conjunction $C\&D$.
   This evidence structure is clearly valid and satisfiable, and its logical depth is decreased by one, so by the induction hypothesis, it builds the needed subproof. □

In the next sections we make precise the ideas encountered in the two informal proofs. We also considerably generalize the method so that we can handle evidence that is not in normal form and is not purely logical. This is a considerable extension of the method showing that we can use evidence terms that involve all of the operators and types of *CTT* or *ITT* or extensions of these theories that satisfy the general properties of the computation system that we have listed. In the general case, the termination argument needs a stronger form of induction given by the Fan Theorem. We are not yet in a position to formalize this argument in *CTT* and prove the theorem in Nuprl, but we aim to reach this state by extending *CTT* and its implementation.

4. Finitary types and structures

In order to prove the validity lemma, we need to refer to a special class of models that are finite structures with decidable equality. We call these *finitary structures*. In these structures we can prove that a finite function diverges on arguments not in its domain. This allows us to show the validity lemma for the input subcase of the function destructor case.

Definition 5. Types A and B are *equipollent* (written $A \sim B$) if there is a bijection $f : A \to B$. A type T is *finite* if $\exists k : \mathbb{N}. T \sim \mathbb{N}_k$ (where $\mathbb{N}_k$ is the type of numbers in the range $0 \leqslant i < k$).

Note that if T is finite then equality in T is decidable and there is a list L_T that enumerates T, i.e. contains all the members of T with no repeats. Using L_T, any function $f : T \to S$ can be converted to a table

$$\mathbf{graph}(f) = \mathbf{map}\big(\lambda x.\langle x, f(x)\rangle, L_T\big)$$

Using the decidable equality in T we can define a table lookup function and recover the function

$$f = \mathbf{lookup}\big(\mathbf{graph}(f)\big)$$

Definition 6. We write $t\downarrow$ to say that term t computes to a value (a canonical form).

The *bar type* $\bar{T}$ is the type of all terms t such that $(t\downarrow) \Rightarrow (t \in T)$.

A function $f : \mathit{Term} \to \bar{T}$ is *strict* if for all terms t

$$\big(f(t)\downarrow\big) \Rightarrow (t\downarrow)$$

A type T is a *value type* if every member of T converges to a value.

A bar type $\bar{T}$ is not a value type, but even without bar types, a rich type theory that includes intersection types or quotient types will have some types that are not value types.

Definition 7. A type T is a *retract* if there is a strict function i_T of type $\mathit{Term} \to \bar{T}$ such that

$$\forall t : T. i_T(t) = t \in T$$

or equivalently

$$i_T = id \in (T \to T)$$

A type T is *finitary* if it is a finite, value type and a retract.

A structure $M \in \mathcal{S}(\mathcal{L})$ is *finitary* if $M(D)$ is finitary and the types $M(R_i)(d_1, \ldots, d_{n_i})$ assigned to the atomic formulas are finitary.

We let **abort** be a fixed term that has no redex but is not a canonical form. For example **abort** could be $0(0)$ or $\mathit{true} + 5$ or a primitive. The term **abort** does not converge to a value. We use this to construct simple examples of finitary types.

Example 1. The type $\mathbb{N}_k$ is a finitary type. The retraction $i_{\mathbb{N}_k}$ is

$$\lambda t.(\mathbf{if}\ 0 \leqslant t\ \&\ t < k\ \mathbf{then}\ t\ \mathbf{else}\ \mathbf{abort})$$

Example 2. The type *Unit* with a single canonical member $\star$ is a finitary type. The retraction is

$$\lambda t.(\textbf{if } t = \star \textbf{ then } t \textbf{ else abort})$$

These examples depend on the existence of primitive computations that recognize the canonical forms of the intended members of the type. We mention here some additional assumptions about the underlying computation system on which our proof of completeness depends. These assumptions are satisfied by the computation system of *CTT* used by Nuprl, but our proof could easily be modified to work for type theories based on different primitive computations.[17]

Assumption 1. *The only primitive redex involving a pair* $\langle t_1, t_2\rangle$ *is*

$$\textbf{spread}\big(\langle t_1, t_2\rangle; x, y.B(x, y)\big) \mapsto B(t_1, t_2)$$

The only primitive redex involving **inl** t *is*

$$\textbf{decide}\big(\textbf{inl } t; x.B(x); y.C(y)\big) \mapsto B(t)$$

The only primitive redex involving **inr** t *is*

$$\textbf{decide}\big(\textbf{inr } t; x.B(x); y.C(y)\big) \mapsto C(t)$$

The only primitive redex involving $\lambda x.B(x)$ *is*

$$\big(\lambda x.B(x)\big)(t) \mapsto B(t)$$

Lemma 1. *If* A *and* B *are finitary then* $A + B$ *is finitary. If* A *is finitary and for all* $a \in A$, $B(a)$ *is finitary then the types* $a : A \to B(a)$ *and* $a : A \times B(a)$ *are both finitary.*

Proof. It is straightforward to prove that the types are finite, value types. The retraction maps i, j, and k for $A + B$, $a : A \to B(a)$, and $a : A \times B(a)$ are

$$\begin{aligned}
i(t) &= \textbf{decide}\big(t; x.i_A(x); y.i_B(y)\big)\\
j(t) &= \textbf{lookup}\big(\textbf{graph}\big(\lambda a.i_{B(a)}\big(t(a)\big)\big)\big)\\
k(t) &= \textbf{spread}\big(t; x, y.\big(\lambda a.\big(\lambda b.\langle a, b\rangle\big)\big(\textbf{val}\, i_{B(a)}(y)\big)\big)\big(\textbf{val}\, i_A(x)\big)\big)
\end{aligned}$$

The operation $f(\textbf{val}\, x)$ is a *call-by-value apply*, so for the retraction $k(t)$ to converge, the term t must evaluate to a pair $\langle x, y\rangle$, the value $a = i_A(x)$ must converge, and the value $b = i_{B(a)}(y)$ must converge, before the pair $\langle a, b\rangle \in a : A \times B(a)$ is formed. □

Corollary 2. *If* $M \in \mathcal{S}(\mathcal{L})$ *is finitary and* $\psi \in \mathcal{F}(\mathcal{L})$ *then* $M(\psi)$ *is finitary.*

Definition 8. We abbreviate $(\lambda a.\langle a, y\rangle)(\textbf{val}\, x)$ as $\langle \textbf{val}\, x, y\rangle$. This operation forms a pair only after the first component has been evaluated.

[17] For example, the computation system could have primitive projection functions π_1 and π_2 rather than the **spread** primitive. It could have primitives for **isl**, **outl**, **isr**, and **outr** rather than the **decide** primitive. Our construction would be easily modified to accommodate such differences.

Definition 9. A term t is *consistent with* a retract type T if $i_T(t) \in T$ or, equivalently, if $i_T(t)\downarrow$.

If A is a retract then a function $f : A \to B$ is *tight* if the domain of f contains only terms consistent with A, i.e. if for all terms t

$$\big(f(t)\downarrow\big) \Rightarrow \big(i_A(t)\downarrow\big)$$

Lemma 2. *If f has type $A \to B$ and A is a retract, then there is a tight function $f' = f \in (A \to B)$.*

Proof. Let $f' = f \circ i_A$ where i_A is the retraction onto A. Since i_A is the identity on A, we have $f' = f \in (A \to B)$. The domain of f' contains only terms in the domain of i_A. □

Definition 10. For any $\sigma : V_0 \to T_0$ that is an injection from a finite subset $V_0 \subset \mathit{Var}$ into a finitary type T_0 we define a finitary $\mathcal{L}$-structure $M_{triv}(\sigma)$ by

$$\begin{aligned} M(D) &= T_0 \\ M(v) &= \sigma(v) \\ M(R_i) &= \lambda x_1, \ldots, x_{n_i}.\mathit{Unit} \end{aligned}$$

Satisfiability Lemma.

Lemma 3. *For any $\psi \in \mathcal{MF}(\mathcal{L})$ with free variables in V_0, $M_{triv} \models \psi$.*

Proof. The structure $M_{triv}(\sigma)$ assigns to every atomic formula the non-empty type *Unit*. It is then clear that $M_{triv}(\sigma)$ assigns a non-empty type to every minimal logic formula ψ and hence $M_{triv} \models \psi$. □

This would not be true for general first-order formulas that include negation and False.

5. Evidence structures

We will use the concept of an *evidence structure* to build a bridge between uniform evidence terms and proofs. An evidence structure will have three parts, a context H, a goal G, and evidence term *evd*. The context H will include some declarations of the form $d_i : D$ where $d_i \in \mathit{Var}$ (the variables in $\mathcal{F}(\mathcal{L})$), but it will also include declarations of the form $v_i : A$ where $A \in \mathcal{MF}(\mathcal{L})$ is a subformula of the original goal ψ and v_i is a variable chosen from another set of variables $\mathit{Var}' = \{v_0, v_1, v_2, \ldots\}$ disjoint from $\mathit{Var} = \{d_0, d_1, \ldots\}$. The context H will also contain *constraints* of the form $f(\mathbf{val}\, d) = t$ where $f \in \mathit{Var}'$ and term t is a *pattern* over H.

Definition 11. Given a set H of variable declarations $v : T$, the set of patterns over H is the set of typed terms defined inductively by:

1. Any $v : T \in H$ is a pattern.
2. If $ptn_1 : A$ and $ptn_2 : B$ are patterns then the following are patterns:
   - $\langle ptn_1, ptn_2\rangle : (A \times B)$
   - **inl** $ptn_1 : (A + B)$
   - **inr** $ptn_2 : (A + B)$.

Definition 12. A typing H over $\mathcal{L}$ is a list of declarations of one of the two forms:

1. $d : D$ where $d \in \mathit{Var}$.

2. $v : A$ where $v \in Var'$ and $A \in \mathcal{MF}(\mathcal{L})$ such that every free variable d of A, is declared in H.

A *model* M of H is a finitary structure for $\mathcal{L}$ extended so that for each $v : T$ in H, $M(v) \in M(T)$.

Definition 13. An *implies constraint* on a typing H is an equation

$$v_i = \mathbf{constant}(t)$$

where $v_i : A \Rightarrow B \in H$ and t is a pattern of type B. The constraint is *stratified* if for any variable v_j in pattern t, $i < j$. The constraint is *unique* in H if there is no other constraint $v_i = \mathbf{constant}(t')$ in H. A model M of H satisfies the constraint if $M(v_i) = \lambda x.M(t) \in (M(A) \to M(B))$.

A *forall constraint* is an equation

$$v_i(\mathbf{val}\, d) = t$$

where $d : D \in H$ and for some formula $P \in \mathcal{MF}(\mathcal{L})$, $v_i : \forall z.\, P \in H$ and t is a pattern of type $P(d)$ over H. The constraint is *stratified* if for any variable v_j in pattern t, $i < j$. The constraint is *unique* in H if there is no other constraint $v_i(\mathbf{val}\, d) = t'$ in H. A model M of H satisfies the constraint if $M(v_i)(M(d)) = M(t) \in M(P(d))$.

An evidence context H over $\mathcal{L}$ is a list of declarations and unique, stratified constraints such that the declarations are a typing over $\mathcal{L}$ and the constraints are constraints on that typing. M is a *model of context* H if it is a model of the typing H that satisfies all the constraints. We write $M \models H$ to say that M is a model of context H; note that this means that $M \in \mathcal{S}(\mathcal{L})$ and M is finitary.

Definition 14. A model $M \models H$ is *tight* if for every $f : A \to B \in H$, the function $M(f)$ is tight. We write $M \models_t H$ when M is tight.

Lemma 4. *Every evidence context H over $\mathcal{L}$ has a tight model.*

Proof. Let V_0 be the variables d_i for which $d_i : D \in H$. We first choose a finitary type T_0 and an injection $\sigma : V_0 \to T_0$ (we can use $\mathbb{N}_k$ for $k > |V_0|$). We construct the model M by extending the model $M_{triv}(\sigma)$, choosing values for the variables that satisfy all the constraints. Since the constraints are stratified, we choose values for the variables in reverse order. Let $v_j \in Var'$ be a variable with a declaration $v_j : T \in H$ and assume that we have chosen values for all variables v_k in H with $j < k$. Assign a value to all patterns t all of whose variables v_k have $k > j$ recursively as follows: $M(\langle p_1, p_2\rangle) = \langle M(p_1), M(p_2)\rangle$, $M(\mathbf{inl}\ p) = \mathbf{inl}\ M(p)$, $M(\mathbf{inr}\ p) = \mathbf{inr}\ M(p)$.

If T is $\forall x.\, P$ for some P, then for each $d_i \in V_0$ we choose a value $w_i \in M(P(d_i))$ as follows: If there is a (unique) constraint $v_j(\mathbf{val}\, d_i) = t_i$ in H then we use $w_i = M(t_i) \in M(P(d_i))$ (which is defined since values for the variables in pattern t_i have already been chosen). Otherwise we choose w_i from the non-empty type $M(P(d_i))$. Since the values $M(d_i)$ are all distinct members of the finite type $M(D) = T_0$, we set the value of v_j to be a function of type $x : M(D) \to M(P(x))$ that maps each $M(d_i)$ to w_i. This function is $\mathbf{lookup}([\langle M(d_i), w_i\rangle | d_i \in V_0])$.

If T is $A \Rightarrow B$ then if there is a (unique) implies constraint $v_j = \mathbf{constant}(t)$ then let $w = M(t)$ and choose the constant function $\lambda x.w$ made tight by applying Lemma 2. If there is no constraint on v_j then we choose any member of the non-empty type $M(A) \to M(B)$ and make the chosen function tight by applying Lemma 2.

Otherwise there are no constraints on v_j, and $M_{triv}(\sigma)(T)$ is non-empty by Lemma 3 so we may choose a value for v_j from this type. □

$$\frac{H \models G_1 \& G_2, \langle evd_1, evd_2\rangle}{H \models G_1, evd_1 \qquad H \models G_2, evd_2}\ \&\text{PAIR}$$

$$\frac{H \models \exists x.\, G, \langle evd_1, evd_2\rangle}{H \models \exists x.\, G, \langle \mathbf{val}\, evd_1, evd_2\rangle}\ \exists\text{PAIR}$$

$$\frac{d : D \in H \models \exists x.\, G, \langle \mathbf{val}\, d, evd\rangle \ (d \textit{ a variable})}{H \models G[x := d], evd}\ \exists\text{VAL PAIR}$$

$$\frac{H \models G_1 \vee G_2, \mathbf{inl}\ evd}{H \models G_1, evd}\ \vee\text{INL} \qquad \frac{H \models G_1 \vee G_2, \mathbf{inr}\ evd}{H \models G_2, evd}\ \vee\text{INR} \qquad \frac{H \models G_1 \Rightarrow G_2, \lambda x.evd}{H; x : G_1 \models G_2, evd}\ \Rightarrow \lambda$$

$$\frac{H \models \forall y.\, G, \lambda x.evd}{H; d : D \models G[y := d], evd}\ \forall\lambda$$

The bound variable in $\lambda x.evd$ in $\Rightarrow \lambda$ is renamed to avoid variables in H and the variable d in $\forall\lambda$ is fresh.

Fig. 1. Rules for evidence structures with canonical evidence.

Definition 15. An evidence structure is a triple $H \models G, evd$ where

1. H is an evidence context.
2. $G \in \mathcal{MF}(\mathcal{L})$.
3. for every $M \in \mathcal{S}(\mathcal{L})$, if $M \models_t H$ then $M(evd)$ is consistent with $M(G)$.

We write $t[v := e]$ for the result of substitution of e for variable v in term t, and we write $(H \models G, evd)[v := e]$ for the result of substitution of e for v everywhere in the evidence structure $H \models G, evd$.

Observation 1. *If evd is uniform evidence for a formula $\psi \in \mathcal{MF}(\mathcal{L})$ then*

$$\models \psi, evd$$

is an evidence structure.

6. Derivation rules for evidence structures

We now define a set of sixteen derivation rules by which we derive evidence structures from evidence structures. We will prove that

1. If $H \models G, evd$ is an evidence structure, then evd computes to evd' that is canonical or has a principal argument that is a variable.
2. $H \models G, evd'$ is an evidence structure that matches one of the sixteen derivation rules.
3. This defines a recursive procedure on evidence structures that results in a tree of *derived evidence structures*.
4. The tree derived from $(\models \psi, evd)$ is finite, and from it we can construct a minimal logic proof of ψ.

The first seven derivation rules shown in Fig. 1 match evidence structures where the evidence is in canonical form.

Definition 16. An evidence derivation rule is valid if for any evidence structure matching the pattern above the line, the derived instances below the line are evidence structures.

Validity Lemma 1

Lemma 5. *The rules in Fig. 1 are valid.*

Proof. Since these rules do not add constraints to the context, we only have to prove that the derived evidence term is consistent with the derived goal.

For the rule &PAIR, suppose $M \models H$, then $\langle evd_1, evd_2\rangle$ is consistent with $M(G_1 \& G_2)$. So,

$$\begin{aligned} i_{M(G_1)\times M(G_2)}\big(\langle evd_1, evd_2\rangle\big)\downarrow &\Rightarrow \\ \big(\lambda a.\big(\lambda b.\langle a,b\rangle\big)\big(\mathbf{val}\, i_{M(G_2)}(evd_2)\big)\big)\big(\mathbf{val}\, i_{M(G_1)}(evd_1)\big)\downarrow &\Rightarrow \\ i_{M(G_1)}(evd_1)\downarrow i_{M(G_2)}(evd_2)\downarrow & \end{aligned}$$

For the rule $\Rightarrow \lambda$, suppose $M \models H; x : G_1$, then $\lambda x.evd$ is consistent with $M(G_1 \Rightarrow G_2)$. So,

$$\begin{aligned} i_{M(G_1)\to M(G_2)}(\lambda x.evd)\downarrow &\Rightarrow \\ \mathbf{graph}\big(\lambda a.i_{M(G_2)}\big(evd(a)\big)\big)\downarrow &\Rightarrow \\ \forall a \in M(G_1).\, i_{M(G_2)}\big(evd(a)\big)\downarrow &\Rightarrow \\ i_{M(G_2)}\big(evd\big(M(x)\big)\big)\downarrow & \end{aligned}$$

The proofs of validity of the other rules for canonical evidence are similar to these. □

Definition 17. The principal subterm $principal(t)$ of term t is defined inductively by:

$$\begin{aligned} principal\big(\mathbf{decide}(d; x.a; y.b)\big) &= principal(d) \\ principal\big(\mathbf{spread}(p; x, y.b)\big) &= principal(p) \\ principal\big(f(b)\big) &= principal(f) \\ principal\big(f(\mathbf{val}\, b)\big) &= principal(b) \\ principal\big(\langle \mathbf{val}\, a, b\rangle\big) &= principal(a) \\ principal(t) &= t, otherwise \end{aligned}$$

We write $t(\underline{x})$ when x is the principal subterm of $t(x)$.

The remaining rules match evidence that is not in canonical form. If a term is not in canonical form but some instance of it will compute to canonical form then the term must have a subterm that is a variable and the computation depends on the value of that variable in order to proceed. We call such a variable the *principal variable* and any subterm in such a position a *principal subterm*.

The rules shown in Fig. 2 match on the operator that is applied to the principal variable in the evidence. When a fresh variable from Var' is needed, we take the least index greater than all the variables already in use. This will guarantee that all the constraints remain stratified.

Lemma 6. *The constraints in the evidence structures derived from the rules in Fig. 2 are unique, stratified constraints.*

Proof. Only the rules $\Rightarrow$ APPLY and $\forall$CBV add new constraints and they apply only when there is not already a similar constraint. The constraints are changed only by the rules (DECIDE, SPREAD, and $\Rightarrow$ APPLY) that substitute a pattern (**inl** x, **inr** y, or $\langle x, y\rangle$) for a variable. In each case, the new variables introduced are

$$\text{VAR}\quad \frac{}{H_1; v : G; H_2 \models G, v}$$

$$\text{DECIDE}\quad \frac{H_1; c : A \vee B; H_2 \models G, evd(\mathbf{decide}(\underline{c}; x.a; y.b))}{(H_1; x : A; H_2 \models G, evd(a))[c := \mathbf{inl}\ x] \qquad (H_1; y : B; H_2 \models G, evd(b))[c := \mathbf{inr}\ y]}$$

$$\&\text{SPREAD}\quad \frac{H_1; p : A\&B; H_2 \models G, evd(\mathbf{spread}(\underline{p}; x, y.t))}{(H_1; x : A; y : B; H_2 \models G, evd(t))[p := \langle x, y\rangle]}$$

$$\exists\text{SPREAD}\quad \frac{H_1; p : \exists z.\, P; H_2 \models G, evd(\mathbf{spread}(\underline{p}; x, y.t))}{(H_1; d : D; y : P[z := x]; H_2 \models G, evd(t))[p := \langle d, y\rangle]}$$

$$\text{APPLY CONST}\quad \frac{f = \mathbf{constant}(v) \in H \models G, evd(\underline{f}(t))}{H \models G, evd(v)}$$

$$\Rightarrow\text{APPLY}\quad \frac{\nexists v. f = \mathbf{constant}(v) \in H_1; f : A \Rightarrow B; H_2 \models G, evd(\underline{f}(t))}{H_1; f : A \Rightarrow B; H_2 \models A, t \qquad H_1; f : A \Rightarrow B; H_2; v : B; f = \mathbf{constant}(v) \models G, evd(v)}$$

$$\forall\text{APPLY}\quad \frac{f : \forall z.\, P \in H \models G, evd(\underline{f}(t))}{H \models G, evd(f(\mathbf{val}\, t))} \qquad \text{APPLY MODEL}\quad \frac{f(\mathbf{val}\, d) = t \in H \models G, evd(f(\mathbf{val}\, \underline{d}))}{H \models G, evd(t)}$$

$$\forall\text{CBV}\quad \frac{\nexists t.\, f(\mathbf{val}\, d) = t \in H, \{f : \forall z.\, P, d : D\} \subseteq H \models G, evd(f(\mathbf{val}\, \underline{d}))}{H; w : P[z := d]; f(\mathbf{val}\, d) = w \models G, evd(w)}$$

The bound variables, d, x, and y, in rules DECIDE and SPREAD are renamed to avoid variables in H. The variables, v and w, introduced in rules $\Rightarrow$ APPLY and CBV are fresh.

Fig. 2. Rules for evidence structure with non-canonical evidence.

fresh, and substituting a pattern for a variable in a pattern results in a pattern, so all the constraints remain unique, stratified patterns. □

Validity Lemma 2

Lemma 7. *The rules in Fig. 2 are valid.*

Proof. Because it depends on the restriction to tight models, we consider first the proof of

$$\Rightarrow\text{APPLY}\quad \frac{\nexists v. f = \mathbf{constant}(v) \in H_1; f : A \Rightarrow B; H_2 \models G, evd(\underline{f}(t))}{H_1; f : A \Rightarrow B; H_2 \models A, t \qquad H_1; f : A \Rightarrow B; H_2; v : B; f = \mathbf{constant}(v) \models G, evd(v)}$$

Assume that the structure above the line is an evidence structure, and let $M \models_t H_1; f : A \Rightarrow B; H_2$. Then $M(evd(\underline{f}(t)))$ is consistent with $M(G)$. Since $i_{M(G)}$ is strict, this implies that $M(t)$ is in the domain of $M(f)$ and since M is tight, $M(t)$ is consistent with $M(A)$. This proves the validity of the first derived structure $H_1; f : A \Rightarrow B; H_2 \models A, t$.

If $M \models_t H_1; f : A \Rightarrow B; H_2; v : B; f = \mathbf{constant}(v)$ then the model M is also a tight model of $H_1; f : A \Rightarrow B; H_2$ so $M(evd(\underline{f}(t)))$ is consistent with $M(G)$ and this implies that $M(evd(v))$ is consistent with $M(G)$ because $M(f(t))$ must converge to $M(v)$. This proves the validity of the second derived structure and finishes the proof of the rule $\Rightarrow$ APPLY.

Consider next the rule

$$\frac{\forall\text{CBV}\quad \nexists t.\, f(\mathbf{val}\, d) = t \in H, \{f : \forall z.\, P, d : D\} \subseteq H \models G, evd(f(\mathbf{val}\, \underline{d}))}{H; w : P[z := d]; f(\mathbf{val}\, d) = w \models G, evd(w)}$$

If $M \models_t H; w : P[z := d]; f(\mathbf{val}\, d) = w$ then $M \models_t H$ and because $M(D)$ is a value type, $M(f(\mathbf{val}\, d)) = M(f(d)) = M(w) \in M(P(d))$. Since $M(evd(f(\mathbf{val}\, \underline{d})))$ is consistent with $M(G)$ and $i_{M(G)}$ is strict, this implies that $M(evd(w))$ is consistent with $M(G)$. So $\forall$CBV is a valid rule.

Consider next the rule

$$\frac{\exists\text{SPREAD}\quad H_1; p : \exists z.\, P; H_2 \models G, evd(\mathbf{spread}(\underline{p}; x, y.t))}{(H_1; d : D; y : P[z := x]; H_2 \models G, evd(t))[p := \langle d, y\rangle]}$$

If $M \models_t (H_1; d : D; y : P[z := x]; H_2)[p := \langle d, y\rangle]$ then the model

$$M' = M\big[p := \big\langle M(d), M(y)\big\rangle\big]$$

is a tight model of $H_1; p : \exists z.\, P; H_2$, so $M'(evd(\mathbf{spread}(\underline{p}; x, y.t)))$ is consistent with $M'(G)$. This implies that $M(evd(t))[p := \langle d, y\rangle]$ is consistent with $M'(G) = M(G)$.

The proofs for the validity of the remaining rules are similar to these. □

Lemma 8. *If $H \models G$, evd is an evidence structure, and evd' is obtained by computing evd until it is canonical or has a principal variable, then $H \models G, evd'$ is an evidence structure.*

Proof. If $M \models_t H$ then $M(evd)$ is consistent with $M(G)$ so $(i_{M(G)}(evd)\downarrow)$. This implies $(i_{M(G)}(evd')\downarrow)$ since the computations are the same. □

Lemma 9. *If $H \models G, evd$ is an evidence structure, and evd is canonical or a principal variable, then $H \models G, evd$ matches one of the sixteen rules in Figs. 1 and 2.*

Proof. By Lemma 4 there is a tight model $M \models_t H$. Thus, $M(evd)$ is consistent with $M(G)$. If *evd* is canonical, then $H \models G, evd$ must match one of the rules in Fig. 1 because the type $M(G)$ must be a product, union, or function type.

If *evd* has a principal variable v then $v : T \in H$ for some $T \in \mathcal{MF}(\mathcal{L})$ and $M(v) \in M(T)$. Since v is principal and $i_{M(G)}$ is strict, the computation $i_{M(G)}(M(evd))$ must reduce the subterm of $M(evd)$ containing $M(v)$. Since $M(T)$ must be a product, union, or function type, only a spread, decide, apply, or call-by-value apply redex can apply. Therefore one of the rules in Fig. 2 must match $H \models G, evd$. □

The preceding lemmas show that there is a well defined procedure that starts with the evidence structure $(\models \psi, evd)$ constructed from uniform evidence for ψ and recursively builds a tree of evidence structures by alternating computation of *evd* until it is canonical or has a principal variable with matching the evidence structure against the sixteen derivation rules and applying the derivation.

It is routine to check that each derivation corresponds to a proof rule of minimal logic. In our implementation of the proof procedure (shown in Appendix A) we need only the evidence term *evd* and the constraints (which we call the "model") because the typing and the goal are just the current hypotheses and goal of the sequent being proved. From this information the recursive Nuprl tactic decides which derivation rule to apply (or that it needs to compute the evidence term) and then updates the evidence and constraints and uses one of the primitive logical rules to get the next typing hypotheses and next goal term.

Our Theorem 1 is proved once we establish that the recursive procedure terminates.

7. Termination of the Proof Procedure

We first show termination under the assumption that *evd* is fully normalized, which will establish Theorem 2.

Lemma 10. *If evd is fully normalized then the evidence structure generation procedure terminates.*

Proof. Let $nc(evd)$ be the number of occurrences of **decide**, **spread**, or **apply** operators in term *evd*. Let $cbv(evd)$ be the number of occurrences of the call-by-value apply operator. Let $npr(evd)$ be the number of occurrences of the $\langle x, y\rangle$ operator. Let $cn(evd)$ be the number of occurrences of the $\langle \mathbf{val}\, x, y\rangle$, **inl** x, or **inr** y operators.

We prove termination by induction on the lexicographically ordered tuple

$$\big\langle nc(evd), cbv(evd), npr(evd), cn(evd)\big\rangle$$

Each rule in Fig. 1 changes *evd* to a subterm of *evd* and removes at least one of the counted operators except for rule ∃PAIR which changes a $\langle x, y\rangle$ into a $\langle \mathbf{val}\, x, y\rangle$, so the measure decreases in each of these steps.

Some of the rules in Fig. 2 reduce the measure by replacing a subterm of *evd* that includes a **decide**, **spread**, or **apply** operator by a variable and then substituting a pattern into the result. This reduces the $nc(evd)$ count and may increase only the $npr(evd)$ and $cn(evd)$ counts because patterns have only $\langle x, y\rangle$, **inl** x, or **inr** y operators. Thus, in every case it is easily checked that the measure decreases.

It remains to show that in the computation steps that compute *evd* until it is canonical or has a principal variable we can in fact fully normalize the evidence term and that this will not increase the measure.

If *evd* is fully normalized, then the only rules which derive evidence that may not be fully normalized are those that substitute a pattern. The resulting term evd', which has some pattern *ptn* in some places where *evd* had a variable, can contain only spread, decide, apply, or call-by-value apply redexes. When these are reduced, they result only in sub-patterns of *ptn* being substituted for variables. Thus, by induction on the size of *ptn* we can show that normalization of evd' terminates and does not increase the measure. □

The proof of termination for the general case where *evd* is not assumed to be fully normalized uses Brouwer's Fan Theorem. For that proof we need the following definitions and lemmas.

Definition 18. A derivation rule is *constant domain* if it does not add a new variable $d_i : D$ to the derived contexts.

All of the derivation rules in Figs. 1 and 2 are constant domain except for the rules ∀λ and ∃SPREAD.

Definition 19. A derivation is a ψ-derivation if it is an instance of one of the derivation rules where the formulas in the context H and goal G are instances of subformulas of ψ.

Definition 20. Context H' is a *constant domain ψ-extension* of context H (written $H <^{\psi}_{cd} H'$) if H' can be obtained from H by applying ψ-derivations that are instances of constant domain derivation rules.

Context H is a *maximal ψ-context* if there is no proper H' with $H <^{\psi}_{cd} H'$.

Lemma 11. *For any formula $\psi \in \mathcal{MF}(\mathcal{L})$, and any context H there are only finitely many H' such that $H <^{\psi}_{cd} H'$.*

Proof. Let D_0 be the set of variables $d_i : D \in H$. Repeated application of the constant domain ψ-derivations will add new declarations and constraints $v : P(d); f(\mathbf{val}\, d) = v$ for all the universally quantified declarations

$f : \forall x.\, P(x)$ and every $d \in D_0$. These new declarations will in turn be instantiated with every $d \in D_0$. Any declarations of the form $v : A \vee B$ will generate two derived extensions where v is replaced by either **inl** x for $x : A$ or by **inr** y for $y : B$. Declarations of the form $p : A\&B$ will be replaced by $x : A; y : B$ and p replaced by $\langle x, y\rangle$. Every subformula of ψ may be added to the context with its free variables replaced by members of D_0. But for a finite D_0 and fixed formula ψ there are only finitely many such extensions. □

Corollary 3. *For any context H there is a finite, non-empty set of maximal ψ-contexts H' such that $H <^{\psi}_{cd} H'$.*

Definition 21. The *one step ψ-extension* of H is obtained from H by adding $d_i : D$ for the least $i \in \mathbb{N}$ for which d_i is not in H, then applying the ∃SPREAD rule to add new domain elements for every existentially quantified formula in H.

Context H' is a *next ψ-extension* of H if it is a maximal constant domain ψ-extension of the one step ψ-extension of H.

$\mathcal{SM}(\psi)$, the *spread of symbolic models* of ψ is the tree with the empty context at the root and the successors of node H being the next ψ-extensions of H.

An infinite path through $\mathcal{SM}(\psi)$ describes a freely-chosen model M with $M(D) = Var$. In this model the evidence term *evd* must compute evidence for $M(\psi)$ and we use the termination of this computation to bar the spread $\mathcal{SM}(\psi)$. Brouwer's Fan Theorem then gives a uniform bar and this implies that our proof procedure terminates on *evd* and produces a minimal logic proof of ψ.

Definition 22. Let α be an infinite path in $\mathcal{SM}(\psi)$. The computation $c(\alpha, \psi, evd, n)$ where $n > 0$, is defined by computing *evd* in the context $\alpha(n)$ (a maximal context along path α) to a term evd' that is canonical or has a principal variable. The computation proceeds by cases:

- if evd' is a variable v and $v : \psi$ is in the context then halt and return n.
- If evd' is **inl** evd_1 and $\psi = \psi_1 \vee \psi_2$ then the computation proceeds with $c(\alpha, \psi_1, evd_1, n)$.
- If evd' is **inr** evd_2 and $\psi = \psi_1 \vee \psi_2$ then the computation proceeds with $c(\alpha, \psi_2, evd_2, n)$.
- If evd' is $\langle evd_1, evd_2\rangle$ and $\psi = \psi_1 \& \psi_2$ then return the maximum of the dovetailed or sequential computation of both $c(\alpha, \psi_1, evd_1, n)$ and $c(\alpha, \psi_2, evd_2, n)$.
- If evd' is $\lambda x.\, evd_1$ and $\psi = \psi_1 \Rightarrow \psi_2$ then since the context $\alpha(n)$ is maximal there is a declaration $v : \psi_1$, so proceed with $c(\alpha, \psi_2, evd_1[x := v], n)$.
- If evd' is $\lambda x.\, evd_1$ and $\psi = \forall x.\, \psi_2$ then in $\alpha(n+1)$ a fresh $d_j : D$ was added so proceed with $c(\alpha, \psi_2[x := d_j], evd_1[x := d_j], n+1)$.
- If evd' has a principal variable v that is the argument to a **decide** operator, then the maximal context specifies that $v =$ **inl** x or $v =$ **inr** y, so replace v and proceed with the computation.
- If evd' has a principal variable v that is the argument to a **spread** operator, then if the maximal context specifies that $v = \langle x, y\rangle$ replace v and proceed with the computation (this must happen if $v : A\&B$ in the context). Otherwise, if $v : \exists z.\, P(z)$ in the context then in the next context, $\alpha(n+1)$, v will be replaced by a pair $\langle d_j, w\rangle$ where $d_j : D$ is new and $w : P(d_j)$, so replace v by $\langle d_j, v\rangle$ to get evd'' and proceed with $c(\alpha, \psi, evd'', n+1)$.
- If evd' has a principal variable v where the principal subterm is $v(d_n)$ then $d_n : D$ is in the context, and since the context is maximal there is a constraint $v(d_n) = w$ in the context, so replace the subterm $v(d_n)$ with w and proceed with the computation.
- Otherwise abort the computation.

Lemma 12. *If evd is uniform evidence for ψ then the computation $c(\alpha, \psi, evd, n+1)$ converges.*

Proof. Any path α through $\mathcal{SM}(\psi)$ defines a model M in which $evd \in M(\psi)$. □

Corollary 4. *The proof procedure terminates and this establishes Theorem 1.*

Proof. For any α, the computation $n = c(\alpha, \psi, evd, 1)$ converges. This defines a bar on the fan $\mathcal{SM}(\psi)$. By Brouwer's theorem, there is a uniform bar N. The length of any branch in the tree of evidence structures produced by the proof procedure is bounded by $c(\alpha, \psi, evd, 1)$ for some path α. Thus the height of the tree of evidence structures is bounded by N. Since it is finitely branching, it is finite. □

We have implemented the proof procedure as a tactic in Nuprl and tested it on a number of examples. We can construct evidence terms from the extracts of Nuprl proofs or construct them by hand. We can then modify the evidence terms using any operators we like so that the resulting term is computationally equivalent to the original. Thus we can introduce abbreviations (which is equivalent to using the CUT rule) and use operators such as π_1 and π_2 (which Nuprl displays as `fst` and `snd` as in ML) and (**if** c **then** a **else** b) that are defined in terms of the primitive **spread** and **decide** operators. In Appendix A we show one such example and describe the implementation of the tactic.

7.1. Observations and corollaries

If evd_1 is uniform evidence for ψ_1 and evd_2 is uniform evidence for $\psi_1 \Rightarrow \psi$ then the application $evd_2(evd_1)$ is uniform evidence for ψ. This observation gives us a semantic proof of cut elimination for first order minimal logic.

Lemma 13. *If $\psi \in \mathcal{MF}(\mathcal{L})$ is provable in minimal logic with the cut rule ($\vdash_{MLC} \psi$) then $\vdash_{ML} \psi$.*

Proof. The evidence term extracted from the proof $\vdash_{MLC} \psi$ is uniform evidence for ψ. By Theorem 1, $\vdash_{ML} \psi$. □

8. Concluding discussion

One classical approach to first-order completeness is based on systematic search for counter examples to a conjecture. Validity of the conjecture is the reason the search fails – halting with a proof. This approach is well illustrated in Smullyan's enduringly valued monograph *First-Order Logic* [72] and Fitting's monograph [27] where the method is adapted to *iFOL* with Kripke semantics. Both Smullyan and Fitting use tableaux proofs which go back to the work of Beth [10,11].[18] Like all other classical proofs of completeness, these are not constructively valid, but they are nearly constructive as Underwood showed in her thesis [82].

We hope that our results will add more weight to the notion that there is a deep connection between proving a theorem and writing a program. We have long stressed this idea in papers treating *proofs as programs* [1,7,20] and conversely *programs as proofs*, additionally in papers treating formal constructive *mathematics as a programming language* [18,20] where types subsume data types. Here we are treating *iFOL* as an abstract *dependently-typed* programming language where formulas are specifications given by dependent types.

Acknowledgements

We are very pleased that this article appears in an issue dedicated to Sergei Artemov, a colleague with whom we worked on constructive logic for many years and whose insights and results have influenced the

[18] Beth invented *semantic tableau* as a bridge from semantics to proofs; we use uniform realizers and their *evidence structures*.

```
⊢ ∀[A,D:Type]. ∀[R,Eq:D → D → ℙ].
    ((∀x,y,z:D.  (R[x;y] ⇒ (R[y;z] ∨ Eq[y;z]) ⇒ R[x;z]))
    ⇒ (∀x:D. (R[x;x] ⇒ A))
    ⇒ (∀x:D. ∃y:D. R[x;y])
    ⇒ (∃m:D. ∀x:D. ((Eq[x;m] ⇒ A) ⇒ R[x;m]))
    ⇒ A)

BY EvidenceTac ⌜λTrans,Irr,Unbdd,MxEx.
                  let m = fst(MxEx) in
                   let bounds = snd(MxEx) in
                   let y,ygtr = Unbdd m
                   in let loop = Trans m y m ygtr in
                       let F = λx.(Irr m (loop x)) in
                       F (inl (bounds y (λeq.(F (inr eq )))) )⌝.
   THENA Auto
```

Fig. 3. Example minimal logic proof from evidence.

```
let EvidenceTac evd  =
   - helper functions here -
letrec evdProofTac M evd  p =
  let op = opid_of_term evd in
  if member op "variable pair inl inr lambda" then
      canonical op M evd p
  else
  let t = get_principal_arg_with_context evd in
  if is_variable_term (subtermn 1 t) then
   let op = opid_of_term t in
   if member op "spread decide callbyvalue apply" then
       noncanonical op t M evd p
   else (AddDebugLabel ‘arg not reducible‘ p)
  else let evd’ = apply_conv (ComputeToC []) evd in
        if alpha_equal_terms evd’ evd then Id p
        else evdProofTac M evd’ p

in Repeat UniformCD
   THEN evdProofTac [] evd
;;
```

Fig. 4. Tactic code for Proof from Uniform Evidence.

development of the Nuprl proof assistant and its type theory. Our results were first presented at the 2012 conference on *The Constructive in Logic and Applications* in honor of Sergei.

The authors would also like to thank the referees for many perceptive comments, suggestions, and corrections, especially for suggesting that we interpret Wim Veldman's semantics in our context.

Appendix A

This example shows how equality can be represented as an atomic relation symbol. The formula states (in minimal logic) that an irreflexive, transitive relation that is unbounded cannot have a maximal element. We have introduced a number of abbreviations into the evidence term to illustrate the fact that the proof procedure does not require normalized terms.

The tactic `EvidenceTac` is shown in Fig. 4. It uses the evidence to generate the proof. In Nuprl, some of the primitive rules of minimal logic (hypothesis, and, or, implies, forall, exists introduction and elimination) create auxiliary subgoals to show that the rules have been applied to well-formed propositions. In the proof in Fig. 3 the tactic `THENA Auto` is used to prove these auxiliary goals.

The basic structure of the tactic is to take off the uniform quantifiers and then start the proof procedure from evidence. If the evidence is canonical it uses one of the rules for that case, otherwise if there is a principal variable it uses one of the rules for non-canonical evidence, and otherwise it computes the evidence term (Figs. 5–7).

```
let UniformCD p  = if is_term ‘uall‘ (concl p)
                   then  (D 0 THENA Auto) p
                   else  Fail p in
let mk_cbv_pair t1 t2 =
    subst [‘x’,t1;‘y’,t2] ⌜let a := x in
                           <a, y>⌝ in
let mk_cbv_ap fun arg =
    subst [‘arg’,arg;‘f’,fun] ⌜let a := arg in
                               f a⌝ in
let do_update v pattern redex result evd M =
     let sub = [v, pattern] in
     subst sub (replace_subterm redex result evd),
     map (\(ap,val). (ap, subst sub val)) M in
let lookup M t =
   let test (ap, val) =
     if alpha_equal_terms ap t then val else fail in
   inl (first_value test M) ? inr () in
```

Fig. 5. Code for helper functions.

```
canonical op M evd p =
 if op = ‘variable‘ then
      let x = dest_variable evd in
        let n = get_number_of_declaration p x in NthHyp n p
 else if op = ‘pair‘ then
    let evd1,evd2 = dest_pair evd in
    if is_term ‘and‘ (concl p)  then
     (D 0 THENL [evdProofTac M evd1; evdProofTac M evd2]) p
    else if is_term ‘variable‘ evd1 then
         (With evd1 (D 0) THENM (evdProofTac M evd2)) p
    else (evdProofTac M (mk_cbv_pair evd1 evd2)) p
 else if op = ‘inl‘ then
      (OrLeft THENM (evdProofTac M (dest_inl evd))) p
 else if op = ‘inr‘ then
      (OrRight THENM (evdProofTac M (dest_inr evd))) p
 else let x,t = dest_lambda evd in
      let z = maybe_new_var x (declared_vars p) in
      let evd1 = if z = x then t else subst [x, mvt z] t in
      SeqOnM [D 0 ; RenameVar z (-1); evdProofTac M evd1] p
```

Fig. 6. Code for canonical evidence.

The helper code includes the tactic for taking off a uniform quantifier, functions for forming the call-by-value pair and apply terms, and code for substituting a pattern into the evidence and constraints (here called the model) in order to eliminate a redex from the non-canonical evidence. The lookup function checks for the existence of a constraint on a given apply term from the evidence. The code for the canonical case comes from the rules in Fig. 1. In each case, the corresponding proof rule of the logic is invoked with the tactic `D 0`. To make life easier for the users, Nuprl has organized all the primitive rules into one tactic named `D` (for decompose). The number `0` indicates that we are applying a primitive rule to decompose the conclusion of the sequent rather than one of the hypotheses. This is because the canonical evidence always indicates that the next proof step is an introduction rule.

The code for the non-canonical case comes from the rules in Fig. 2. In these cases we use an elimination rule, indicated by the fact that the tactic calls on `D n` where `n` is the hypothesis number for the declaration of the principal variable. The code for the apply case is shown in Fig. 8. When the type of the declared variable (`T = h n p`) is an implies we use the rule $\Rightarrow$ APPLY that adds a constraint that the declared function is a constant function. In this implementation we substitute the constant function for the variable and eliminate it entirely. We can prove that this results in behavior that is equivalent to the derivation rules.

```
noncanonical op t M evd p =
 if op = ‘spread‘  then
     let t1,bt = dest_spread t in
     let v = dest_variable t1 in
     let [x;y],body = rename_bvs p bt in
     let pattern = mk_pair_term (mvt x) (mvt y) in
     let evd1, M’ = do_update v pattern t body evd M in
     let n = get_number_of_declaration p v in
       Seq [D n
           ; RenameVar x n
           ; RenameVar y (n+1)
           ; evdProofTac M’ evd1] p
  else if op = ‘decide‘  then
     let t1, bt1, bt2 = dest_decide t in
     let v = dest_variable t1 in
     let [x],case1 = rename_bvs p bt1 in
     let pattern1 = mk_inl_term (mvt x) in
     let evd1, M1 = do_update v pattern1 t case1 evd M in
     let [y],case2 = rename_bvs p bt2 in
     let pattern2 = mk_inr_term (mvt y) in
     let evd2, M2 = do_update v pattern2 t case2 evd M in
     let n = get_number_of_declaration p v in
      (D n THENL [ RenameVar x n THEN  evdProofTac M1 evd1
                 ; RenameVar y n THEN  evdProofTac M2 evd2
                 ]) p
  else if op = ‘callbyvalue‘  then
     let kind, arg, ([x], B) = dest_callbyvalue t in
     let B’ = subst [x, arg] B in
     evdProofTac M (replace_subterm t B’ evd ) p
  else  apply_case t M evd p
```

Fig. 7. Code for non-canonical evidence.

```
apply_case t M evd p =
  let fun,arg = dest_apply t in
  let v = dest_variable fun in
  let n = get_number_of_declaration p v in
  let T = h n p in
  if is_term ‘implies‘ T  then
    let x = maybe_new_var ‘x’ (declared_vars p) in
    let pattern = mk_lambda_term ‘z’ (mvt x) in
    let evd1, M’ = do_update v pattern t (mvt x) evd M in
    ((D n  THEN Fold ‘implies‘ n)
       THENL [ evdProofTac M arg
             ; RenameVar x (-1) THEN  evdProofTac M’ evd1]) p
  else if is_term ‘all‘ T then
    if is_variable_term arg then
      let w = dest_variable arg in
      let ans = lookup M t in
      if isl ans then
        evdProofTac M (replace_subterm t (outl ans) evd) p
      else
        let x = maybe_new_var ‘x’ (declared_vars p) in
        let evd1 = replace_subterm t (mvt x) evd in
        let M’ = (t , (mvt x)).M in
        (SimpleInstHyp arg n THENM
          (Seq [ RenameVar x (-1); evdProofTac M’ evd1])) p
   else let evd’ = replace_subterm t (mk_cbv_ap fun arg) evd in
        evdProofTac M evd’ p
 else (AddDebugLabel ‘fun in apply has wrong type‘ p)
```

Fig. 8. Code for apply case of non-canonical evidence.

References

[1] S. Allen, M. Bickford, R. Constable, R. Eaton, C. Kreitz, L. Lorigo, E. Moran, Innovations in computational type theory using Nuprl, J. Appl. Log. 4 (2006) 428–469.

[2] S. Artemov, Uniform provability realization of intuitionistic logic, modality and lambda-terms, Electron. Notes Theor. Comput. Sci. 23 (1999), http://www.elsevier.nl/entcs/.

[3] M. van Atten, On Brouwer, Wadsworth Philosophers Series, Thompson/Wadsworth, Toronto, Canada, 2004.
[4] R.C. Backhouse, P. Chisholm, G. Malcolm, E. Saaman, Do-it-yourself type theory, Form. Asp. Comput. 1 (1989) 19–84.
[5] H.P. Barendregt, Lambda Calculi with Types, Handbook of Logic in Computer Science, vol. 2, Oxford University Press, 1992, pp. 118–310.
[6] J.L. Bates, A logic for correct program development, Ph.D. thesis, Cornell University, 1979.
[7] J.L. Bates, R.L. Constable, Proofs as programs, ACM Trans. Program. Lang. Syst. 7 (1985) 53–71.
[8] H. Benl, U. Berger, H. Schwichtenberg, et al., Proof theory at work: Program development in the Minlog system, in: W. Bibel, P.G. Schmitt (Eds.), Automated Deduction, vol. II, Kluwer, 1998.
[9] Y. Bertot, P. Castéran, Interactive Theorem Proving and Program Development; Coq'Art: The Calculus of Inductive Constructions, Texts in Theoretical Computer Science, Springer-Verlag, 2004.
[10] E.W. Beth, Semantical considerations on intuitionistic mathematics, Indag. Math. (N.S.) 9 (1947) 572–577.
[11] E.W. Beth, Semantic construction of intuitionistic logic, Koninklijk Nederlandse Akademie 19 (11) (1957) 357–388.
[12] E.W. Beth, The Foundations of Mathematics, North-Holland, Amsterdam, 1959.
[13] M. Bickford, R.L. Constable, Polymorphic logic, in: Logic, Construction, Computation, Ontos Verlag, 2012, Festschrift for Helmut Schwichtenberg.
[14] E. Bishop, Foundations of Constructive Analysis, McGraw Hill, New York, 1967.
[15] A. Bove, P. Dybjer, U. Norell, A brief overview of Agda – a functional language with dependent types, in: S. Berghofer, T. Nipkow, C. Urban, M. Wenzel (Eds.), Theorem Proving in Higher Order Logics, in: LNCS, vol. 5674, Springer, 2009, pp. 73–78.
[16] L. Brouwer, Intuitionism and formalism, Bull. Amer. Math. Soc. 20 (1913) 81–96.
[17] A. Church, The Calculi of Lambda-Conversion, Annals of Mathematical Studies, vol. 6, Princeton University Press, Princeton, 1941.
[18] R.L. Constable, Constructive mathematics and automatic program writers, in: Proceedings of the IFIP Congress, North-Holland, 1971, pp. 229–233.
[19] R.L. Constable, The semantics of evidence (also appeared as Assigning Meaning to Proofs), Constr. Methods Comput. Sci. F55 (1989) 63–91.
[20] R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P. Panangaden, J.T. Sasaki, S.F. Smith, Implementing Mathematics with the Nuprl Proof Development System, Prentice-Hall, NJ, 1986.
[21] T. Coquand, G. Huet, The calculus of constructions, Inform. and Comput. 76 (1988) 95–120.
[22] T. Coquand, C. Paulin, Inductively defined types, in: P. Martin-Löf, G. Mints (Eds.), Conference on Computer Logic, in: Lecture Notes in Computer Science, vol. 417, Springer-Verlag, 1988, pp. 50–66.
[23] T. Coquand, J.M. Smith, An application of constructive completeness, in: Proceedings of the Workshop TYPES '95, Springer-Verlag, 1995, pp. 76–84.
[24] H.B. Curry, Functionality in combinatory logic, Proc. Nat. Acad. Sci. 20 (1934) 584–590.
[25] H.B. Curry, R. Feys, W. Craig, Combinatory Logic, vol. I, Studies in Logic and the Foundations of Mathematics, North-Holland, Amsterdam, 1958.
[26] M. Dummett, Elements of Intuitionism, Oxford Logic Series, Clarendon Press, 1977.
[27] M. Fitting, Intuitionistic Model Theory and Forcing, North-Holland, Amsterdam, 1969.
[28] J.Y. Girard, P. Taylor, Y. Lafont, Proofs and Types, Cambridge Tracts in Computer Science, vol. 7, Cambridge University Press, 1989.
[29] M. Gordon, R. Milner, C. Wadsworth, Edinburgh LCF: A Mechanized Logic of Computation, Lecture Notes in Computer Science, vol. 78, Springer-Verlag, New York, 1979.
[30] C.C. Green, An application of theorem proving to problem solving, in: IJCAI-69, Washington, DC, 2013, pp. 219–239.
[31] T.G. Griffin, Notational definition and top-down refinement for interactive proof development systems, Ph.D. thesis, Cornell University, 1988.
[32] A. Grzegorczyk, A philosophically plausible interpretation of intuitionistic logic, Indag. Math. (N.S.) 26 (1964) 596–601.
[33] C.A. Gunter, Semantics of Programming Languages: Structures and Techniques, Foundations of Computing Series, MIT Press, 1992.
[34] R. Harper, F. Honsell, G. Plotkin, A framework for defining logics, J. Assoc. Comput. Mach. 40 (1993) 143–184. A revised and expanded version of '87 paper.
[35] A. Heyting, Mathematische Grundlagenforschung. Intuitionismus. Beweistheorie, Springer, Berlin, 1934.
[36] A. Heyting, Infinitistic methods from a finitist point of view, in: Proceedings of the Symposium on Foundations of Mathematics, International Mathematical Union and Mathematical Institute of the Polish Academy of Sciences, Pergamon Press, Warsaw, 1959, pp. 185–192.
[37] A. Heyting, Intuitionism, An Introduction, North-Holland, Amsterdam, 1966.
[38] A. Heyting, E.J. Brouwer (Eds.), Collected Works, vol. 1, North-Holland, Amsterdam, 1975, pp. 1–6 (see on the foundations of mathematics 11–98).
[39] J. Hickey, A. Nogin, R.L. Constable, B.E. Aydemir, E. Barzilay, Y. Bryukhov, R. Eaton, A. Granicz, A. Kopylov, C. Kreitz, V.N. Krupski, L. Lorigo, S. Schmitt, C. Witty, X. Yu, MetaPRL — A modular logical environment, in: D. Basin, B. Wolff (Eds.), Proceedings of the 16th International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2003, in: Lecture Notes in Computer Science, vol. 2758, Springer-Verlag, 2003, pp. 287–303.
[40] W. Howard, The formulas-as-types notion of construction, in: To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism, Academic Press, NY, 1980, pp. 479–490.
[41] D.J. Howe, Equality in lazy computation systems, in: Proceedings of the 4th IEEE Symposium on Logic in Computer Science, Asilomar Conference Center, Pacific Grove, California, IEEE Computer Society Press, 1989, pp. 198–203.
[42] D.J. Howe, On computational open-endedness in Martin-Löf's type theory, in: Proceedings of the 6th Symposium on Logic in Computer Science, Vrije University, Amsterdam, The Netherlands, IEEE Computer Society Press, 1991, pp. 162–172.

[43] D. Ilik, Continuation-passing style models complete for intuitionistic logic, Ann. Pure Appl. Logic 164 (2013) 651–662.
[44] S. Kleene, On the interpretation of intuitionistic number theory, J. Symbolic Logic 10 (1945) 109–124.
[45] S.C. Kleene, Introduction to Metamathematics, D. Van Nostrand, Princeton, 1952.
[46] S.C. Kleene, R.E. Vesley, Foundations of Intuitionistic Mathematics, North-Holland, 1965.
[47] A. Kolmogorov, On the principle of the excluded middle, in: J. van Heijenoort (Ed.), From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931, Harvard University Press, Cambridge, MA, 1967, pp. 414–437.
[48] A.N. Kolmogorov, Zur deutung der intuitionistischen logik, Math. Z. 35 (1932) 58–65.
[49] G. Kreisel, Weak completeness of intuitionistic predicate logic, J. Symbolic Logic 27 (1962) 139–158.
[50] C. Kreitz, The Nuprl Proof Development System, version 5, Reference Manual and User's Guide, Cornell University, Ithaca, NY, 2002, http://www.nuprl.org/html/02cucs-NuprlManual.pdf.
[51] S.A. Kripke, Semantical analysis of intuitionistic logic, in: Formal Systems and Recursive Functions, North-Holland, Amsterdam, 1965, pp. 92–130.
[52] H. Lauchli, An abstract notion of realizability for which intuitionistic predicate calculus is complete, in: J. Myhill, A. Kino, R. Vesley (Eds.), Intuitionism and Proof Theory, North-Holland, Amsterdam, 1970, pp. 227–234.
[53] D. Leivant, Syntactic translations and provably recursive functions, J. Symbolic Logic 50 (1985) 682–688.
[54] J. Lipton, M.J. O'Donnell, Some intuitions behind realizability semantics for constructive logic: Tableau and Läuchli countermodels, Ann. Pure Appl. Logic 81 (1996) 187–239.
[55] P. Martin-Löf, Notes on Constructive Mathematics, Almqvist & Wiksell, Uppsala, 1970.
[56] P. Martin-Löf, An intuitionistic theory of types: Predicative part, in: Logic Colloquium '73, North-Holland, Amsterdam, 1973, pp. 73–118.
[57] P. Martin-Löf, Constructive mathematics and computer programming, in: Proceedings of the Sixth International Congress for Logic, Methodology, and Philosophy of Science, North-Holland, Amsterdam, 1982, pp. 153–175.
[58] P. Martin-Löf, Intuitionistic Type Theory, number 1 in Studies in Proof Theory, Lecture Notes, Bibliopolis, Napoli, 1984.
[59] P. Martin-Löf, An intuitionistic theory of types, in: G. Sambin, J.M. Smith (Eds.), Twenty-Five Years of Constructive Type Theory, in: Oxford Logic Guides, vol. 36, Clarendon Press, Oxford, 1998, pp. 127–172.
[60] D. McCarty, Undecidability and intuitionistic completeness, J. Philos. Logic 25 (1996) 559–565.
[61] D. McCarty, Completeness and incompleteness for intuitionistic logic, J. Symbolic Logic 73 (2008) 1315–1327.
[62] J.C. Mitchell, Foundations of Programming Languages, MIT Press, 1996.
[63] B. Nordström, K. Petersson, J.M. Smith, Programming in Martin-Löf's Type Theory, 1990, Oxford Science Publications, Oxford.
[64] F. Pfenning, C. Schürmann, Twelf — A meta-logical framework for deductive systems, in: H. Ganzinger (Ed.), Proceedings of the 16th International Conference on Automated Deduction, vol. 1632, Trento, Italy, 1999, pp. 202–206.
[65] B.C. Pierce, Types and Programming Languages, MIT Press, 2002.
[66] B.C. Pierce, C. Casinghino, M. Greenberg, V. Sjöberg, B. Yorgey, Software Foundations, Electronic, 2011.
[67] G.D. Plotkin, LCF considered as a programming language, J. Theoret. Comput. Sci. 5 (1977) 223–255.
[68] G.D. Plotkin, A structural approach to operational semantics, Technical Report DAIMI-FN-19, Aarhus University, Aarhus University, Computer Science Department, Denmark, 1981.
[69] H. Raisowa, R. Sikorski, The Mathematics of Metamathematics, Panstowe Wydawnictwo Naukowe, Warsaw, 1963.
[70] A. Ranta, Type-Theoretical Grammar, Oxford Science Publications, Clarendon Press, Oxford, England, 1994.
[71] G. Sambin, J.M. Smith (Eds.), Twenty-Five Years of Constructive Type Theory, Oxford Logic Guides, vol. 36, Clarendon Press, Oxford, 1998.
[72] R.M. Smullyan, First-Order Logic, Springer-Verlag, New York, 1968.
[73] M. Sørensen, P. Urzyczyn, Lectures on the Curry–Howard Isomorphism, Elsevier, 2006.
[74] W.P. van Stigt, Brouwer's Intuitionism, North-Holland, Amsterdam, 1990.
[75] H. de Swart, An intuitionistically plausible interpretation of intuitionistic logic, J. Symbolic Logic 42 (1977) 564–578.
[76] W.W. Tait, Intensional interpretation of functionals of finite type, J. Symbolic Logic 32 (1967) 189–212.
[77] A. Tarski, Der aussangenkalkul und die topologie, Fund. Math. 31 (1938) 103–134.
[78] S. Thompson, Type Theory and Functional Programming, Addison-Wesley, 1991.
[79] A. Troelstra, Realizability, in: S. Buss (Ed.), Handbook of Proof Theory, Elsevier Science, 1998, pp. 407–473.
[80] A. Troelstra, D. van Dalen, Constructivism in Mathematics, An Introduction, vols. I, II, North-Holland, Amsterdam, 1988.
[81] A.S. Troelstra, History of constructivism in the 20th century, ITLI Publ. Ser. ML-91-05 (1991) 1–32.
[82] J.L. Underwood, Aspects of the computational content of proofs, Department of Computer Science TR94-1460, Cornell University, Ithaca, NY, 1994.
[83] W. Veldman, An intuitionistic completeness theorem for intuitionistic predicate calculus, J. Symbolic Logic 41 (1976) 159–166.