Intuitionistic Ancestral Logic as a Dependently Typed Abstract Programming Language

Liron Cohen[1] and Robert L. Constable[2]

[1] Tel-Aviv University, Tel-Aviv, Israel
`liron.cohen@math.tau.ac.il`
[2] Cornell University, Ithaca, NY, USA
`rc@cs.cornell.edu`

Abstract. It is well-known that concepts and methods of logic (more specifically constructive logic) occupy a central place in computer science. While it is quite common to identify 'logic' with 'first-order logic' (FOL), a careful examination of the various applications of logic in computer science reveals that FOL is insufficient for most of them, and that its most crucial shortcoming is its inability to provide inductive definitions in general, and the notion of the transitive closure in particular. The minimal logic that can serve for this goal is ancestral logic (AL).
In this paper we define a constructive version of AL, pure intuitionistic ancestral logic (iAL), extending pure intuitionistic first-order logic ($iFOL$). This logic is a dependently typed abstract programming language with computational functionality beyond $iFOL$, given by its realizer for the transitive closure operator TC, which corresponds to recursive programs. We derive this operator from the natural type theoretic definition of TC using intersection type. We show that provable formulas in iAL are uniformly realizable, thus iAL is sound with respect to constructive type theory. We further outline how iAL can serve as a natural framework for reasoning about programs.

1 Introduction

In the famous paper with the telling name "On the Unusual Effectiveness of Logic in Computer Science" [15], it is forcefully noted that "at present concepts and methods of logic occupy a central place in computer science, insomuch that logic has been called 'the calculus of computer science' [19]". To demonstrate this claim, this paper then studies an impressive (yet explicitly non-exhaustive) list of applications of logics in different areas of computer science: descriptive complexity, database query languages, applications of constructive type theories, reasoning about knowledge, program verification and model checking.

But *what* logic has such effectiveness? Pure first-order logic (FOL) is one of the most widely studied and taught systems of logic[3]. It is the base logic in which

[3] We use the term *pure* to indicate that equality, constants, and functions are not built-in primitives.

two of the most studied mathematical theories, Peano Arithmetic (PA) and Zermelo/Fraenkel set theory with choice (ZFC), are presented. However, a simple check of the above list of applications from [15] reveals that FOL is sufficient for none of them. All these examples indicate that the crucial shortcoming of FOL is its inability to provide inductive definitions in general, and the notion of the transitive closure of a given binary relation in particular. The minimal logic that can serve for this goal is ancestral logic (AL) which is a well known extension of FOL, obtained by adding to it a transitive closure operator (see, e.g., [4,9,17])[4]. Its expressive power exceeds that of FOL, since in AL one can give a categorical characterization of concepts such as the natural numbers and the concept of finiteness, which are not expressible in FOL (hence it is not compact). In [4] it was argued that AL provides a suitable framework for the formalization of mathematics as it is appropriate for defining fundamental abstract formulations of transitive relations that occur commonly in basic mathematics. AL is also fundamental in computer science as reasoning effectively about programs clearly requires having some version of a transitive closure operator in order to describe such notions as the set of nodes reachable from a program's variables.

The intuitionistic versions of the well-known systems mentioned above, intuitionistic first-order logic ($iFOL$), Heyting arithmetic (HA), intuitionistic ZF (IZF) [13] and the related CZF [2], are also well studied. These intuitionistic logics are important in constructive mathematics, linguistics, philosophy and especially in computer science. Computer scientists exploit the fact that intuitionistic theories can serve as programming languages [6,21] and that $iFOL$ can be read as an abstract programming language with dependent types. Since we are interested in extensions of intuitionistic first-order logic that clearly reveal the duality between logic and programming, and can capture general logical principles that have applicable computational content, it seemed natural to develop an intuitionistic version of AL – iAL, as a refinement of AL and an extension of $iFOL$. We believe that rather than $iFOL$, iAL should be taken as the basic logic which underlies most applications of logic to Computer Science. Many proofs in iAL turn out to have interesting computational content that exceeds that of $iFOL$ in ways of interest to computer scientists. We prove that iAL is sound with respect to constructive type theory by showing that provable formulas are *uniformly realizable.*We further outline how iAL can serve as a natural programming logic for specififying, developing, and reasoning about programs.

We adopt the presentation of $iFOL$ from *Intuitionistic Completeness of First-Order Logic* [10] where the computational content is made explicit using evidence semantics based on the propositions as types principle [20] aka the Curry Howard isomorphism [24]. A formal semantics of the logic we present could be based on extensional constructive type theories such as Intuitionistic Type Theory (ITT) [20] or Constructive Type Theory (CTT) [12,11,8]. However, the precise details of the semantical metatheory are not that critical to our results, so we remain informal. For other notions of truth and validity, one can refer to the accounts given in [25].

[4] Ancestral Logic is also sometimes called Transitive Closure Logic in the literature.

2 The System *iFOL*

This section reviews pure *iFOL* along the lines of [10]. The semantics of evidence for *iFOL* is simply a compact type theoretic restatement of the propositions-as-types realizability semantics given in [20,21,12]. This semantics plays an important role in building correct-by-construction software and in the semantics of strong constructive typed systems, such as Computational Type Theory (CTT) [11], Intuitionistic Type Theory (ITT) [20], Intensional-ITT [8,23], the Calculus of Inductive Constructions (CIC) [7], and Logical Frameworks such as Edinburgh LF [16]. The basic idea behind the semantics is that constructive proofs provide evidence terms (also called realizers) for the propositions they prove, and these realizers allow to directly extract programs from the proofs.

Let $\mathcal{L}$ be a first-order signature of predicates $P_i^{n_i}$ (with arity n_i) over a domain D of individuals of a model $\mathcal{M}$ for $\mathcal{L}$. The domain of discourse, D, can be any constructive type, $[D]_{\mathcal{M}}$.[5] Every formula A over $\mathcal{L}$ is assigned a type of objects denoted $[A]_{\mathcal{M}}$, called the *evidence* for A with respect to $\mathcal{M}$. We normally leave off the subscript $\mathcal{M}$ when there is only one model involved. Below is how evidence is defined for the various kinds of first-order propositional functions. The definition will also implicitly provide a syntax of first-order formulas.

Definition 1. (First-order formulas and their evidence)

- ***atomic propositional functions*** $P_i^{n_i}$ *are interpreted as functions from* D^{n_i} *into* $\mathbb{P}$ *the type of propositions. For the atomic proposition* $P_i^{n_i}(a_1, ..., a_{n_i})$, *the basic evidence must be supplied, say by objects* p_i. *In the uniform treatment, we consider all of these objects to be equal, and we denote them by the unstructured atomic element* $\star$. *Thus if an atomic proposition is known by atomic evidence, the evidence is the single element* $\star$ *of the unit type,* $\{\star\}$.[6]
- ***conjunction*** $[A \wedge B] \;=\; [A] \times [B]$, *the Cartesian product.*
- ***existential*** $[\exists x.B(x)] \;=\; x : [D]_{\mathcal{M}} \times [B(x)]$, *the dependent product.*
- ***implication*** $[A \Rightarrow B] \;=\; [A] \rightarrow [B]$, *the function space.*[7]
- ***universal*** $[\forall x.B(x)] \;=\; x : [D]_{\mathcal{M}} \rightarrow [B(x)]$, *the dependent function space.*
- ***disjunction*** $[A \vee B] \;=\; [A] + [B]$, *disjoint union.*
- ***false*** $[False] \;=\; \emptyset$ *the void type.*

Negation is defined by $\neg A := A \Rightarrow False$.

[5] As a first approximation readers can think of types as *constructive sets* [5]. Peter Aczel [1] shows how to interpret constructive sets as types in *ITT* [21]. Intuitionists might refer to *species* instead.

[6] It might seem that we should introduce atomic evidence terms that might depend on parameters, say $p(x, y)$ as the *atomic evidence* in the atomic proposition $P(x, y)$ but this is unnecessary and uniformity would eliminate any significance to those terms. In *CTT* and *ITT*, the evidence for atomic propositions such as equality and ordering is simply an unstructured term such as $\star$.

[7] This function space is interpreted type theoretically and is assumed to consist of *effectively computable deterministic functions.*

Fig. 1. The proof system *iFOL*

And Construction
$H \vdash A \wedge B$ *by* $pair(slot_a; slot_b)$
$H \vdash A$ *by* $slot_a$
$H \vdash B$ *by* $slot_b$

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

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

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

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* $slot_r$

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* $slot_b(d)$

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

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

Implication Decomposition
$H, f : A \Rightarrow B, H' \vdash G$ *by* $apseq\,(f; slot_g; v.sl_g\,[{}^{ap(f;slot_a)}/_{v}])$ *new* v[8]
$H \vdash A$ *by* $slot_a$
$H, v : B, H' \vdash G$ *by* $slot_g(v)$

Or Decomposition
$H, y : A \vee B, H' \vdash G$ *by* $decide(y; l.slot_{left}(l); r.slot_{right}(r))$
$H, l : A, H' \vdash G$ *by* $slot_{left}(l)$
$H, r : B, H' \vdash G$ *by* $slot_{right}(r)$

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* $slot_g(d, r)$

All Decomposition
$H, f : \forall x.B(x), H' \vdash G$ *by* $apseq(f; d; v.slot_g\,[{}^{ap(f;d)}/_{v}])$
$H \vdash d \in D$ *by* $obj(d)$
$H, v : B(d), H' \vdash G$ *by* $slot_g(v)$[9]

False Decomposition
$H, f : False, H' \vdash G$ *by* $any(f)$

It is easy to prove classically that a formula A is satisfied in a model $\mathcal{M}$ iff there is evidence in $[A]_{\mathcal{M}}$ [10]. This shows that this evidence semantics can be read classically, and it will correspond to Tarski's semantics for FOL.

Definition 2. *The proof system iFOL is given in Fig. 1.*

The rules of the system $iFOL$ are presented in the "top down style" (also called refinement style) in which the goal comes first and the rule name with parameters generates subgoals. This style is compatible with the highly successful tactic mechanism of the Edinburgh LCF proof assistant [14] and the style for rules and proofs used in the Nuprl book [12]. Thus the sequent style trees are grown with the root at the top. This is also compatible with the standard writing style in which a theorem is stated first followed by its proof. For a more detailed explanation of the syntax used in the proof rules see [10].

3 The System iAL

3.1 The Transitive Closure Operator

A standard mathematical definition of the transitive closure of a binary relation R, denoted by R^+, is as follows. Let $\mathbb{N}$ be the set of natural numbers. For $n \in \mathbb{N}$ define: $R^{(0)} = R$, $R^{(n+1)} = R^{(n)} \circ R$, where the composition of relations R and S is defined by $(S \circ R)(x, y)$ iff $\exists z\,(S(x, z) \wedge R(z, y))$.

Definition 3. *The* transitive closure R^+ *of binary relation* R *is defined by*

$$R^+(x, y) = \exists n : \mathbb{N}.R^{(n)}(x, y)$$

At appropriate places we use the notation xRy instead of $R(x, y)$.

Note that we are using an intuitionistic semantics in our metatheory, so, for instance, the definition of composition means that we can effectively find the value z. Moreover, the constructive nature of the definition entails that xR^+y implies we know a natural number witness for the number of iterations of the relation R. Hence we can prove in the semantics that given elements x and y in D, xR^+y iff we can *effectively find* a finite list of elements $x_1, ...x_n$ from D, such that $xRx_1 \wedge x_1Rx_2 \wedge ... \wedge x_nRy$.

While this definition is perfectly acceptable, it depends essentially on the type of natural numbers with its attendant notion of equality and induction. Thus, it requires invoking a version of intuitionistic ω-logic (e.g. [22]) as an underlying logic. In search of simplicity, our axiomatic definition will be given in terms of finite lists without mentioning the natural numbers explicitly. This will allow us to frame iAL in a more generic and polymorphic way.

Observe the following (equivalent) definition for the transitive closure.

[8] This notation shows that $ap(f; sl_a)$ is substituted for v in $g(v)$. In the CTT logic we stipulate in the rule that $v = ap(f; sl_a)\ in\ B$.

[9] In the CTT logic, we use equality to stipulate that $v = ap(f; d)\ in\ B(v)$ just before the hypothesis $v : B(d)$.

Proposition 1. *R^+ is the minimal transitive relation L such that $R \subseteq L$, i.e.*

$$R^+ = \bigcap_{R \subseteq L \& Transitive(L)} L$$

where a relation R is said to be transitive if $\forall x, y, z.(xRy \wedge yRz) \Rightarrow xRz$.

This definition uses the *intersection type* of Constructive Type Theory (CTT) used in h[3], the type $\bigcap_{x:A} B(x)$. Its elements are those that belong to all of the types $B(x)$. It generalizes the binary intersection $A \cap B$, consisting of the elements that belong to both types A and B. For instance $\{x : \mathbb{N}|Even(x)\} \cap \{x : \mathbb{N}|Prime(x)\}$ is the unit type $\{2\}$. We shall use this definition to form our axiomatic system. This is a key step toward a polymorphic account of *iAL* which will support our claim that a type theoretic semantics can be not only *elementary*, but even *uniform*.

3.2 Realizability Semantics for *iAL*

Instead of defining evidence for transitive closure using $\mathbb{N}$, we use more generic and polymorphic constructs to give evidence for the transitive closure, in the spirit of using polymorphic functions, pairs, and tags. To know $R^+(x, y)$ for x and y in D, we construct a *list* of elements of D, say $[d, ..., d']$, and a list of evidence terms $[r, ..., r']$ such that r is evidence for $R(x, d)$ and r' is evidence for $R(d', y)$ and the intermediate terms form an *evidence chain*. These relationships hold because of the way the evidence is built up, so we do not need the numerical indices to define the relationship. It is crucial to notice that the concept of lists is subsumed into the realizers and does not appear in the logic itself.

Notice that any well-formed formula (wff) together with a pair of distinct variables may be viewed as defining a binary relation. The notation $A_{x,y}$ will be used to specify that we treat the formula A as defining a binary relation with respect to x and y distinct variables (other free variables that may occur in A are taken as parameters). Thus, one may apply the transitive closure operator not only to atomic predicates, but to any wff. We write $A_{x,y}(u, v)$ for the formula obtained by substituting u for x and v for y in A. For simplicity of presentation, in what follows the subscript x, y is omitted where there is no chance of confusion.

Definition 4. (*iAL* formulas and their evidence)
iAL formulas are defined as iFOL formulas with the following addition:

- *If A is a formula, x, y distinct variables, and u, v variables, then $A^+_{x,y}(u, v)$ is a formula.*
- *The evidence type for $A^+_{x,y}(u, v)$ consists of lists of the form $[\langle u, d_1, r_1\rangle, \langle d_1, d_2, r_2\rangle, ..., \langle d_n, v, r_{n+1}\rangle]$ where $u, d_1, ..., d_n, v : [D]_{\mathcal{M}}$, $r_1 \in [A_{x,y}(u, d_1)]$, $r_{n+1} \in [A_{x,y}(d_n, v)]$, and $r_i \in [A_{x,y}(d_{i-1}, d_i)]$ for $1 < i \leq n$*

Notice that the realizers for transitive closure formulas are all polymorphic and thus independent of realizers for particular atomic formulas.

Recall that according to Def. 3, $R^{+}(x,y)$ iff $\exists n \left(N(n) \wedge R^{(n)}(x,y)\right)$. This is not a legal formula in our language, but this is intuitively what we mean, if we had the natural numbers at our disposal. The realizer for this "formula" is of the form: $\langle n, \langle nat(n), \langle x, d_1, ..., d_n, y, \langle r_1, ..., r_{n+1}\rangle\rangle\rangle\rangle$ where $nat(n)$ realizes $N(n)$. The realizer of the transitive closure correlates nicely to this realizer. A realizer for a formula $R^{+}(x,y)$ of the form $\langle n, \langle nat(n), \langle x, d_1, ..., d_n, y, \langle r_1, ..., r_{n+1}\rangle\rangle\rangle\rangle$ can be easily converted into the form $[\langle x, d_1, r_1\rangle, \langle d_1, d_2, r_2\rangle, ..., \langle d_n, y, r_{n+1}\rangle]$ simply by rearranging the data. For the converse, the data can also be rearranged, but some additional data is required: n – which is the length of the list minus 1; and the realizer for it being a natural number – which is available as the length of a list is always a natural number.

3.3 Proof System for *iAL* over Domain *D*

We present a proof system for *iAL* which extends *iFOL* [10] by adding construction and decomposition rules for the transitive closure operator. We here use the standard canonical operator $[\,]$ for list constructor, and the non-canonical operator associated with it, *concat*, for concatenating two lists.

Definition 5. *The proof system iAL is defined by adding to iFOL the following rules for the transitive closure operator.*

– ***TC Base***

$H, x:D, y:D, H' \vdash A^{+}(x,y)\ by\ \ [\langle x, y, slot\rangle]$
$H, x:D, y:D, H' \vdash A(x,y)\ by\ slot$

– ***TC Trans***

$H, x:D, y:D, H' \vdash A^{+}(x,y)\ \ by\ \ concat\,(slot_l, slot_r)$
$H, x:D, z:D, H' \vdash A^{+}(x,z)\ \ by\ \ slot_l$
$H, y:D, z:D, H' \vdash A^{+}(z,y)\ \ by\ \ slot_r$

– ***TC Ind***

$H, x:D, y:D, r^{+}:A^{+}(x,y), H' \vdash B(x,y)\ by\ tcind\left(r^{+}; u, v, w, b_1, b_2.tr(u,v,w,b_1,b_2);\right.$
$\left. u, v, r.st(u,v,r)\right)$
$H, u:D, v:D, w:D, b_1:B(u,v), b_2:B(v,w), H' \vdash B(u,w)\ by\ tr(u,v,w,b_1,b_2)$
$H, u:D, v:D, r:A(u,v), H' \vdash B(u,v)\ by\ st(u,v,r)$
where u, v, w are fresh variables.

Rule TC Base states that the list consisting of the triple $[\langle x, y, r\rangle]$ where r realizes $A(x,y)$ is the realizer for the transitive closure $A^{+}(x,y)$. The crucial point about Rule TC Trans is that it does not nest lists of triples for the same goal; instead we "flatten the lists out" as proofs are constructed. This means that proofs of transitive closure have a distinguished realizer. Furthermore, it provides an adequate mechanism for creating a flat chain of evidence needed for the transitive closure induction rule.

The realizer for Rule TC Ind computes on the list r^+ and is recursively defined as follows:
$tcind(r^+; u, v, w, b_1, b_2.tr(u, v, w, b_1, b_2); u, v, r.st(u, v, r))$ computes to:
If $base(r^+)$ then $st(r^+.1_1, r^+.1_2, r^+.1_3)$; else
$tr(r^+.1_1, r^+.1_2, r^+.2_2, tcind\left(rest(r^+); u, v, w, b_1, b_2.tr(u, v, w, b_1, b_2); u, v, r.st(u, v, r)\right)$.

The operator $base(r^+)$ is true when r^+ is simply the singleton triple. We use the notation $r^+.u$ to denote the uth element in the list r^+, and the subscript $r^+.u_i$ selects the ith elements of the triple ($i \in \{1, 2, 3\}$). The operator $rest(r^+)$ returns the list r^+ without its first element.

The more commonly used induction rule (see [4,17]) is derivable in iAL.

Proposition 2. *The following rule is derivable in iAL:*
$H, x : D, y : D, r^+ : A^+(x, y), g : G(x), H' \vdash G(y)$
$H, u : D, v : D, r : A(u, v), g' : G(u), H' \vdash G(v)$
where u, v are fresh variables.

We next demonstrate that iAL is an adequate system for handling the transitive closure operator by showing that fundamental, intuitionistically valid statements concerning the TC operator are provable in iAL. Given a signature with a binary relation R, intuitively we may think of its interpretation as a directed graph whose vertices are the elements of the domain and two vertices are adjoined by an edge iff their interpretations are in the interpretation of the relation R. Then, R^+ is interpreted by the existence of a path between two vertices. Observe the following basic statement: "if there is a path between x and y in a graph G, then either x and y are neighbors, *or* there is a neighbor z of x, such that from z there is a path to y". This statement is classically valid, and though at first sight one may doubt that it is intuitionistically valid (as it contains a disjunction), it is provable in iAL.

Proposition 3. *The following are provable in iAL:*

$$A^+(x, y) \vdash A(x, y) \vee \exists z \left(A(x, z) \wedge A^+(z, y)\right) \tag{1}$$
$$A^+(x, y) \vdash A(x, y) \vee \exists z \left(A^+(x, z) \wedge A(z, y)\right) \tag{2}$$

Another basic statement in graph theory is: "if there is a path between x and y in a graph G, then x and y are not isolated". Again, while it may seem to be intuitionistically invalid because of the existential nature of the argument, it turns out to be provable in iAL.

Proposition 4. *The following are provable in iAL:*

$$A^+(x, y) \vdash \exists z A(x, z) \tag{3}$$
$$A^+(x, y) \vdash \exists z A(z, y) \tag{4}$$

The above proposition is based on the more general fact that the existential quantifier is definable by the transitive closure operator (see [4]).

Proposition 5. *The following is provable in iAL:*

$$\vdash \exists x A \leftrightarrow \left(A\left\{\frac{u}{x}\right\} \vee A\left\{\frac{v}{x}\right\}\right)^{+}_{u,v}(u,v)$$

where u and v are fresh variables.[10]

Notice that there is a strong connection between our choice for the realizer of the transitive closure and the standard realizers for $iFOL$. For example, Prop. 5 entails that the existential quantifier is definable using the transitive closure operator. The standard realizer for $\exists x P(x)$ is a pair $\langle d, \star\rangle$, since P is an atomic relation. The realizer for the defining formula, $(P(u) \vee P(v))^{+}(u,v)$, is of the form $[\langle u, d_1, r_1\rangle, \langle d_1, d_2, r_2\rangle, ..., \langle d_n, v, r_{n+1}\rangle]$ where each r_i is a realizer for $P(d_i) \vee P(d_{i+1})$, and $d_0 := u$ and $d_{n+1} := v$. Now, suppose we have a realizer of the form $\langle d, \star\rangle$ of $\exists x P(x)$. The realizer for the defining formula in iAL will be $[\langle u, d, inr(\star)\rangle]$. For the converse, suppose we have a realizer of the form $[\langle u, d_1, r_1\rangle, \langle d_1, d_2, r_2\rangle, ..., \langle d_n, v, r_{n+1}\rangle]$. Then we can create a realizer for $\exists x P(x)$ in the following way: if r_1 is $inl(\star)$ return $\langle u, \star\rangle$, else return $\langle d_1, \star\rangle$.

3.4 Soundness for *iAL*

We next prove that iAL is sound by showing that every provable formula is realizable, and even uniformly realizable. We do this by giving a semantics to sequents and then proceed by induction on the structure of the proofs. It is important to note that the realizers are all polymorphic, they do not contain any propositions or types as subcomponents and thus serve to provide evidence for any formulas built from any atomic propositions.

Given a type D (empty or not) as the domain of discourse, and given atomic propositional functions from D to propositions, $\mathbb{P}$, for the atomic propositions, and given the type theoretic meaning of the logical operators and the transitive closure operator, we can interpret an iAL sequent over dependent types by saying that a sequent $x_1 : T_1, x_2 : T_2(x_1), ..., x_n : T_n(x_1, ..., x_{n-1}) \vdash G(x_1, ..., x_n)$ defines an effectively computable function from an n-tuple of elements of the dependent product of the types in the hypothesis list to the type of the goal, $G(x_1, ..., x_n)$.

Theorem 1. ***(Realizability Theorem for iAL)***
Every provable formula of iAL is realizable in every model.

Proof. The proof is carried out by induction on the structure of proofs in iAL. The proof rules for iAL show how to construct a realizer for the goal sequent given realizers for the subgoals. Also, the atomic (axiomatic) subgoals are of the form $x_1 : T_1, x_2 : T_2(x_1), ..., x_n : T_n(x_1, ..., x_{n-1}) \vdash T_j(x_1, ..., x_n)$, which are clearly realizable.

Since propositions-as-types realizability is usually regarded as the definition of constructive truth, this theorem allows us to also say that every provable formula is true in every constructive model (i.e. intuitionistically valid).

[10] The notation $A\left\{\frac{u}{x}\right\}$ denotes substituting u for x in A.

Theorem 2. ***(Soundness Theorem for iAL)***
Every provable formula of iAL is intuitionistically valid.

Corollary 1. ***(Consistency Theorem for iAL)***
iAL is consistent, i.e. $False$ is unprovable in iAL.

3.5 Reasoning about Programs in *iAL*

In this section we provide some examples of how iAL can be used for reasoning about programs, and the benefits of using it for this task. We leave for an extended version of this paper a more detailed insight on the connections between iAL and programming.

$iFOL$ can be viewed and used as an abstract programming language, particularly suitable for correct-by-construction style of programming [10]. This is due to the following key feature of $iFOL$: proofs of specifications in its proof system carry their computational content in the realizers. Thus, proving an $iFOL$ formula results in a realizer which can be thought of as holding the computational element of a program, and so one can extract programs from proofs that $iFOL$ specifications are solvable. iAL enjoys these features too, but has greater expressive and proof-theoretic power. Accordingly, iAL can serve as a much better framework for specifying, developing, and reasoning about programs. There are many meaningful statements about programs that cannot be formulated in $iFOL$ but can be captured in iAL, such as "there is a state to which each run gets to (on any input)", which can be formulated in iAL by the formula $\exists y \forall x P^{+}(x, y)$. Moreover, even simple provable assertions, such as $A^{+}(x, y) \Rightarrow \exists z A(z, y)$, have interesting realizers that depend on the tcind realizer, and thus correspond to recursive programs.

When reasoning about programming, it is important to notice that the **if** and **while** program constructs can be encoded in iAL (similar to the way it is done in propositional Dynamic Logic). For instance, "while b do p" can be encoded by the formula $(B\left(x\right) \wedge P\left(x, y\right))^{+} \wedge \neg B\left(y\right)$. Thus, there is also a strong connection to programming that derives from the relation between iAL and the theory of flowchart schemes (e.g., [18]). A flowchart scheme is a vertex-labeled graph that represents an uninterpreted program, and a central question in the theory of flowchart schemes is scheme equivalence. In [18] Manna presents examples of equivalence proofs done by transformations on the graphs of the schemes. Since each flowchart scheme can be assigned a iAL formula (as it is simply a vertex-labeled graph), the question of scheme equivalence can be replaced by the question of equivalence between two iAL formulas in the effective, constructive proof system iAL.

4 Further Research

We argue that iAL is a natural "next step" one needs to take, starting from $iFOL$, in order to capture many applications in computer science. Further work is still needed to investigate the natural scope of iAL and to demonstrate its usefulness by exploring several applications of it in diversity of areas of computer science and mathematics, such as database query languages, specification languages, and programs development and verification. For instance, one such example might be related to distributed protocols. One can develop efficient deployed algorithms from the natural constructive proofs of theorems about data structures expressible in iAL, and explore specific direct use of such proofs, e.g. in building distributed protocols and making them attack-tolerant.

Acknowledgment

This research was partially supported by the Ministry of Science, Technology and Space, Israel, and the Cornell University PRL Group.

References

1. Peter Aczel. The type theoretic interpretation of constructive set theory. In Leszek Pacholski Angus Macintyre and Jeff Paris, editors, *Logic Colloquium '77*, volume 96 of *Studies in Logic and the Foundations of Mathematics*, pages 55 – 66. Elsevier, 1978.
2. Peter Aczel. The type theoretic interpretation of constructive set theory: Inductive definition. In *Logic, Methodology and Philosophy of Science VII*, pages 17–49. Elsevier Science Publishers, 1986.
3. Stuart F. Allen, Mark Bickford, Robert L. Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and E. Moran. Innovations in computational type theory using nuprl. *J. Applied Logic*, 4(4):428–469, 2006.
4. Arnon Avron. Transitive closure and the mechanization of mathematics. In *Thirty Five Years of Automating Mathematics*, pages 149–171. Springer, 2003.
5. Bruno Barras. Sets in coq, coq in sets. *Journal of Fromalized Reasoning*, 3(1):29–48, 2010.
6. Joseph L. Bates and Robert L. Constable. Proofs as programs. *ACM Trans. Program. Lang. Syst.*, 7(1):113–136, January 1985.
7. Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions.* springer, 2004.
8. Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of agda–a functional language with dependent types. In *Theorem Proving in Higher Order Logics*, pages 73–78. Springer, 2009.
9. Liron Cohen and Arnon Avron. Ancestral logic: A proof theoretical study. In U. Kohlenbach et al., editor, *Logic, Language, Information, and Computation*, volume 8652 of *Lecture Notes in Computer Science*, pages 137–151. Springer Berlin Heidelberg, 2014.
10. Robert Constable and Mark Bickford. Intuitionistic Completeness of First-Order Logic. *Annals of Pure and Applied Logic*, 165(1):164–198, January 2014.
11. Robert L. Constable, Stuart F. Allen, Mark Bickford, Richard Eaton, Christoph Kreitz, Lori Lorigo, and E. Moran. Innovations in computational type theory using nuprl. *J. Applied Logic*, 4(4):428–469, 2006.

12. Robert L. Constable, Stuart F. Allen, Mark Bromley, Rance Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, Todd B. Knoblock, N. P. Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing mathematics with the Nuprl proof development system.* Prentice Hall, 1986.
13. Harvey Friedman. The consistency of classical set theory relative to a set theory with intuitionistic logic. *The Journal of Symbolic Logic*, 38(2):pp. 315–319, 1973.
14. Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: a mechanized logic of computation*, volume 78. Springer-Verlag, NY, 1979.
15. Joseph Y. Halpern, Robert W. Harper, Neil Immerman, Phokion G. Kolaitis, Moshe Y. Vardi, and Victor Vianu. On the unusual effectiveness of logic in computer science. *Bulletin of Symbolic Logic*, 7(02):213–236, 2001.
16. Robert W. Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM (JACM)*, 40(1):143–184, 1993.
17. Immerman N. Reps T. Sagiv M. Srivastava S. Lev-Ami, T. and G. Yorsh. Simulating reachability using first-order logic with applications to verification of linked data structures. In *Automated Deduction*, volume 3632 of *Lecture Notes in Computer Science*, pages 99–115. Springer Berlin Heidelberg, 2005.
18. Zohar Manna. *Mathematical Theory of Computation.* McGraw-Hill, Inc., New York, NY, USA, 1974.
19. Zohar Manna and Richard Waldinger. *The logical basis for computer programming*, volume 1. Addison-Wesley Reading, 1985.
20. P. Martin-Löf and G. Sambin. *Intuitionistic type theory.* Studies in proof theory. Bibliopolis, 1984.
21. Per Martin-Löf. Constructive mathematics and computer programming. *Studies in Logic and the Foundations of Mathematics*, 104:153–175, 1982.
22. J.D. Monk. *Mathematical Logic.* Number 1-243 in Graduate Texts in Mathematics. Springer, 1976.
23. B. Nordström, K. Petersson, and J.M. Smith. *Programming in Martin-Löf's type theory: an introduction.* International series of monographs on computer science. Clarendon Press, 1990.
24. M.H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomoprhism.* Elsevier, 2006.
25. A.S. Troelstra and D. Dalen. *Constructivism in Mathematics: An Introduction.* Number 1 in Constructivism in Mathematics. North-Holland, 1988.

Appendix

In this appendix we provide full proofs of some of the results above.

Proof of Proposition 2:

The result immediately follows from TC Ind by taking $A(u,v)$ to be the formula $G(u) \Rightarrow G(v)$. □

Lemma 1. *The following are provable in iAL:*

$$A(x,z), A^{+}(z,y) \vdash A^{+}(x,y) \tag{5}$$

$$A^{+}(x,z), A(z,y) \vdash A^{+}(x,y) \tag{6}$$

Proof of Lemma 1:

Both sequents are derivable by one application of TC Base followed by an application of TC Trans. □

Proof of Proposition 3:

Denote by $\varphi(x,y)$ the formula $\exists z (A(x,z) \wedge A^{+}(z,y))$. For (1) apply TC Ind on the following two subgoals:

1. $A(u,v) \vdash A(u,v) \vee \varphi(u,v)$
2. $A(u,v) \vee \varphi(u,v), A(v,w) \vee \varphi(v,w) \vdash A(u,w) \vee \varphi(u,w)$

(1) is clearly provable in $iFOL$. For (2) it suffices to prove the following four subgoals, from which (2) is derivable using Or Decomposition and Or Composition:

1. $A(u,v), A(v,w) \vdash \varphi(u,w)$
2. $A(u,v), \varphi(v,w) \vdash \varphi(u,w)$
3. $\varphi(u,v), A(v,w) \vdash \varphi(u,w)$
4. $\varphi(u,v), \varphi(v,w) \vdash \varphi(u,w)$

We prove $\varphi(u,v), \varphi(v,w) \vdash \varphi(u,w)$, the other proofs are similar. It is easy to prove (using Lemma 1) that $\exists z (A(v,z) \wedge A^{+}(z,w)) \vdash A^{+}(v,w)$. By TC Trans we can deduce $d : D, A(u,d), A^{+}(d,v), A^{+}(v,w) \vdash A^{+}(d,w)$, from which $\exists z (A(u,z) \wedge A^{+}(z,w))$ is easily derivable.

The proof of (2) is similar. □

Proof of Proposition 4:

(3) is derivable applying TC Ind on the following two subgoals:

1. $A(x,y) \vdash \exists z A(x,z)$, which is easily provable in $iFOL$.
2. $\exists z A(u,z), \exists z A(v,z) \vdash \exists z A(u,z)$, which is valid due to Hypothesis.

The proof of (4) is symmetric. □

Proof of Proposition 5:

Denote by $\varphi(u,v)$ the formula $A(u,\overrightarrow{y}) \vee A(v,\overrightarrow{y})$. The right-to-left implication follows from Prop. 4 since $\exists z(A(u,\overrightarrow{y}) \vee A(z,\overrightarrow{y})) \vdash \exists x A(x,\overrightarrow{y})$ can be easily proven in $iFOL$, and Prop. 4 entails that $\varphi^{+}(u,v) \vdash \exists z \varphi(u,z)$. For the left-to-right implication it suffices to prove $d : D, A(d,\overrightarrow{y}) \vdash \varphi^{+}(u,v)$. Clearly, in $iFOL$, $d : D, A(d,\overrightarrow{y}) \vdash \varphi(d,v)$ is provable, from which we can deduce by TC Base $d : D, A(d,\overrightarrow{y}) \vdash \varphi^{+}(d,v)$. Since we also have $d : D, A(d,\overrightarrow{y}) \vdash \varphi(u,d)$, by Lemma 1 we obtain $d : D, A(d,\overrightarrow{y}) \vdash \varphi^{+}(u,v)$. □

Proposition 6. *The following is provable in iAL:*

$$\left(A^{+}\right)^{+}(x,y) \vdash A^{+}(x,y)$$

Proof of Proposition 6:

Applying TC Ind on $A^{+}(u,v), A^{+}(v,w) \vdash A^{+}(u,w)$ (which is is derivable using TC Trans) and $A^{+}(x,y) \vdash A^{+}(x,y)$ (which is clearly provable) results in the desired proof. □