Assigning Meaning to Proofs: a semantic basis for problem solving environments

Robert L. Constable*

Cornell University

Ithaca, NY 14853

Abstract

According to Tarski's semantics, the denotation of a sentence in the classical predicate calculus with respect to a model is its truth value in that model. In this paper we associate with every sentence a set comprising evidence for it and show that a statement is true in a model exactly when there is evidence for it. According to this semantics, the denotation of a sentence is this set of evidence.

Proofs are regarded as expressions which denote evidence. Assigning this meaning to proofs gives them the same status as other algebraic expressions, such as polynomials. There are laws governing equality and simplification of proofs which can be used to explain the notion of constructive validity. A proof is called *constructive* when the evidence it denotes can be *computed* from it. A sentence is constructively valid when it has a constructive proof. These proofs turn out to be practical ways to present algorithms as has been demonstrated by an implementation of them in the Nuprl proof development system.

*This research was supported in part by NSF grants MCS-81-04018 and (joint Cornell-Edinburgh) MCS-83-03336.

NATO ASI Series, Vol. F 55
Constructive Methods in Computing Science
Edited by M. Broy

1 Introduction

If I correctly stated the winning lottery number before it has been publicly announced, many people will be more interested in my evidence for the assertation than in its truth. Even for routine utterances we are interested in the evidence. "How do you know?" we ask. In formal logic the "truth" of a sentence can be defined following Tarski [Tar44] who put the matter this way for the case of a universal statement: $\forall x.B(x)$ is true in a model $\mathbf{m}$ if the model assigns to B some propositional function $\mathbf{m}(B)$ which has value true for every element of the *universe of discourse* D of the model. That definition ignores evidence. We want to give a precise definition of evidence and relate it to truth as defined by Tarski.

In mathematics there is a persistent interest in evidence even though the official definition of truth does not refer to it. So if I claim that there is a regular 17-gon, then you may wish to see one. The ancient Greeks would require that I *construct* one or in some way actually exhibit it. As another interesting example, suppose that I claim that there are two irrational numbers, say x and y, such that x^y is rational. I might prove that they exist this way. Consider $\sqrt{2}^{\sqrt{2}}$, it is either rational or irrational. If it is rational, take $x = \sqrt{2}$, $y = \sqrt{2}$. If it is irrational, take $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$. Then $x^y = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^2 = 2$. So in either case there are the desired x, y. But notice that the evidence for existence here is *indirect*. I have not actually exhibited x by this method, even though you might be convinced that the statement is *true* in Tarski's sense. A constructive proof of this statement would actually exhibit x and y, say for example $x = \sqrt{2}$ and $y = 2 \cdot \log_2 3$ are irrational plus the computation $2^{(2 \cdot \log_2 3)/2} = 3$.

We begin to understand how the concept of evidence is defined if we examine its use in ordinary discourse. Existence statements like those above are especially significant. Evidence for **there is an x such that B holds**, symbolized $\exists x.B(x)$, consists of an element a and evidence for B holding

on a. Consider now evidence for a universal statement such as *for every natural number x there is a pair of prime numbers, $p, p+2$ greater than x.* Given 0, the evidence could be 2, 3 and given 2 it could be 5, 7, etc. But what is evidence for the universal statement? It should include in some way these instances, so it could be a function f which given a number n produces $f(n)$ as evidence for the assertion, e.g. $f(n)$ could be a pair $p, p+2$ and proof that p is greater than n.

Consider next evidence for a conjunction such as **n is odd and n is perfect.** We would expect to have evidence for both statements, say a pair containing (or comprising) a factorization of n as $2m+1$ and a summation of all of the factors in n.[1] Evidence for a disjuntion such as A or B will be either evidence for A or evidence for B. We also imagine that we could tell which case the evidence applied to.

Consider this implication: if there is a polynomial time algorithm to decide whether an arbitrary boolean expression is satisfiable, then there is an algorithm to determine whether a finite graph has a clique of any given size. This is a typical statement in modern computational complexity theory. The truth value of the antecedent is not known, yet the implication is true. It is true because we have a method to transform any (hypothetical) algorithm for satisfiability into one for clique. It is this method which bears witness for the implication. A more mundane example is this. When I say "if there is a 6 in the array A, then there is a 6 in the array B obtained from A by replacing each odd value and only odd values by 5;" you can recognize this as a true implication because you know a method of taking evidence for "6 is in A", say an index i and the value $A[i] = 6$, to evidence that "6 is in B," the same index i and $B[i] = 6$ will suffice for instance.

What is evidence for a negation such as "4 is not prime." First, notice that I mean "it is not the case that 4 is prime" rather than taking "not prime" as a separate predicate meaning 4 is composite. So in general we are looking at a uniform negation, $\neg A$. This is sometimes equivalent to

[1] A perfect number is one which is equal to the sum of its proper factors, e.g. $6 = 3 + 2 + 1$.

some positive statement, such as "4 is composite." But if we do not know anything about A other than that it is a proposition, then we cannot hope to reformulate it positively; so we ask whether we can say anything about negation uniformly in A. Intuitively it seems sensible to say that evidence for $\neg A$ is evidence that there is no evidence for A. So the question then is "how do we show that sets are empty?" We cannot do this by producing an element.

One way to show that a set, say described by term A, is empty is to show it as a subset of the canonical empty set, say ϕ, i.e. $A \subseteq \phi$. This amounts to showing an implication , if $x \epsilon A$ then $x \epsilon \phi$. One formula whose evidence set is clearly empty is *false*; that is the definition of the formula *false*, one with no proofs. So one way of proving $\neg F$ uniformly is to prove $F \Rightarrow false$. This is not the only way, but it is sufficient. We might for example prove that $F = G$ and $\neg G$ holds. But also, if we know $\neg F$, then we know that the evidence set for F is empty, so it is a subset of ϕ and so $(F \Rightarrow false)$. Thus we can take $F \Rightarrow false$ as the *definition* of a general negation; we adopt this definiton.

These explanations may not be definitive, but they provide a good starting point. In the next section we define a particular specimen of formal logic and give a precise definition of both truth and evidence. Then in section 3 we examine proofs and show how they encode evidence.

This interpretaion of evidence is not new, its origins go back at least to L.E.J. Brouwer who discovered in the early 1900's that to treat mathematics in a constructive or computational way, one must also treat logic in such a manner. He showed that this could be done by basing logic on the judgement that a sentence is *known to be true from evidence* rather than on the judgement that it is true. At this point in the discussion, we are not concerned with computable evidence exclusively, but with an abstract notion of evidence. In some sense we are extending Brouwer's ideas to classical logics as well. Later we will make connection to constructive logic via the so-called *propositions-as-types* principle due to Curry, Howard, and de Bruijn (for references see [CAB+86,deB80]). This principle can be seen as formalizing the notion of evidence in *type theory*, see also [ML84], [CAB+86].

2 The Logic

Syntax

We present a standard kind of predicate calculus. The formulas of our logic are built using the binary propositional connectives &, |, ¬, ⇒ (and, or, not, implies) and the quantifiers $\forall x$, $\exists x$ (*all and some*) where x ranges over the class of objects called *individual variables*. There must be at least one *propositional function constant*, there may be several but only a finite number of them (for simplicity), say $P_1, P_2, \ldots, P_k$. There may also be ordinary *function constants*, say $f_1, f_2, \ldots, f_m$. With each propositional or ordinary function constant we specify the number of arguments it takes; this is called its *arity*. The arity can be zero. The language will have *terms*, these include (individual) variables written $x_1, x_2, \ldots, \ldots$. If there are ordinary function constants, then the terms include expressions of the form $f_i(t_1, \ldots, t_n)$ where the t_j $j = 1, \ldots, n$ are terms and the f_i has arity n. If the arity of f_i is zero, then f_i is also an individual constant.

A *formula* of the logic is defined as follows:

(i) *false* is a formula (it is a *propositional constant*)

(ii) if P_i is a propositional function of arity n, and $t_1, \ldots, t_n$ are terms, then $P_i(t_1, \ldots, t_n)$ is a formula

(iii) if A *and* B are formulas, then so are

$(A\&B)$

$(A \mid B)$

$(A \Rightarrow B)$

(iv) if B is a formula, then so are

$(\exists x.B)$ and $(\forall x.B)$.

A particular instance of this language that is quite familiar arises by taking 0 and 1 as individual constants (arity zero ordinary functions), taking + and * as ordinary function constants, (writing $+(1,2)$ for $1+2$) and $=$ and $<$ as propositional functions of arity 2 so that $=(x,y)$ and $<(x\ +(x,1))$ stand for $x = y$ and $x < (x+1)$. In the remaining examples we will write these familiar functions in their usual infix manner taking them as abbreviations for the "official" form. Formulas in (i) and (ii) are atomic. Those in (iii) are *compound* with principle operator &, |, or $\Rightarrow$ (in that order). Those in (iv) are quantified formulas, and their principle operators, are $\forall x$ or $\exists x$.

The quantifiers $\forall x$, $\exists x$ are called *binding operators* because they bind occurrences of the variable x in the formulas to which they are applied. In $\forall x.B$ and in $\exists x.B$, the formula part B is called the *scope* of the quantifier. Any occurrence of x in B is bound. It is bound by the quantifier of smallest scope that causes it to be bound.

A variable occurrence in a formula F which is not bound is said to be *free*. If x is a free variable of F, and t is a term, we write $F(t/x)$ to denote the formula that results by substituting t for all free occurrences of x and renaming bound variables of F as necessary to prevent *capture of* free variables of t. Thus if t *contains a free variable* y and x occurs in the scope of a subformula whose quantifier binds y, say $\exists y.C$, then the quantifier is rewritten with a new variable, say $\exists y'.C$ because otherwise y would be captured by $\exists y.A$. For example, in the formula $\exists y.x < y$ the variable x is free. If we substitute the term $(y+1)$ for x we do not obtain $\exists y.(y+1) < y$ but instead $\exists y'.(y+1) < y'$. See [Kle52,ML82,ML84] for a thorough discussion of these points.

A formula with no free variables is called *closed*. A closed formula is called a sentence.

If for convenience we want to avoid writing all of the parentheses, then we adopt the convention that all the binary operators are right associative with the precedence &, |, $\Rightarrow$ and then quantifiers. Thus $\forall x.B(x)\&C \Rightarrow \exists y.P(x,y) \mid B(y)$ abbreviates $(\forall x.((B(x)\&C) \Rightarrow (\exists y.(P(x,y) \mid B(y)))))$.

Semantics of Truth

The meaning of a formula is given with respect to a model which consists of a set D, called the *domain* of discourse, and a function $\mathbf{m}$ called *interpretation* which maps each ordinary function constant f of arity n to a function from D^n into D denoted $\mathbf{m}(f)$ and maps each propositional function constant P of arity n to a function from D^n into $\{T, F\}$ denoted $\mathbf{m}(P)$.

To give the meaning of formulas with free variables x_i we need the idea of a *state* which is a mapping of variables to values, that is $s(x_i)$ belongs to D. When we want to alter a state at a variable x_i we write $s[x_i := a]$ which denotes $s(y)$ if $y \neq x_i$ and denotes a if $y = x_i$. We define the relation that formula F is true in model $\mathbf{m}$ and state s, written

$$\mathbf{m} \models_s F.$$

Preliminary to this concept we need to define the meaning of a term in state s, written $\mathbf{m}(t)(s)$. The meaning of constants is given by $\mathbf{m}$, so $\mathbf{m}(c_i)(s) = \mathbf{m}(c_i)$. The meaning of variables is given s, so $\mathbf{m}(x_i)(s) = s(x_i)$. The meaning of $f(t_1, \ldots, t_n)$ is $\mathbf{m}(f(t_1, \ldots, t_n))(s) = \mathbf{m}(f)(\mathbf{m}(t_1)(s), \ldots, \mathbf{m}(t_n)(s))$.

Truth Conditions

1. $\mathbf{m} \models_s P(t_1, \ldots, t_n)$

   $iff\ \mathbf{m}(P)(s)(n(t_1)(s), \ldots, n(t_n(s)) = T$

   for P a propositional function constant of arity n.

2. $\mathbf{m} \models_s (A \& B)$

   $iff\ \mathbf{m} \models_s A$ *and* $\mathbf{m} \models_s B$

3. $\mathbf{m} \models_s (A \mid B)$

   $iff\ \mathbf{m} \models_s A$ *or* $\mathbf{m} \models_s B$

4. $\mathbf{m} \models_s (A \Rightarrow B)$

   $iff\ \mathbf{m} \models_s A$ *implies* $\mathbf{m} \models_s B$

5. $\mathbf{m} \models_s \forall x.B$

   $iff\ \mathbf{m} \models_{s'} B$ *for all* $s' = s[x := a]$ with a in D

6. $\mathbf{m} \models_s \exists x.B$

   $iff\ \mathbf{m} \models_{s'} B$ *for some* $s' = s[x := a]$ for a in D

Semantics of Evidence

The following set constructors are needed in the semantics of evidence. Given sets A and B, let $A \times B$ denote their cartesian product, let $A + B$ denote their disjoint union, and let $A \rightharpoonup B$ denote all the functions from A to B. Given $B(x)$ a family of sets indexed by A, let

$$\prod_{x \epsilon A} B(x)$$

denote the set of functions f from A into,

$$\cup_{x \epsilon A} B(x)$$

such that $f(a)$ belongs to $B(a)$. We also take

$$\sum_{x \epsilon A} B(x)$$

to be the disjoint union of the family of sets. It can be defined as $\{< a, b > |\ a \epsilon A,\ b \epsilon B(a)\}$.

Now we define $\mathbf{m}[A](s)$, the evidence for formula A in model $\mathbf{m}$ and state s

1. $\mathbf{m}[false](s) = the\ empty\ set$
2. $\mathbf{m}[P(t_1, \ldots, t_n)](s) = \{T\}\ if\ \mathbf{m} \models_s P(x_1, \ldots, x_n)$
   empty otherwise for P a propositional function constant.
3. $\mathbf{m}[A \& B](s) = \mathbf{m}[A](s) \times \mathbf{m}[B](s)$
4. $\mathbf{m}[A \mid B](s) = \mathbf{m}[A](s) + \mathbf{m}[B](s)$
5. $\mathbf{m}[A \Rightarrow B](s) = \mathbf{m}[A](s) \rightarrow \mathbf{m}[B](s)$
6. $\mathbf{m}[\forall x.B](s) = \Pi_{y \epsilon D} \mathbf{m}[B](s[x := y\ |)$
7. $\mathbf{m}[\exists x.B] = \{< a, b > |\ a \epsilon D \& b \epsilon \mathbf{m}[B](s[x := a])\} = \sum_{y \epsilon D} \mathbf{m}[B](s[x := y])$

So we have defined inductively on the structure of a formula A a collection of objects that constitute the evidence for A in a particular model $\mathbf{m}$. In the base case, 1, the definition relies on

the semantics of truth. Here is an example of evidence: $< 6, T >$ belongs to $\mathbf{m}[\exists y.5 < y](s)$. We need to know that $5 < 6$ is true so that T belongs to $\mathbf{m}[5 < 6](s)$. Truth and evidence are related in this simple way.

Theorem 1 *For every sentence B, model $\mathbf{m}$ and state s*

$$\mathbf{m} \models_s B \textit{ iff there is } b \in \mathbf{m}[B](s).$$

proof

The proof is accomplished by induction on the structure of B showing both directions of the biconditional at each step. The easiest direction at each step is showing that if $b \epsilon \mathbf{m}[B](s)$, then $\mathbf{m} \models_s B$. We do these steps first, but the induction assumption at each step is the statement of the theorem for subformulas of B. To determine the subterms we proceed by case analysis on the outer operator of B. (We drop the state when it is not needed.)

1. If B is atomic, then the result is immediate.

   (1) B is $B_1 \& B_2$

   Then $b \in \mathbf{m}[B_1 \& B_2]$ so b is a pair, say, $< b_1, b_2 >$ and $b_1 \epsilon\ \mathbf{m}[B_1]$ and $b_2 \epsilon\ \mathbf{m}[B_2]$. By induction then, $\mathbf{m} \models B_1$ and $\mathbf{m} = B_2$ so $\mathbf{m} \models B_1 \& B_2$.

   (2) *B is $B_1 \mid B_2$*

   Given $b \in \mathbf{m}[B_1 \mid B_2] = \mathbf{m}[B_1] + [B_2]$, it must be in one disjunct or the other. That disjunct will be true by the induction hypothesis, so the whole disjunction is true.

   (3) B is $B_1 \Rightarrow B_2$

   Given $f \epsilon\ \mathbf{m}[B_1 \Rightarrow B_2]$, we consider two cases. Either $\mathbf{m}[B_1]$ is empty or there is some $b_1 \epsilon\ \mathbf{m}[B_1]$. In the later case $f(b_1) \epsilon\ \mathbf{m}[B_2]$ so B_2 is true and so is $B_1 \Rightarrow B_2$. If $\mathbf{m}[B_1]$ is empty, then by the hypothesis B_1 is false. So $B_1 \Rightarrow B_2$ is true.

(4) B is $\forall x.B_1$

Given

$$f\epsilon \prod_{v\epsilon D} \mathbf{m}(B_1)(s[x := v])$$

then for any $a\epsilon D$, $f(a)\epsilon\ \mathbf{m}(B_1)(s[x := a])$. So B_1 is true for all elements of D. Thus $\forall x.B$ is true.

(5) B is $\exists x.B_1$

Given c in $\{< a, b >|\ a\ \epsilon D \& b\ \epsilon\ \mathbf{m}[B_1](s[x := a])\}$ we have that B_1 is true on a. So B is true.

2. Now we must show that if B is true in model $\mathbf{m}$ and state s, then there is evidence for $\mathbf{m}[B](s)$. Again we proceed by induction on the structure of B. Clearly B cannot be false, and the result is immediate for other atomic B.

(1) B is $B_1 \& B_2$

Both B_1 and B_2 must be true if B is. So by the induction hypothesis there are $b_1\epsilon\ \mathbf{m}[B_1], b_2\epsilon\ \mathbf{m}[B_2]$. By definition $< b_1, b_2 > \epsilon\ \mathbf{m}[B_1 \& B_2]$.

(2) B is $B_1 \mid B_2$

Either B_1 or B_2 is true. In either case, by induction there is an element of $\mathbf{m}[B_1]$ or of $\mathbf{m}[B_2]$.

(3) B is $B_1 \Rightarrow B_2$

B_2 is either true or false. If it is true, then by the induction hypothesis there is $b_2\epsilon\ \mathbf{m}[B_2]$. So the constant function returning this value is evidence for $B_1 \Rightarrow B_2$. If B_2 is false, then B_1 must also be false. This means by the induction hypothesis that $\mathbf{m}[B_1]$ is empty. But then the identity function is evidence for $B_1 \Rightarrow B_2$.

(4) B is $\forall x.B_1$

Since this is true, we know that for every element a of D,

$$\mathbf{m} \models B_1 \ for \ s[s' = a].$$

By the axiom of choice there is a function f such that $f(a) \epsilon \ \mathbf{m}[B_1](s[x := a])$.

(5) B is $\exists x.B_1$

For B to be true, there must be some a in D on which B_1 is true. By the induction hypothesis, there is $a \ b_1 \epsilon \ \mathbf{m}[B_1](s[x := a])$. Then $< a, b_1 >$ is evidence for B_1.

qed.

3 Proofs

We now want to show that proofs are notations for evidence. They are expressions which denote objects and thus have direct mathematical meaning. The explanation of proofs comes in three parts. We define first their simple algebraic structure. Then we discuss the conditions needed to guarantee that they are meaningful expressions and to determine what they prove. The statement of these conditions corresponds most closely to what we think of as "proof rules." The format suggests also rules for determining type correctness of expressions. Finally we give the meaning of proof expressions with respect to a model. The method here is similar to that for giving meaning to algebraic expressions or to programs. We can in fact use rewrite rules to define most of the constructors.

The Syntax of Proof Expressions

Let $a, b, c, d, e, f, g, p, l$ range over proof expressions and let m, v, w, x, y denote variables (we will use x, y to denote ordinary variables over D and m, v, w, z to denote variables over proof expressions). Let A, B, C, G, L, R denote formulas. Then the following are proof expressions: variables z, w and

$andin(L : l;\ R : r)$ $\quad$ $andel(P : p;\ u : L,\ v : R.G : g)$

$somein(D : w;\ B : b)$ $\quad$ $somel(P : p;\ x : D, v : B.G : g)$

$impin(z : A.B : b)$ $\quad$ $impel(F : f;\ A : a;\ u : B.G : g)$

$allin(x : D.B : b)$ $\quad$ $allel(F : f;\ D : a;\ x : D,\ u : B.G : g)$

$orinl(L : l)$ $\quad$ $orel(T : d;\ u : L.G : g_1;\ v : R.G : g_2)$

$orinr(R : r)$

$absurd(u)$

$seq(S : s;\ u : S.G : g)$

$magic(F)$

In $andel(P : p\ ;\ l : L, v : R.G : g)$ the variables u, v are *binding occurrences* whose scope is $G : g$ so that all occurrences of u *and* v in G or g are *bound.* In $somel(P : p\ ;\ x : D, v : B.G : g)$ x, v are binding occurrences whose scope is $G : g$; so an occurrence of x in G is bound by $x : D$. In $impin(z : A.B : b)$ and $allin(x : D.B : b)$, z and x are binding occurrences whose scope is $B : b$. In $impel(F : f\ ; A : a\ ;\ u : B.G : g)$ and $allel(F : f\ D : a\ x : D,\ u : B.G : g)$, x, u are binding occurrences whose scope is $G : g$. In $orel(T : d;\ u : L.G : g1;\ v : R.G : g2)$ u is a binding occurrence whose scope is $G : g_1$ and v is one whose scope is $G : g_2$. In $seq(S : s;\ u : s.G : g)$ u is a binding occurrence whose scope is $G : g$.

To preserve the pattern of the notation we introduce a name for the domain of discourse D. For simplicity we take D to be the name of itself; this should cause no confusion since we could take a

name like "domain" for D in the proof expressions and agree that the meaning of "domain" is D.

Correctness Restrictions

We impose certain restrictions on the parts of these expressions when we say what it means for a proof expression a to prove a formula A. For example in $impel(F : f, A : a; u : B.G : g)$ the expression F must be an implication, say $A \Rightarrow B$, and f must denote a proof F and a proof of A. The result is a proof expression for G. The constructor name, *impel*, is mnemonic for *implication elimination*, which is a rule usually written in logic books as shown below (and sometimes called *modus ponens* in the special case when G is B.

$$\frac{A, \quad A \Rightarrow B}{B}$$

In the implication introduction form, $impin(z : A.B : b)$, it must be that z denotes an assumption of a formula A and b a proof expression for B, and the $impin(z : A.B : b)$ is a proof expression for $A \Rightarrow B$. The expression b may use assumption z. One might think of z as a label for the assumption. In an informal proof we might see these elements in this relationship:

show $A \Rightarrow B$

assume $z : A$

show B

$\vdots$

B

qed

The proof of B from assumption z actually shows how to build a proof expression b which may refer to the label z. For example here is an informal proof $A \Rightarrow A$.

show $A \Rightarrow A$

assume $z : A$

A by assumption z

qed.

The proof expression built by this derivation is $impin(z : A.A : z)$.

It is interesting to note that the part of a derivation that is sometimes called the *justification* [BC85,CO78] corresponds closely to the proof expression. For example, suppose we look at this fragment of a proof

$$A : a$$
$$\vdots$$
$$B : b$$
$$A\&B \text{ by } \textit{and introduction} \text{ from } a, b.$$

The rule name, *and introduction*, is used in conjunction with labels (or expressions themselves) to justify this line of the proof.

Generally in a proof expression, the operator names, such as *andin, andel* etc., correspond to the names of inference rules. Subexpressions of the form $z : A$ correspond to *labeled assumptions*, and subexpressions of the form $B : b$ correspond to subproofs of B by b. Thus we can read the following informal proof as the counterpart of the proof expression $impin(z : A.(B \Rightarrow A) : impin(u : B.A : z))$

show $A \Rightarrow (B \Rightarrow A)$ by

assumption z that A holds

show $(B \Rightarrow A)$ by

assumption u that B holds

show A by assumption z.

In short, variables occur to the left of the colon and indicate an assumption of the formula while proof expressions appear to the right of the colon and indicate why the formula is true in the context of all of the assumptions in whose scope the formula lies.

The correctness conditions on proof expressions are given by rules that are thought of as *proof rules*. Thus the rule for *and introduction* is written

$$\frac{A \quad B}{A\&B}$$

The formulas above the line are the hypotheses, those below the line are conclusions. If we include the proof expressions as justifications, we would get a rule of the form

$$\frac{A\ by\ a \quad B\ by\ b}{A\&B\ by\ andin(A : a; B : b)}$$

This last rule shows the pattern that we will adopt. But one additional feature is needed to keep track of the variables. A proof expression such as $impin(z : A.B : y)$ has a free variable y in it. This represents some undischarged assumption. There are no such variables in a completed proof. But at some points in building a proof expression, there will be free variables and we must keep track of them. We must know what formula or what type the variable refers to so that the type conditions and correctness conditions can be checked. Thus it is usual in presenting a proof to have

a mechanism for indicating the assumptions and variable bindings known at any point. This is done by keeping an environment with every rule and showing how the rules change the environment.

Environments will be represented as lists of variable bindings $x_1 : A_1, \ldots, x_n : A_n$. The A_i are either the domain D or formulas. The type bindings arise from *all introduction* while the formula bindings arise from *implication introductions*.

The use of environments may be familiar from certain logic texts. For example, they appear explicitly in Gentzen's sequent calculus [Kle52]. They are carefully defined in refinement logics [BC85]. In programming logics like PL/CV [CO78] they appear as they do in block structured programming languages. Some textbooks on natural deduction introduce the analogue of a block at least for propositional arguments.

The format we adopt for rules is to take as basic units the triple consisting of: a proof expression, the formula it proves and the environment for the variables, written together as

$$\text{from } H \text{ infer } A \text{ by } a$$

where a is a proof expression, A is a formula and H is a list of bindings, $x_1 : A_1, \ldots, x_n : A_n$. We sometimes isolate a specific binding by writing the environment as $H, x : A, H'$ where H, H' are the surrounding context. We call these basic units *sequents*, following Gentzen. Let $S_1, S_2, \ldots$ denote them.

A rule has the form of a production as is customary in logic:

$$\frac{S_1, \ldots, S_n}{S}$$

The S_i are the hypothesis; S is the conclusion. Here are the rules. These define the relationship a ***is a proof expression for*** A inductively.

Rules

1. $\dfrac{\textit{from } H \textit{ infer } L \textit{ by } l \quad \textit{from } H \textit{ infer } R \textit{ by } r}{\textit{from } H \textit{ infer } L\&R \textit{ by } andin(L{:}l;\ R{:}r)}$

2. $\dfrac{\textit{from } H \textit{ infer } P \textit{ by } p \quad \textit{from } H,\ x{:}\ L,\ y{:}R \textit{ infer } G \textit{ by } g}{\textit{from } H \textit{ infer } G \textit{ by } andel(P{:}p;\ x{:}L,\ y{:}R.G{:}g)}$

3. $\dfrac{\textit{from } H \textit{ infer } B[w/x] \textit{ by } b \quad \textit{for } w \textit{ a term}}{\textit{from } H \textit{ infer } \exists x.B \textit{ by } somin(D{:}w;B{:}b)}$

4. $\dfrac{\textit{from } H \textit{ infer } P \textit{ by } p \quad \textit{from } H,\ x{:}D,\ y{:}B \textit{ infer } G \textit{ by } g}{\textit{from } H \textit{ infer } G \textit{ by } somel(P{:}p;\ x{:}D,y{:}B.G{:}g)}$

5. $\dfrac{\textit{from } H,\ x{:}A \textit{ infer } B \textit{ by } b}{\textit{from } H \textit{ infer } A{\Rightarrow}B \textit{ by } impin(x{:}A;B{:}b)}$

6. $\dfrac{\textit{from } H \textit{ infer } A{\Rightarrow}B \textit{ by } f \quad \textit{from } H \textit{ infer} A \textit{ by } a \quad \textit{from } H,y{:}B \textit{ infer } G \textit{ by } g}{\textit{from } H \textit{ infer } G \textit{ by } impel(A{\Rightarrow}B{:}f;\ A{:}a;y{:}B.G{:}g)}$

7. $\dfrac{\textit{from } H,\ x{:}D \textit{ infer } B \textit{ by } b}{\textit{from } H \textit{ infer } \forall x.B \textit{ by } allin(x{:}D.B{:}b)}$

8. $\dfrac{\textit{from } H \textit{ infer } \forall x.B \textit{ by } f \quad \textit{from } H,x{:}D,u{:}B \textit{ infer } G \textit{ by } g \quad \textit{for } a \textit{ a term}}{\textit{from } H \textit{ infer } G \textit{ by } allel(\forall x.B{:}f;\ D{:}a;\ x{:}D,u{:}B.G{:}g)}$

9. $\dfrac{\textit{from } H \textit{ infer } L \textit{ by } l}{\textit{from } H \textit{ infer } L|R \textit{ by } orinl(L{:}l)}$

10. $\dfrac{\textit{from } H \textit{ infer } R \textit{ by } r}{\textit{from } H \textit{ infer } L|R \textit{ by } orinr(R{:}r)}$

11. $\dfrac{\textit{from } H \textit{ infer } L|R \textit{ by } d \quad \textit{from } H,u{:}L \textit{ infer } G \textit{ by } g_1 \quad \textit{from } H,\ v{:}R \textit{ infer}G \textit{ by } g_2)}{\textit{from } H \textit{ infer } G \textit{ by } orel(L|R{:}d;\ u{:}L.G{:}g_1;\ v{:}R.G{:}g_2)}$

12. $\textit{from } H, x : \textit{false infer } G \textit{ by } absurd(x)$

13. $\textit{from } H \textit{ infer } P \mid \neg P \textit{ by } magic(P)$

14. $\dfrac{\textit{from } H \textit{ infer } S \textit{ by } s \quad \textit{from } H,u{:}S \textit{ infer } G \textit{ by } g}{\textit{from } H \textit{ infer } G \textit{ by } seq(S{:}s;\ u{:}S.G{:}g)}$

A proof expression is built inductively using the constructors starting from the axiom *and* ob-

serving the correctness restrictions. These restrictions can be thought of as type restrictions on the formation of proof expressions. We give an example.

$\neg\forall x.\neg B(x) \Rightarrow \exists x.B(x)$:

$impin(h : \neg\forall x.\neg B(x).$

$\exists x.B(x) :\ seq(\exists x.B(x) \mid \neg\exists x.B(x) : magic(\exists x.B(x));$

$d1 : \exists x.B(x) \mid \neg\exists x.B(x).$

$\exists x.B(x) :\ orel(\exists x.B(x) \mid \neg\exists x.B(x) : d1\ ;$

$u : \exists x.B(x).\ \exists x.B(x) : u\ ;$

$v : \neg\exists x.B(x).$

$\exists x.B(x) : seq(false :$

$impel(\forall x.\neg B(x) \Rightarrow false : h\ ;$

$\forall x.\neg B(x) : allin(x : D; \neg B(x) :$

$seq(B(x) \mid \neg B(x) :\ magic(B(x));$

$d2 : B(x) \mid \neg B(x).\neg B(x) :$

$orel(B(x) \mid \neg B(x) : d2;$

$z : B(x).\neg B(x) :$

$impel(\exists x.B(x) \Rightarrow false : v$

$\exists x.B(x) : somin(D : x; B(x) : z);$

$u : false.\neg B(x) : absurd(u))$

$w : \neg B(x).\ \neg B(x) : w)$

$u : false.false : u)$

$v : false\ .\ \exists x.B(x) : absurd(v))\ \ ))$

the proof expression without the subformulas displayed is:

$inpin(h.$

$seq(magic(\exists x.B(x));$

$d1.orel(d1;$

$u.u;$

$v.seq(impel(h;$

$allin(x.seq(magic(B(x));$

$d2.orel(d2;$

$z.impel(v;$

$somin(x,z);$

$u.absurd(u));$

$w.w));$

$u.u);$

$v.absurd(v))\ ))$

Semantics of Proof Expressions

We now assign meaning to proof expressions with respect to some model. The definition is given inductively on the structure of the expression and is only given for proof expressions which are correct, i.e. only for expresisons a for which we know that there is a formula A such that a proves A. We will not know that the definition makes sense until Theorem 2 is proved.

In the course of the definition we must apply the meaning function over a model $\mathbf{m}$ to a function body. To explain this we extend the definition of a state to account for variables ranging over formulas. We want to say that $s(z) \epsilon\ \mathbf{m}[A](s')$. But now A may depend on other variables over D whose meaning is given by a state, say s'.

We observe that the variables occurring in a proof expression a and in a formula A which it

proves can be listed in order of dependency. For simplicity, assume that all variables, both free and bound, are uniquely and uniformliy named, say $x_1, x_2, x_3, \ldots,$. Let A_i be the type or formula over which x_i ranges. Then these can be listed in order, for simplicity $A_1, A_2, \ldots$ such that there are no free variables in A_1, only x_1 is free in A_2, only x_1, x_2 are free in A_3, etc. Let us call this a *cascade of variables* and write it as $x_1 : A_1, x_2 : A_1(x_1), \ldots, x_n : A_n(x_1, \ldots, x_{n-1})$. Now a state s will map x_i into $\mathbf{m}[A_i(x_i, \ldots, x_1)](s)$ and the appearence of s in the definition of $s(x_i)$ will be sensible. For the remainder of this section we assume that we are dealing with such a state. Now give the meaning of a proof expression with respect to a model and a state.

1. $\mathbf{m}(andin(L : l; R : r))(s) =< \mathbf{m}(L : l)(s), \mathbf{m}(R : r)(s) >$

2. $\mathbf{m}(andel(P : p; u : L, v : R.G : g))(s) =$
   $\mathbf{m}(g)(s[u := 1of\ (\mathbf{m})(p)(s)), v := 2of\ (\mathbf{m}(p)(s))])$

3. $\mathbf{m}(somin(D : w; B : b))(s) =$
   $< \mathbf{m}(w)(s), \mathbf{m}(b)(s) >$

4. $\mathbf{m}(somel(P : somin(D : w; B : b);\ x : D, y : B.G : g))(s) =$
   $\mathbf{m}(g)(s[x := 1of\ (\mathbf{m}(p)(s)), y := 2of\ (\mathbf{m}(p)(s))])$

5. $\mathbf{m}(impin(z : A; B : b))(s) = \lambda x : \mathbf{m}(A)(s).\mathbf{m}(b)(s[z := x])$

6. $\mathbf{m}(impel(F : f; A : a; u : B.G : g))(s) =$
   $\mathbf{m}(g)(s[u := (\mathbf{m}(f)(s))(\mathbf{m}(a)(s))])$

7. $\mathbf{m}(allin(z : D; B : b))(s) = \lambda x : D.\mathbf{m}(b)(s[z := x])$

8. $\mathbf{m}(allel(F : f; D : a; x : D, u : B.G : g))(s) =$
   $\mathbf{m}(g)(s[u := (\mathbf{m}(f)(s))(\mathbf{m}(a)(s))])$

9. $\mathbf{m}(orinl(L : l))(s) = inl(\mathbf{m}(l)(s))$

10. $\mathbf{m}(orinr(R : r))(s) =\ inr(\mathbf{m}(r)(s))$

11. $\mathbf{m}(orel(T : t; u : L.G : g_1; v : R.G.g_2))(s) =$

    $\mathbf{m}(g_1)(s[u := l])\ if\ \mathbf{m}(t)(s)\ is\ inl(l)$

    $\mathbf{m}(g_2)(s[v := r])\ if\ \mathbf{m}(t)(s)\ is\ inr(r)$

12. $\mathbf{m}(absurd(u))(s) =$ any $(s(u))$ when any maps the empty set into any set

13. $\mathbf{m}(magic(P))(s) =$ an element of $\mathbf{m}(P)(s) + \mathbf{m}(\neg P)(s)$

14. $\mathbf{m}(seq(T : t; u : T.G : g))(s) =$

    $\mathbf{m}(g)(s[u := \mathbf{m}(t)(s)])$

The operations inl, inr are injections of a set into its disjunct in a disjoint union, i.e.

$$inl : L \rightarrow (L + R)$$
$$inr : R \rightarrow (L + R)$$

We know by correctness that *orinl* applies only if l is a proof expression for L and $orinl(l)$ is one for $L|R$. Similarly for *orinr*. So the mappings make sense in clauses 9 and 10.

In $orel(T_1, d : ;\ u.T_2.G : g_1\ ; v : T_3.G : g_2)$ we know that d must be a proof expression for a disjunction, so T_1 is $A|B$. Thus $\mathbf{m}(d)(s)$ will be a member of $\mathbf{m}[A|B](a)$ as we show. Thus $\mathbf{m}(d)(s)$ is either $inl(a)$ or $inr(b)$ for a in $\mathbf{m}[A](s)$ and b in $\mathbf{m}[B](s)$.

The analysis of $somel(P : p\ ;\ x : D,\ y : B.G : g)$ is just as for *andel*. We know that $\mathbf{m}(p)(s)$ is a pair consisting of an element of D and evidence that the element is a witness for an existential quantifier. In case of $magic(A)$, we must use the axiom of choice to pick out an element of the inhabited type. We conclude this section with a theorem that shows that the meaning of a proof expression is well-defined when the proof expression proves a formula.

Theorem 2 *If a is a proof expression for formula A, and if $x_1 : A_1, x_2 : A_1(x_1), \ldots, x_n : A_1(x_1, \ldots, x_{n-1})$ is a cascaded enumeration of the free variables of a and A with their bindings, and if* $\mathbf{m}$ *is any model and s any state assigning $s(x_i)$ to $\mathbf{m}[A_i(x_i, \ldots, x_{i-1})](s)$, then $\mathbf{m}(a) \epsilon \mathbf{m}[A](s)$.*

proof

The proof is by induction on the structure of the proof expression a. In the base case, a is some variable x_i or $magic(C)$ for a formula C. If a is a variable, then by hypotheses $\mathbf{m}(a)(s) = s(a) = s(x_i) \epsilon \mathbf{m}[A_i(x_1, \ldots, x_{i-1})](s)$. If a is $magic(C)$, then A is $C|\neg C$ and $\mathbf{m}(a)(s) \epsilon \mathbf{m}[C](s) + \mathbf{m}[\neg C](s)$.

Now consider the induction case. We assume that the result holds for any subexpression b of a, in any state s' assigning proper values to all free variables of b as required by the antecedent of the induction hypothesis.

$$induction\ hypothesis : assume\ \mathbf{m}(b)(s') \epsilon \mathbf{m}[B](s)$$

where B is the formula proved by b for b a subexpression of a.

We proceed by cases on the outer structure of a (see the syntax of proof expressions).

1. a is $andin(L : l; R : r)$

   Then A must be the conjunction $L\&R$ because otherwise a would not be a proof expression for A. By the induction hypothesis the subexpressions of a satisfy the theorem, so $\mathbf{m}(l)(s) \epsilon \mathbf{m}[L](s)$ and $\mathbf{m}(r)(s) \epsilon \mathbf{m}[R](s)$. But then the result holds by the definition that $\mathbf{m}(andin(L : l; R : r))(s) =< \mathbf{m}(l)(s), \mathbf{m}(r)(s) > \epsilon \mathbf{m}[L\&R](s)$.

2. a is $andel(P : p; u : L, v : R.G : g)$

   Then A is G according to the typing rules. Moreover, P must be the conjunction $L\&R$,

L and R must be formulas, p must be a proof expression for $L\&R$, and g is a proof expression for G in which variables u, v can occur. We also know by the typing rules that $u : L, v : R$ are hypotheses in typing g. Now consider any state s' which extends s and assigns $s'(u)$ to $\mathbf{m}(L)(s)$ and $s'(v)$ to $\mathbf{m}(R)(s)$. For this state and for g, the induction hypothesis holds, so that $\mathbf{m}(g)(s')\epsilon\ \mathbf{m}(G)(s')$. By definition, $\mathbf{m}(andel(P : p; u : L, v : R.Gg))(s) = \mathbf{m}(g)(s[u := 1of\ (\mathbf{m}(p)(s)), v := 2of\ (\mathbf{m}(p)(s))])$, and $s[u := 1of\ (\mathbf{m}(p)(s)), v := 2of\ (\mathbf{m}(p)(s))]$ is a state satisfying the condition on s' since $\mathbf{m}(p)(s)\epsilon\ \mathbf{m}[L\&R](s)$ is true by the induction hypothesis.

3. a is $somin(D : w; B : b)$

   Then according to the typing rules, A must be $\exists x.B$, w must be a term and b is a proof expression for $B[w/x]$. By the induction hypothesis, $\mathbf{m}(b)(s)\epsilon\ \mathbf{m}[B[x/x]](s)$, so by definition the meaning of a belongs to the right set.

4. a is $somel(P; p; x : D, y : B.G : g)$

   Then by the typing rules, P must be the existential statement $\exists x.B(x)$ and g is a proof expression for G using x a variable over D and y a variable over B. By the induction hypothesis, the meaning of p is a pair, say $< \mathbf{m}(w)(s), b >$ with $\mathbf{m}(w)(s)$ in D and b in $\mathbf{m}[B[w/x]](s)$. Just as in 2 we conclude $\mathbf{m}(a)(s)\epsilon\ \mathbf{m}[A](s)$.

5. a is $impin(z : P; B : b)$

   Then A is $P \Rightarrow B$ and b is a proof expression for B which can use the variable z assumed to range over $\mathbf{m}[P](s)$. This means that by the induction hypothesis, $\mathbf{m}(b)(s[z := l])$ is in $\mathbf{m}[B](s)$ for all l in $\mathbf{m}[P](s)$. Thus $\lambda x : \mathbf{m}[P](s).\mathbf{m}(b)(s[z := x])$ is a function from $\mathbf{m}[P](s)$ into $\mathbf{m}[B](s)$ as required to prove this case.

6. a is $impel(F : f; P : p; u : B.G : g)$

   By the typing rules, F must be the implication $P \Rightarrow B$, f is a proof expression for it, and the induction hypothesis, $\mathbf{m}(f)(s)\epsilon\ \mathbf{m}[F](s)$. Also, $\mathbf{m}(p)(s)\epsilon\ \mathbf{m}(P)(s)$ and for all states s' extend-

ing s and assigning u a value in $\mathbf{m}[B](s)$, $\mathbf{m}(g)(s')\epsilon\ \mathbf{m}[G](s)$. Let b denote $(\mathbf{m}(f)(s))(\mathbf{m}(p)(s))$, since b belongs to $\mathbf{m}(B)(s)$, then $\mathbf{m}(g)(s[u := b])\epsilon\ \mathbf{m}[G](s)$; so the result holds.

7. a is $allin(z : D; B : b)$

   This case is like 5.

8. a is $allel(F : f; P : p; x : D, y : B.G : g)$

   This case is like 6.

9. a is $orinl(L : l)$

   Then A must be $L \mid R$ and l is a proof expression for L. So by the induction hypothesis $\mathbf{m}(l)(s)\epsilon\ \mathbf{m}(L)(s)$. Thus $inl(\mathbf{m}(l)(s))$ belongs to $\mathbf{m}[A](s)$ as required.

10. a is $orin(R : r)$

    This case is like 9.

11. a is $orel(T : t; u : L.G : g_1; v : R.G : g_2)$

    By the typing rules, T must be $L \mid R$, and by the induction hypothesis $\mathbf{m}(t)(s)\epsilon\ \mathbf{m}(T)(s)$. Thus t is either $inl(l)$ or $inr(r)$. Also by the typing rules we know that g_1 and g_2 are proof expressions for G under variable bindings for u and v so by induction hypothesis we know that $\mathbf{m}(g_1)(s[u := l])\epsilon\ \mathbf{m}(G)(s)$ and $\mathbf{m}(g_2)(s[v := r])\ \epsilon\ \mathbf{m}(G)(s)$. This is what is needed to show $\mathbf{m}(a)(s)\epsilon\ \mathbf{m}[A](s)$.

12. a is $absurd(u)$

    By the typing rules, A is proved from the hypothesis $u : false$. That is, A is proved under the assumption that there is something in the empty set, $s(u)\epsilon\phi$. But there is a trivial map from the empty set to any set, call it any, so $any(s(u))$ belongs to any set, in particular to $\mathbf{m}[\![A]\!](s)$.

13. a is $magic(P)$

    Then A must be $P \mid \neg P$ and by definition $\mathbf{m}(magic(P))(s)\epsilon\ \mathbf{m}[\![P]\!](s) + \mathbf{m}[\![\neg P]\!](s)$ which is $\mathbf{m}[\![A]\!](s)$.

14. a is $seq(T : t; u : T.G : g)$

    By the induction hypothesis $\mathbf{m}(t)(s) \epsilon \mathbf{m}[\![T]\!](s)$ and $\mathbf{m}(g)(s') \epsilon\ \mathbf{m}(G)(s)$ for any s' where $s'(u) \epsilon\ \mathbf{m}[\![T]\!](s)$

    thus $\mathbf{m}(g)(s[u := \mathbf{m}(t)(s)])$ since $s[u := \mathbf{m}(t)(s)]$ satisfies the condition for s'.

qed.

Computational Semantics and Constructive Logic

With the exception of $magic(A)$, all proof expressions are given meaning in terms of recursively defined equations. For example, $m(andin(L : l;\ R : r))(s) =< m(L : l)(s),\ m(R : r)(s) >$.

If the law of excluded middle, $P \mid \neg P$, is removed from the predicate logic, then we know that in some sense the underlying theory of evidence is *computable*. If we add expressions and rules which can be explained in terms of computable evidence, then the entire theory can be explained this way.

Predicate logics without the law of excluded middle or its equivalents are in some sense *constructive*, sometimes they are called *Intuitionistic logics* after Brouwer [Bro23]. Arithmetic based on this logic and the Peano axioms is called *Heyting arithmetic* after the Intuitionist A. Heyting [Hey66]. These topics are treated thoroughly in Kleene [Kle52], Dummett [Dum77] and Troelstra [Tro73]. Analysis built on such a logic extended to higher order is sometimes called *constructive analysis* see Bishop [Bis67]. These topics are discussed in Troelstra [Tro73] and Bridges [BB85].

Programming

The PRL programming systems built at Cornell in the early 1980's [BC85,ML84] are based on the idea that formal constructive logic, because of its conputational semantics, provides a new kind of very high level programming language. This idea was first explored in Constable [Con71] and

Bishop [Bis70]. It was later developed by Bates and put into practice by Bates and Constable [BC85]. The semantics of evidence discussed here is quite close to the actual implementation ideas in Nuprl [ML82].

Acknowledgements

I want to thank Ryan Stansifer and Todd Knoblock for helpful comments and Elizabeth Maxwell for preparing the manuscript.

References

[Acz78] Peter Aczel. The type theoretic interpretation of constructive set theory. In *Logic Colloquium '77*, pages 55–66. Amsterdam:North-Holland, 1978.

[BB85] Errett Bishop and Douglas Bridges. *Constructive Analysis.* NY:Springer-Verlag, 1985.

[BC85] Joseph L. Bates and Robert L. Constable. Proofs as programs. *ACM Trans. Program. Lang. and Syst.*, 7(1):53–71, 1985.

[Bis67] E. Bishop. *Foundations of Constructive Analysis.* McGraw-Hill, New York, 1967.

[Bis70] E. Bishop. Mathematics as a numerical language. In *Intuitionism and Proof Theory.*, pages 53–71. NY:North-Holland, 1970.

[Bro23] L.E.J. Brouwer. On the significance of the principle of excluded middle in mathematics. In *J. fur die Reine und Angewandte Math*, volume 154, pages 1–2, 1923.

[CAB+86] Robert L. Constable, S. Allen, H. Bromely, W. Cleveland, and et al. *Implementing Mathematics with the Nuprl Development System.* NJ:Prentice-Hall, 1986.

[CO78] Robert L. Constable and Michael J. O'Donnell. *A Programming Logic.* Mass:Winthrop, 1978.

[Con71] Robert L. Constable. Constructive mathematics and automatic program writers. In *Proc. IFP Congr.*, pages 229–33, Ljubljana, 1971.

[deB80] N.G. deBruijn. A survey of the project automath. *Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 589–606, 1980.

[Dum77] M. Dummet. *Elements of Intuitionism, Oxford Logic Series.* Claredon Press, Oxford, 1977.

[Gir71] J-Y. Girard. Une extension de l'interpretation de godel a l'analyse, et son application a l'elimination des coupures dans l'analyse et la theorie des types. In *2nd Scandinavian Logic Symp.*, pages 63–69. NY:Springer-Verlag, 1971.

[Hey66] A. Heyting. *Intuitionism.* North-Holland, Amsterdam, 1966.

[Kle52] Stephen C. Kleene. *Introduction to Metamathematics.* Princeton:van Nostrand, 1952.

[LS86] J. Lambeck and P. Scott. *Introduction to higher order categorical logic.* Cambridge University Press, Cambridge, 1986.

[ML82] P. Martin-Lof. Constructive mathematics and computer programming. In *Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–75. Amsterdam:North Holland, 1982.

[ML84] P. Martin-Lof. *Intuitionistic Type Theory.* Bibioplois, Napoli, 1984.

[Rus03] B. Russell. Mathematical logic as based on a theory of types. *Am. J. Math.*, 30:222–62, 1903.

[Sco70] D. Scott. Constructive validity. In *Symp. on Automatic Demonstration, Lecture Notes in Math.*, volume 125, pages 237–275. Springer-Verlag, 1970.

[Tai67] W. Tait. Intensional interpretation of functionals of finite type. In *J. Symbolic Logic*, volume 32(2), pages 187–199, 1967.

[Tar44] A. Tarski. The semantic conception of truth and the foundations of semantics. *Philos. and Phenom. Res.*, 4:341–376, 1944.

[Tro73] A. Troelstra. *Metamathematical Investigation of Intuitionistic Mathematics.* Springer-Verlag, New York, 1973.

[vD80] D. van Daalen. *The Language Theory of AUTOMATH.* PhD thesis, Tech. University of Edinburgh, 1980.