Intuitionistic Completeness of First-Order Logic

Robert Constable and Mark Bickford

October 7, 2011

Abstract

We establish completeness for *intuitionistic first-order logic, iFOL*, showing that is a formula is provable if and only if it is *uniformly valid* under the *Brouwer Heyting Kolmogorov (BHK)* semantics, the intended semantics of iFOL. Our proof is intuitionistic and provides an effective procedure *Prf* that converts uniform evidence into a formal first-order proof. We have implemented *Prf*. Uniform validity is defined using the intersection operator as a universal quantifier over the domain of discourse and atomic predicates. Formulas of *iFOL* that are uniformly valid are also intuitionistically valid, but not conversely. Our strongest result requires the Fan Theorem; it can also be proved classically by showing that *Prf* terminates using König's Theorem.

The fundamental idea behind our completeness theorem is that a single evidence term *evd* witnesses the uniform validity of a minimal logic formula F. Finding even one uniform realizer guarantees validity because *Prf*(F, evd) builds a first-order proof of *F*, establishing its uniform validity and providing a purely logical normalized realizer.

We establish completeness for *iFOL* as follows. Friedman showed that *iFOL* can be embedded in minimal logic (*mFOL*). By his transformation, mapping formula A to $Fr(A)$. If A is uniformly valid, then so is $Fr(A)$, and by our Basic Completeness result, we can find a proof of $Fr(A)$ in minimal logic. Then we prove A from $Fr(A)$ in intuitionistic logic by a proof procedure fixed in advance. Our result resolves an open question posed by Beth in 1947.

1 Introduction

1.1 Overview

approaches to completeness We introduce a new approach to completeness questions. It provides the first intuitionistic completeness proof for the intended semantics of intuitionistic logic, a question investigated by Beth starting in 1947 and open ever since. Our result provides an answer, however not the one expected by comparison with Gödel's completeness proof for classical first-order logic. We briefly review previous completeness results below.

We came to our approach because we use on a daily basis the fact that from constructive proofs of a theorem in computational type theory we can automatically extract programs that meet the specification given by the theorem. These programs are evidence for validity of the theorem. For intuitionistic first-order logic (iFOL), a subtheory of type theory, the extracted programs are *uniform* witnesses for validity of the theorems. We call them *uniform realizers.* We can express this uniformity using a universal quantifier defined using the intersection type in computational type theory [1]. Moreover for first-order logic we know that the realizers are not only uniform, but they are in normal form and consist entirely of logical operators. This is a basic fact about the extraction of computational content (see [11, 31]).

In many cases we could see clearly the proof structure in the realizers. This led us to conjecture that iFOL is complete with respect to uniform semantics because uniformity eliminates terms that are not essentially built from the logical operators. It was a longer road to establish this in detail, and we report succinctly on that journey here, giving all of the technical details. In a longer forthcoming article we will provide more motivation, examples, and practical applications under the proposed title *Intuitionistic Completeness of First-Order Logic with Respect to Uniform Evidence Semantics*. There we also explicitly prove some of the basic results about extraction in the simple setting of iFOL.

The common approach to first-order completeness is based on systematic search for counter examples to a conjecture, and 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* [44] and Fitting's monograph [15]. Like all other classical proofs of completeness, these are not constructively valid. We take a very different approach, effectively converting uniform evidence for validity into a proof. We do this by building objects called *evidence structures* that reveal the evidence term layer by layer. For instance, when we see evidence of the form $\lambda(x.b(x))$ for a formula $A \Rightarrow B$, then we add to the context of the evidence structure the assumption that $x : A$ and continue by analyzing $b(x)$ after normalizing it by symbolic computation. This computation reveals the operations that must be performed on the context to expose more of the evidence term $b(x)$. For example, if the assumption A is $A_1 \& A_2$, then we decompose x into $x_1 : A_1$ and $x_2 : A_2$ and substitute the pair $< x_1, x_2 >$ into the logical operator mentioning x in the evidence term we are analyzing. Because the evidence is uniform, the normalization process eliminates any operators on non-logical terms. We can thus convert the operators on evidence terms to proof steps in first-order minimal logic.

Our realizers are *effectively computable* functions operating on data types; we call this approach *Brouwer realizability* or *evidence semantics*. We do not rely on Church's Thesis for any of these results, and according to Kleene [23, 45], our use of the Fan Theorem precludes it.[1]

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, 4, 11] and conversely *programs as proofs*, additionally in papers treating formal constructive *mathematics as a programming language* [9, 11]. Here we are treating iFOL as an abstract programming language where formulas are specifications given by dependent types. We build the proof from the program which is a uniform Brouwer realizer.

intuitionistic model theory This article contributes to an *intuitionistic model theory* as proposed by Beth in 1947 [6] and greatly advanced by Per Martin-Löf [31, 32]. Beth's methods led to *Beth models* and *Kripke models* whose computational meaning is not as strong as in the realizability tradition, even given Veldman's intuitionistic completeness theorem for Kripke models [49]. We work in the realizability tradition started by Kleene, developed further by Martin-Löf, extended and implemented by the PRL Group as reported in the book *Implementing Mathematics* [11] and in numerous doctoral dissertations and articles [1]. This is the tradition framing and motivating our completeness results.

The semantic tradition is grounded in precise knowledge of the underlying computation system and its efficient implementation made rigorous by researchers in programming languages. Our operational semantics of evidence terms follows the method of *structured operational semantics* of Plotkin [40, 41]. The few basic results about programming language semantics we mention can be found in the comprehensive textbooks on the subject [36, 38]. Many results from this theory are now being formalized in proof assistants and applied directly to building better languages and systems [39].

[1] Nuprl was designed to use an open-ended notion of effective computability from the start [11].

1.2 Background

Classical first-order logic, FOL Tarski's semantics [46] for classical first-order formulas faithfully captures their intuitive *truth-functional interpretation.* Gödel proved his classical completeness theorem for first-order logic with respect to to this intended semantics, showing that an FOL formula is provable if and only if truth functionally valid. This has become a fundamental result in logic which is widely taught to undergraduates. There are many excellent textbook proofs such as [44].

Intuitionistic first-order logic, iFOL The BHK semantics for iFOL is the intended semantics, faithful to the intuitionistic conception of knowledge. In contrast to the classical situation, there has been no intuitionistic completeness proof with respect to the intended semantics. To explain this contrast, we look briefly at the origin of intuitionism. At nearly the same time that a truth-functional approach to logic was being developed by Frege [16] and Russell [43], *circa* 1907, Brouwer [19, 48] imagined a very different meaning for mathematical statements and thus for logic itself. Brouwer's meaning is grounded in the *mental constructions* that cause an *individual mathematician* to know that mathematical objects can be created with certain properties.

Brouwer developed a very rich informal model of computation in terms of which he could interpret most concepts and theorems of mathematics, including from set theory. Brouwer's approach anticipated a precise meaning that Church, Turing, and now legions of computer scientists give to mathematical statements whose meaning is grounded in computations executed by modern digital computers. Brouwer's intuitive interpretation has come to be known among logicians as *Brouwer, Heyting, Kolmogorov (BHK) semantics* when applied to formal intuitionistic logical calculi, as first done by Heyting [20] and Kolmogorov [24]. In 1945 Kleene [22, 45] invented his *realizability* semantics for intuitionistic number theory in order to connect Brouwer's informal notion of computability to the precise theory of partial recursive functions. He used indexes of general recursive functions as *realizers*, and by 1952 [21] he viewed realizability as a formal account of BHK semantics under the assumption of Church's Thesis.

By 1982 Martin-Löf [31, 32] building on the work of Kleene refined the BHK approach and raised it to the level of a *semantic method* for constructive logics grounded in structured operational semantics [41]. Martin-Löf often referres to BHK as the *propositions as types principle.* In computer science other terminology is "proofs as programs" or the "Curry-Howard isomorphism". 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 [30] based on a topological model of Borel sets in the Cantor Space.

This semantics plays an important role in the business of building *correct by construction* software and in the semantics of the constructive logics such as *Computational Type Theory* (CTT) [1, 11], *Intuitionistic Type Theory* (ITT) [31, 32], *Intensional-ITT* [8, 33, 37], the *Calculus of Inductive Constructions* (CIC)[5], and *Logical Frameworks* such as Edinburgh LF [18]. All of these logics are implemented by *proof assistants* such as Agda, Coq, MetaPRL, Nuprl, and Twelfth among others.

Previous completeness theorems A constructive completeness theorem for iFOL and mFOL is a very strong result because it says that if we know that a formula is valid, thus true in every possible model, *then we can effectively find a first-order proof based on that knowledge.* This seems highly unlikely as the sixty four year long investigation of the problem has shown. In all previous work, the idea is to try to construct a proof and use the evidence for truth to argue that the proof construction must succeed. Classically this requires König's Lemma, and constructively some use of Markov's Principle or the Fan Theorem or something of that kind. Those efforts do not try to use the information that $\forall\mathcal{M} : Model. \models F$ to build the proof. Nevertheless, our results show exactly how to build the proof from *uniform evidence* for validity, which is a single object. Moreover, we can actually execute our result using a tactic executed by the Nuprl prover [11, 1, 26]. We give that procedure in the Appendix.

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 [13, 14, 25, 30, 49, 35]. Some efforts led to apparently more technically tractable semantic alternatives to BHK such as *Beth models* [7, 49], *Kripke models* [27], topological models [46, 42, 17, 30, 12], intuitionistic model theoretic validity [47], and provability logic [2]. Dummett [14] discusses completeness issues extensively. The value of developing a precise mathematical semantics for intuitionistic mathematics in the spirit of Tarski's work dates at least from Beth 1947 [6] with technical progress by 1957 [7]. So the completeness issue has been identified yet unsettled for sixty four years. An important early attempt to base completeness on BKH is the (nonconstructive) work of Läuchli [28, 29] 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. We do.

The closest correspondingly faithful constructive completeness theorem for *intuitionistic validity* is by Friedman in 1975 (presented in [47]), 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 is from 1998 by Artemov using *provability logic* [3]. Results suggest how delicate completeness theorems are since constructive completeness with respect to full intuitionistic validity contradicts Church's Thesis [25, 47] and implies *Markov's Principle* as well [34, 35].[2]

1.3 Summary of Results

Results in this article We first review *evidence semantics.* [3]. Using evidence semantics, we then 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 semantics because we can establish uniform validity by *exhibiting a single polymorphic object.* For example, the propositional formula $A \Rightarrow A$ is uniformly valid exactly when there is an object in the intersection of all evidence types for this formula for each possible choice of A among the type of propositions, $\mathbb{P}$. We write this intersection as $\forall[A : \mathbb{P}].A \Rightarrow A$ or as $\bigcap A : \mathbb{P}.A \Rightarrow A$.[4] 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.[5] Truth tables do this for classical propositional logic. Unlike for classical first-order 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 $\lambda(h.\lambda(x.\lambda(p.h(< x, p >))))$ establishes the uniform minimal (logic) validity of

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

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.*

[2] Church's Thesis is not an issue for us because we do not assume it.

[3] We can extend this semantics to classical logic if we allow oracle computations [10] to justify the law of excluded middle, $P \vee \sim P$, with an operator *magic*(P). We make some observations about classical logic based on this *classical evidence semantics*

[4] 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 $\mathbb{P}_i$.

[5] Contrast this with the kind of evidence need for classical or intuitionistic *model theoretic validity.* In those cases, we need a whole class of models to witness validity of a single formula.

Uniform validity also raises the semantic problem that forces us to consider minimal logic first. Consider the intuitionistically valid assertion $False \Rightarrow A$ for any proposition A. One semantic object that witnesses uniform validity is $\lambda(x.x)$, and other witnesses for uniform validity include any constant functions, say $\lambda(x.17)$ or even a *diverging* term such as *div*. The claim being made is that if x belongs to the evidence type for $False$, then 17 or *div* belongs to the evidence type for A.[6] This claim is vacuously true since no element can be evidence for $[False]$ whose evidence is the empty type. From the constant function with value 17, $\lambda(x.17)$, we cannot reconstruct the proof. In minimal logic, we don't have the atomic propositional constant $False$, we use instead the *arbitrary* propositional constant $\perp$ whose interpretation allows non-empty types as well as empty ones. For the same reason, avoiding vacuous hypotheses, we require that all domains of discourse for minimal logic can be non-empty.

Discussion Our results also suggest why completeness with respect to satisfiability in all constructive models, let alone all intuitionistic models, is unlikely (even impossible according to McCarty [34, 35]); such completeness is unlikely because we show that provability captures exactly *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 the intended BHK semantics, the semantics that enables strong connections to computer science.

2 The main theorems

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 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 [7], $\mathcal{MF}(\mathcal{L})$, are the formulas in $\mathcal{F}(\mathcal{L})$ that do not use either negation or False.*

In type theory, the propositions, $\mathbb{P}$, are identified with types. 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 $M(D)^{n_i} \rightarrow \mathbb{P}$. We write $\mathcal{S}(\mathcal{L})$ for the type[8] 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.*

Definition 3. *Given $M \in \mathcal{S}(\mathcal{L})$ that has been extended to map the variables $V_0 \subseteq Var$ into $M(D)$,*

[6] We can use the fixed point combinator, say $\mathbf{Y}$ or fix to define div. For instance, $fix(\lambda(x.x))$ computes to itself, where fix is an operator such as the $\mathbf{Y}$ combinator $\lambda(f.ap(\lambda(x.ap(f;ap(x;x)));\lambda(x.ap(f;ap(x;x)))))$.

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

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

we extend the mapping M to all formulas in $\mathcal{F}(\mathcal{L})$ with free variables in V_0 by:

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

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 $M \models \psi$.

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 $M : \mathcal{S}(\mathcal{L}) \rightarrow M(\psi)$ 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 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:

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

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 Nuprl, Minlog, Agda, and Coq 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.

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

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

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

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

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

Proof. Since the A-transform satisfies $\vdash_{IL} ((\psi \vee A) \Rightarrow\ A(\psi))$ uniform evidence for $\psi \vee A$ can be converted into uniform evidence for $A(\psi)$. The A-transform also satisfies $\vdash_{ML} A(\psi) \Rightarrow \vdash_{IL} \psi$. Thus,

$$\begin{aligned} \forall[M : \mathcal{S}(\mathcal{L})].\, M \models \psi &\quad \Rightarrow \\ \forall[M : \mathcal{S}(\mathcal{L})].\, M \models (\psi \vee A) &\quad \Rightarrow \\ \forall[M : \mathcal{S}(\mathcal{L})].\, M \models A(\psi) &\quad \Rightarrow \\ \vdash_{ML} A(\psi) &\quad \Rightarrow \\ \vdash_{IL} \psi &\quad \Rightarrow \\ \forall[M : \mathcal{S}(\mathcal{L})].\, M \models \psi & \end{aligned}$$

□

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 constructive logic, so we must avoid the use of excluded middle for propositions that are not decidable and in particular we can not assume the proposition $evd \in M(\psi)$ is decidable. Because of this, we will need the concept that evidence term *evd* is *consistent with* the type $M(G)$. One notion of consistency that is sufficent for our proof is that there is no structure M for which $evd \notin M(G)$. However, the resulting proof is logically more complex than the one we give below where consistency is based on interpeting the types in *finitary* structures.

3 Finitary types and structures

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 $0 \le i < k$).*

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

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

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

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

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

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

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 $Term \to \bar{T}$ 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 $true + 5$ or a primitive. The term **abort** does not converge to a value. We use this to construct simple examples of of finitary types.

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

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

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

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

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 used by Nuprl, but our proof could easily be modified to work for type theories based on different primitive computations [9]

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

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

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

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

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

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

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

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

[9] 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 accomodate such differences.

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

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

The operation $f(\mathbf{val}\, x)$ 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 $(\lambda a.\langle a, y\rangle)(\mathbf{val}\, x)$ as $\langle \mathbf{val}\, x, y\rangle$. This operation forms a pair only after the first component has been evaluated.*

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*

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

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 $V_0 \subset Var$ into a finitary type T_0 we define a finitary $\mathcal{L}$-structure $M_{triv}(\sigma)$ by*

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

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. □

4 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 $d_i \in Var$ (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 orginal goal ψ and v_i is a variable chosen from another set of variables $Var' = v_0, v_1, v_2, \ldots$ disjoint from $Var = \{d_0, d_1, \ldots\}$. The context H will also contain *constraints* of the form $f(\mathbf{val}\, d) = t$ where $f \in Var'$ 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:*

   - $\langle ptn_1, ptn_2 \rangle : (A \times B)$
   - $\mathbf{inl}\ ptn_1 : (A + B)$
   - $\mathbf{inr}\ ptn_2 : (A + B)$

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

1. *$d : D$ where $d \in Var$.*

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. □

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.

5 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 it matches one of the sixteen derivation rules.

3. This defines a recursive procedure on evidence structures that results is 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 figure 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.*

Lemma 5. *The rules in figure 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.

$$\textsc{$\wedge$pair}\quad \frac{H \models G_1 \wedge G_2, \langle evd_1, evd_2\rangle}{H \models G_1, evd_1 \qquad H \models G_2, evd_2} \qquad\qquad \textsc{$\exists$pair}\quad \frac{H \models \exists x.\, G, \langle evd_1, evd_2\rangle}{H \models \exists x.\, G, \langle \mathbf{val}\, evd_1, evd_2\rangle}$$

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

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

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

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

Figure 1: Rules for evidence structures with canonical evidence.

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

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

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,

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

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

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*

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

$$principal(\mathbf{decide}(d; x.a; y.b)) = principal(d)$$

$$principal(\mathbf{spread}(p; x, y.b)) = principal(p)$$

$$principal(f(b)) = principal(f)$$

$$principal(f(\mathbf{val}\, b)) = principal(b)$$

$$principal(\langle \mathbf{val}\, a, b\rangle) = principal(a)$$

$$principal(t) = t,\ otherwise$$

VAR

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

DECIDE

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

∧SPREAD

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

∃SPREAD

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

APPLY CONST

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

⇒APPLY

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

∀APPLY

$$\frac{f : \forall z.\, P \in H \models G, evd(\underline{f}(t))}{H \models G, evd(f(\mathbf{val}\, t))}$$

APPLY MODEL

$$\frac{f(\mathbf{val}\, d) = t \in H \models G, evd(f(\mathbf{val}\, \underline{d}))}{H \models G, evd(t)}$$

∀CBV

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

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 ⇒APPLY and CBV NEW are fresh.

Figure 2: Rules for evidence structure with non canonical evidence.

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

The rules shown in figure 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 figure 2 are unique, stratified constraints.*

Proof. Only the rules ⇒APPLY and ∀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 ⇒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 fresh, and substituting a pattern for a variable in a pattern results in a pattern, so all the constraints remain unique, stratified patterns. □

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

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

⇒APPLY

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

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 ⇒APPLY

Consider next the rule

∀CBV

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

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 ∀CBV is a valid rule.

Consider next the rule

∃SPREAD

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

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

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

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 figure 1 and figure 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 figure 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 figure 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 the Appendix) 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.

6 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

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

Each rule in Figure 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 Figure 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 figures 1 and 2 are constant domain except for the rules ∀λ and ∃SPREAD.

Definition 19. *A derivation is a* ψ*-deriviation 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 $\mathbf{inl}\ x$ for $x : A$ or by $\mathbf{inr}\ y$ for $y : B$. Declarations of the form $p : A \wedge B$ 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 $\exists$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 alpha) 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 $\psi = \psi_1 \wedge \psi_2$ 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 $v =$* **inl** *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 $v : A \wedge B$ 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 establishesTheorem 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 8 we show one such example and describe the implementation of the tactic.

7 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. ByTheorem 1, $\vdash_{ML} \psi$ □

8 Appendix

```
⊢ ∀[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
```

Figure 3: Example minimal logic proof from evidence

This example shows how equality can be represented as an atomic relation symbol. The formula states (in minimal logic) that an irreflexive, transtitive 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 Figure 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 auxilliary subgoals to show that the rules have been applied to well-formed propositions. In the proof in Figure 3 the tactic `THENA Auto` is used to prove these auxilliary goals.

```
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
;;
```

Figure 4: Tactic code for Proof from Uniform Evidence

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. The helper code include the tactic for taking off a uniform quantifier,

```
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
```

Figure 5: Code for helper functions

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.

```
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
```

Figure 6: Code for canonical evidence

The code for the canonical case comes from the rules in Figure 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.

```
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
```

Figure 7: Code for non-canonical evidence

The code for the non-canonical case comes from the rules in Figure 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 Figure 8. When the type of the declared variable (`T = h n p`) is an implies we use the rule ⇒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.

```
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)
```

Figure 8: Code for apply case of non-canonical evidence

References

[1] Stuart Allen, Mark Bickford, Robert Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and Evan Moran. Innovations in computational type theory using Nuprl. *Journal of Applied Logic*, 4(4):428–469, 2006.

[2] Sergei Artemov. On explicit reflection in theorem proving and formal verification. In Harald Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 267–281, Berlin, July 7–10 1999. Trento, Italy.

[3] Sergei Artemov. Uniform provability realization of intuitionistic logic, modality and lambda-terms. *Electronic Notes on Theoretical Computer Science*, 23(1), 1999. `http://www.elsevier.nl/entcs/`.

[4] J. L. Bates and Robert L. Constable. Proofs as programs. *ACM Transactions of Programming Language Systems*, 7(1):53–71, 1985.

[5] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development; Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. Springer-Verlag, 2004.

[6] Evert W. Beth. Semantical considerations on intuitionistic mathematics. *Indagationes mathematicae*, 9:572 – 577, 1947.

[7] Evert W. Beth. Semantic construction of intuitionistic logic. *Koninklijke Nederlandse Akademie van Wetenschappen, Mededelingen, Nieuwe Reeks*, 19 (11):357 – 388, 1957.

[8] Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of Agda – a functional language with dependent types. In Christian Urban Makarius Wenzel Stefan Berghofer, Tobias Nipkow, editor, *LNCS 5674, Theorem Proving in Higher Order Logics*, pages 73–78. Springer, 2009.

[9] Robert L. Constable. Constructive mathematics and automatic program writers. In *Proceedings of the IFIP Congress*, pages 229–233. North-Holland, 1971.

[10] Robert L. Constable. The Semantics of Evidence (Cornell Technical Report TR85-684 also appeared as Assigning Meaning to Proofs). *Constructive Methods of Computing Science*, F55:63–91, 1989.

[11] Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the* Nuprl *Proof Development System.* Prentice-Hall, NJ, 1986.

[12] Thierry Coquand and Jan M. Smith. An application of constructive completeness. In *In Proceedings of the Workshop TYPES '95*, pages 76–84. Springer-Verlag, 1995.

[13] H.C.M. de Swart. An intuitionistically plausible interpretation of intuitionistic logic. *Journal of Symbolic Logic*, 42:564 – 578, 1977.

[14] Michael Dummett. *Elements of Intuitionism.* Oxford Logic Series. Clarendon Press, 1977.

[15] Melvin Fitting. *Intuitionistic model theory and forcing.* North-Holland, Amsterdam, 1969.

[16] Gottlob Frege. Begriffsschrift, a formula language, modeled upon that for arithmetic for pure thought. In J. van Heijenoort, editor, *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, pages 1–82. Harvard University Press, Cambridge, MA, 1967.

[17] A. Grzegorczyk. A philosophically plausible interpretation of intuitionistic logic. *Indag Math*, 26:596 – 601, 1964.

[18] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993. A revised and expanded verion of '87 paper.

[19] A. Heyting, editor. *L. E. J. Brouwer. Collected Works*, volume 1. North-Holland, Amsterdam, 1975. (see On the foundations of mathematics 11-98.).

[20] Arend Heyting. *Mathematische Grundlagenforschung. Intuitionismus.Beweistheorie*. Springer, Berlin, 1934.

[21] S. C. Kleene. *Introduction to Metamathematics.* D. Van Nostrand, Princeton, 1952.

[22] S.C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10:109 – 124, 1945.

[23] S.C. Kleene and R.E. Vesley. *The Foundations of Intuitionistic Mathematics.* North-Holland, Amsterdam, 1965.

[24] A. N. Kolmogorov. Zur deutung der intuitionistischen logik. *Mathematische Zeitschrift*, 35:58–65, 1932.

[25] G. Kreisel. Weak completeness of intuitionistic predicate logic. *Journal of Symbolic Logic*, 27:139–158, 1962.

[26] Christoph Kreitz. The Nuprl Proof Development System, version 5, Reference Manual and User's Guide. Cornell University, Ithaca, NY, 2002.

[27] Saul A. Kripke. Semantical analysis of intuitionistic logic. In *Formal Systems and recursive functions*, pages 92–130. North-Holland, Amsterdam, 1965.

[28] H. Läuchli. An abstract notion of realizability for which intuitionistic predicate calculus is complete. In *Intuitionism and Proof Theory*, pages 227–34. North-Holland, Amsterdam, 1970.

[29] James Lipton and Michael J. O'Donnell. Some intuitions behind realizability semantics for constructive logic: Tableau and Läuchli countermodels. *Annals of Pure and Applied Logic*, 81:187 – 239, September, 1996.

[30] Per Martin-Löf. *Notes on Constructive Mathematics.* Almqvist & Wiksell, Uppsala, 1970.

[31] Per Martin-Löf. Constructive mathematics and computer programming. In *Proceedings of the Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, Amsterdam, 1982. North Holland.

[32] Per Martin-Löf. *Intuitionistic Type Theory.* Number 1 in Studies in Proof Theory, Lecture Notes. Bibliopolis, Napoli, 1984.

[33] Per Martin-Löf. An intuitionistic theory of types. In Giovanni Sambin and Jan M. Smith, editors, *Twenty-Five Years of Constructive Type Theory*, volume 36 of *Oxford Logic Guides*, pages 127–172, Oxford, 1998. Clarendon Press.

[34] David McCarty. Undecidability and intuitionistic completeness. *J. Philos Logic*, 25:559–565, 1996.

[35] David McCarty. Completeness and incompleteness for intuitionistic logic. *Journal of Symbolic Logic*, 73(4):1315–1327, 2008.

[36] John C. Mitchell. *Foundations of Programming Languages.* MIT Press, 1996.

[37] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory.* Oxford Sciences Publication, Oxford, 1990.

[38] Benjamin C. Pierce. *Types and Programming Languages.* MIT Press, 2002.

[39] Benjamin C. Pierce et al. *Software Foundations.* Open Source, 2011.

[40] Gordon D. Plotkin. LCF considered as a programming language. *Journal of Theoretical Computer Science*, 5:223–255, 1977.

[41] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN-19, Aarhus University, Aarhus University, Computer Science Department, Denmark, 1981.

[42] H. Raisowa and R. Sikorski. *The Mathematics of Metamathematics.* Panstowe Wydawnictwo Naukowe, Warsaw, 1963.

[43] Bertrand Russell. *The Principles of Mathematics.* Cambridge University Press, Cambridge, 1908.

[44] R. M. Smullyan. *First–Order Logic.* Springer-Verlag, also Dover, New York, 1995, New York, 1968.

[45] A. S.Troelstra. Realizability. In S.R. Buss, editor, *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*, pages 407–473. Elsevier, 1998.

[46] A. Tarski. Der aussangenkalkul und die topologie. *Fundamenta Mathematicae*, 31:103–134, 1938.

[47] A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics, An Introduction*, volume I, II. North-Holland, Amsterdam, 1988.

[48] Walter P. van Stigt. *Brouwer's Intuitionism.* North-Holland, Amsterdam, 1990.

[49] W. Veldman. An intuitionistic completeness theorem for intuitionistic predicate calculus. *Journal of Symbolic Logic*, 41:159–166, 1976.