The Semantics of Evidence

Robert L. Constable
TR 85-684
May 1985

Department of Computer Science
Cornell University
Ithaca, New York 14853
THE SEMANTICS OF EVIDENCE

Robert L. Constable
Cornell University
Ithaca, NY 14853

ABSTRACT
The usual meaning of a sentence in the predicate calculus is its truth value. In this paper we show that there is associated with every statement a set of elements comprising evidence for it. A statement is true in a model exactly when there is evidence for it. Proofs can be regarded as expressions which denote evidence. A statement is constructively true when the evidence can be computed from its proofs. Proofs are useful in practical computations when evidence for statements is needed. They are especially valuable in relating computations to the problems they solve.*

I. INTRODUCTION
If I correctly state the winning lottery number before it has been publicly announced, many people will be more interested in my evidence for the assertion 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 [22] who put the matter this way for the case of a universal statement: \( \forall x. B(x) \) is true in a model \( m \) if the model assigns to \( B \) some propositional function \( m(B) \) 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 \( xy \) is rational. I might prove that they exist this way. Consider \( \sqrt{2^2} \). It is either rational or irrational. If it is rational, take \( x = \sqrt{2}, y = \sqrt{2} \). If it is irrational, take \( x = \sqrt{2^2} \) and \( y = \sqrt{2} \). Then \( xy = (\sqrt{2^2})^2 = \sqrt{2^2} \). 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

*This research was supported in part by NSF grants MCS-81-04018 and (joint Cornell-Edinburgh) MCS-83-03336.
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 $y = 2 \cdot \log_2 3$. The evidence would include this pair of numbers and some argument that $\sqrt{2}$ and $2 \cdot \log_2 3$ are irrational plus the computation $2^{2 \cdot \log_2 3 \sqrt{2}} = 3$.

We can come to understand how the concept of evidence might be defined if we examine how it is used 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 $p, p + 2$ bigger 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 bigger 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 $n$. Evidence for a disjunction 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.

Implications are a bit more subtle. Suppose I say "if there is a 6 in the list $A$, then there is a six on the list $B$ obtained by replacing all odd numbers on $A$ by 5". One kind of evidence for this would be a function which took any evidence for the hypothesis, such as an index $i$ of 6 on $A$, say $A[i] = 6$ and mapped it to evidence for the conclusion, say mapped it to the same index and to $B[i] = 6$. We are going to treat the negation of $A$, in symbols $\neg A$, as $A$ implies false. So if we understand that there is no evidence for false, then we have some kind of explanation of the evidence for $\neg A$.

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 III we examine proofs and show how they encode evidence.

This interpretation 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 by 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

†A perfect number is one which is equal to the sum of its proper factors, e.g. $6 = 3 + 2 + 1$. 
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, Lauchli, and de Bruijn (for references see [10, 11]). This principle can be seen as formalizing the notion of evidence in type theory, see also Martin-Löf[18], Constable et al. [10].

II. THE LOGIC

Syntax

We present a standard kind of predicate calculus, see [15]. The formulas of our logic are built using the binary propositional connectives \&, |, \neg, \Rightarrow (and, or, not, implies) and the quantifiers \forall x, \exists x (all and some). There are predicate constants A,A(x), A(x,y),..., B,B(x), B(x,y),..., C,C(x), C(x,y). Each predicate has an arity which is its number of argument positions, e.g. an arity n predicate with variables in the argument positions appears in formulas as B(x_1,...,x_n) showing that the argument positions are given by the x_i. There are also terms, usually denoted t_1, t_2,... These are built from variables x, y, z,... and constants c_1,c_2,c_3,...,c.. Constants also have an arity; an arity n constant c is used to build a term of the form c(t_1,..,t_n) where t_i are terms.

A formula of the logic is defined as follows.

(i) false is a formula

(ii) if B is a predicate constant of arity n, and t_1,...,t_n are terms, then B(t_1,...,t_n) is a formula

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

(A & B)

(A | B)

(A \Rightarrow B)

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

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

For example, over the set of natural numbers,\{0,1,2,...\}, x = y and x < y are arity 2 predicates on N and + (x,y) and *(x,y) are arity 2 terms on N, then \forall x.\exists y. (+ (x,y) *(x,y)) is a formula. 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, is \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 [15,17,18] 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 $\forall x. B(x) & C \Rightarrow \exists y. P(x,y) \| B(y)$ abbreviates $(\forall x. (B(x) & C) \Rightarrow (\exists y. (P(x,y) \| B(y))))$.

The meaning of a formula is given with respect to a model $m$ (or structure or interpretation) which consists of a set $D$, called the domain of discourse, and for each constant $c$ of arity $n$ a function from $D^n$ into $D$ denoted $m(c)$ and for each predicate constant $B$ of arity $n$ a function from $D^n$ into $\{T,F\}$, denoted $m(B)$.

Semantics of Truth

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 the value of $a$ if $y = x_i$. We define the relation that formula $F$ is true in model $m$ and state $s$, written

$$m \models s F$$

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

Truth Conditions

1. $m \models s c(t_1,\ldots,t_m)$
   
   iff $m(c)(s)(m(t_1)(s),\ldots,m(t_m)(s)) = T$

   for $c$ a predicate constant of arity $m$. 


2. \( m \models_s (A \& B) \)
   \( \iff \ m \models_s A \text{ and } m \models_s B \)
3. \( m \models_s (A \mid B) \)
   \( \iff m \models_s A \text{ or } m \models_s B \)

4. \( m \models_s (A \Rightarrow B) \)
   \( \iff m \models_s A \text{ implies } m \models_s B \)

5. \( m \models_s \forall x.B \)
   \( \iff m \models_s B \text{ for all } s' = s[x_i = a] \text{ with } a \text{ in } D. \)

6. \( m \models_s \exists x.B \)
   \( \iff m \models_s B \text{ for some } s' = s[x_i = a] \text{ for } a \text{ in } D. \)

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 \( A \rightarrow B \) denote all the functions from \( A \) to \( B \). Given \( B(x) \) a family of sets indexed by \( A \), let

\[
\Pi_{x \in A} B(x)
\]

denote the set of functions \( f \) from \( A \) into,

\[
\Sigma_{x \in A} B(x)
\]

such that \( f(a) \) belongs to \( B(a) \). (We mean by

\[
\Sigma_{x \in A} B(x)
\]

the disjoint union of the family of sets.) Notice that these are all ordinary set operations.

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

1. \( m[\text{false}](s) = \text{the empty set} \)
2. \( m[\text{c}(t_1, \ldots, t_n)](s) = \{T\} \text{ if } m \models_s c(x_1, \ldots, x_n) \text{ otherwise} \) \( \text{for } c \text{ a predicate constant}. \)
3. \( m[A \& B](s) = m[A](s) \times m[B](s) \)
4. \( m[A \mid B](s) = m[A](s) + m[B](s) \)
5. \( m[A \Rightarrow B](s) = m[A](s) \rightarrow m[B](s) \)
6. \( m(\forall x. B)(s) = \Pi m(B)[s[x := y]] \quad y \in D \)

7. \( m(\exists x. B)(s) = \{ <a, b> | a \in m(A) \& b \in m(B)[s[x := a]] \} \)

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. In the base case, 1, the definition relies on the semantics of truth. Here is an example of evidence: \( m(\exists y. 5 < y)(s) = \{ <6, T> \} \). We need to know that \( 5 < 6 \) is true so that T belongs to \( m(5 < 6)(s) \). Truth and evidence are related in this simple way.

Theorem 1: For every sentence B, model m and state s

\[ m \models s B \iff \text{there is } b \in m(B)(s). \]

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 \( b \in m(B)(s) \), then \( m \models s B \). 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. (0) If \( B \) is atomic, then the result is immediate.

   (1) \( B \) is \( B_1 \& B_2 \)

   Then \( b \in m(B_1 \& B_2) \) so \( b \) is a pair, say \( <b_1, b_2> \) and \( b_1 \in m(B_1) \) and \( b_2 \in m(B_2) \). By induction then, \( m \models B_1 \) and \( m \models B_2 \) so \( m \models B_1 \& B_2 \).

   (2) \( B \) is \( B_1 \Downarrow B_2 \)

   Given \( b \in m(B_1 \Downarrow B_2) = m(B_1) + m(B_2) \), 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 \( f \in m(B_1 \Rightarrow B_2) \), we consider two cases. Either \( m(B_1) \) is empty or there is some \( b_1 \in m(B_1) \). In the later case \( f(b_1) \in m(B_2) \) so \( B_2 \) is true and so is \( B_1 \Rightarrow B_2 \). If \( m(B_1) \) is empty, then by the induction hypothesis \( B_1 \) is false. So \( B_1 \Rightarrow B_2 \) is true.

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

   Given

   \[ f \in \Pi m(B_1)(s)[x := v] \quad v \in D \]

   then for any \( a \in D, f(a) \in m(B_1)[s[x := a]] \). 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 \( \{ <a, b> | a \in D \& b \in m(B_1)[s[x := a]] \} \) 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 \( m \) and state \( s \), then there is evidence for \( m[B](s) \). 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 induction hypothesis there are \( b_1 \in m[B_1], b_2 \in m[B_2] \). By definition \( < b_1, b_2 > \in m[B_1 \& B_2] \).

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 \( m[B_1] \) or of \( m[B_2] \).

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 \( b_2 \in m[B_2] \). 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 \( m[B_1] \) 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 \( D \),
   
   \[ m \models B_1 \text{ in } s[x := a]. \]
   
   By the axiom of choice there is a function \( f \) such that \( f(a) \in m[B_1](s)[(x := a)] \).

5. \( B \) is \( \exists x. B \)
   
   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 \( b_1 \in m[B_1](s[x := a]) \). Then \( < a, b_1 > \) is evidence for \( B_1 \).

QED

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

\[
\begin{align*}
\text{andin}(a:A,b:B) & \quad \text{andell}(p:C) \\
\text{somein}(a,b:B) & \quad \text{somet}(p:C,f:F) \\
\text{orinl}(a:C) & \quad \text{orinr}(b:C) \\
\text{orel}(p:A,f:B,g:C) & \\
\text{impin}(z:A,b:B) & \quad \text{impel}(f:F,a) \\
\text{allin}(z,b:B) & \quad \text{allel}(f,F,a) \\
\text{absurd}(a:C) & \quad \text{magic}(B)
\end{align*}
\]

Variables over proof expressions are called \textit{assumptions}.

The operators \textit{impin} and \textit{allin} are binding operators, like quantifiers. The expression $b$ is their \textit{scope} and occurrences of the variable before the semi-colon are bound in the scope including those in the formulas $A,B,C$.

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 $\text{impel}(f:F,a:A)$ the expression $F$ must be an implication, say $A \Rightarrow B$, and $f$ must denote a proof of $F$ and $a$ proof of $A$. The result is a proof expression for $B$. The constructor name, \textit{impel}, is mnemonic for \textit{implication elimination}, which is a rule usually written in logic books as shown below (and sometimes called \textit{modus ponens}):

\[
\begin{array}{c}
A \quad A \Rightarrow B \\
\hline \\
B
\end{array}
\]

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

\[
\begin{align*}
\text{show } A \Rightarrow B \\
\text{assume } x:A. \\
\hline \\
B \\
\text{qed}
\end{align*}
\]
The proof of $B$ from assumption $x$ actually shows how to build a proof expression $b$ which may refer to the label $x$. For example here is an informal proof of $A \Rightarrow A$.

\[
\begin{align*}
\text{show } & \ A \Rightarrow A \\
\text{assume } & \ x : A \\
& \ A \text{ by assumption } x \\
\text{qed.}
\end{align*}
\]

The proof expression built by this derivation is $\text{impin}(x:A,x:A)$.

It is interesting to note that the part of a derivation that is sometimes called the justification [2,9] corresponds closely to the proof expression. For example, suppose we look at this fragment of a proof

\[
\begin{array}{c}
a : A \\
\vdots \\
\vdots \\
\vdots \\
b : B \\
A \& B \text{ by and introduction from } a,b.
\end{array}
\]

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

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

\[
\begin{array}{c}
A \\
B
\hline
\text{A}\&\text{B}
\end{array}
\]

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

\[
\begin{array}{c}
\text{A by a} \\
\text{B by b}
\hline
\text{A}\&\text{B by andin}(a:A,b:B)
\end{array}
\]

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 $\text{impin}(x:A,y:B)$ 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 [15]. They are carefully defined in refinement logics [2]. In programming logics like PL/CV [9] 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

\[
\text{a proves } A \text{ from } H
\]

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 Post 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**

and introduction

\[
a \text{ proves } A \text{ from } H \quad b \text{ proves } B \text{ from } H \\
\text{and}(a:A,b:B) \text{ proves } A\&B \text{ from } H
\]

and elimination

\[
p \text{ proves } A\&B \text{ from } H \\
\text{and}!l(p:A\&B) \text{ proves } A \text{ from } H
\]

or _introduction

\[
a \text{ proves } A \text{ from } H \\
\text{or}!l(a:A\mid B) \text{ proves } A\mid B \text{ from } H
\]
or_elimination

\[
\frac{d \text{ proves } A \lor B \text{ from } H \quad f_1 \text{ proves } A \Rightarrow C \text{ from } H \quad f_2 \text{ proves } B \Rightarrow C \text{ from } H}{\text{orel}(d:A\lor B, f_1:A \Rightarrow C, f_2:B \Rightarrow C) \text{ proves } C \text{ from } H}
\]

implication_introduction

\[
\frac{b \text{ proves } B \text{ from } H, x:A}{\text{impin}(x:a; b:B) \text{ proves } A \Rightarrow B \text{ from } H}
\]

implication_elimination

\[
\frac{f \text{ proves } A \Rightarrow B \text{ from } H \quad a \text{ proves } A \text{ from } H}{\text{impel}(f:A \Rightarrow B, a:A) \text{ proves } B \text{ from } H}
\]

false_elimination

\[
\frac{f \text{ proves false from } H}{\text{absurd}(f:A) \text{ proves } A \text{ from } H}
\]

all_introduction

\[
\frac{b \text{ proves } B \text{ from } H, x:A}{\text{allin}(x;b:B) \text{ proves } \forall x.B \text{ from } H}
\]

all_elimination

\[
\frac{f \text{ proves } \forall x.B \text{ from } H}{\text{allel}(f:\forall x.B, a) \text{ proves } B(a/x) \text{ from } H}
\]

some_introduction

\[
\frac{b \text{ proves } B(a/x) \text{ from } H}{\text{somein}(a, b:B(a/x)) \text{ proves } \exists x.B(x) \text{ from } H}
\]

some_elimination

\[
\frac{p \text{ proves } \exists x.B \text{ from } H \quad f \text{ proves } \forall x.(B \Rightarrow C) \text{ from } H}{\text{somell}(p:\exists x.B, f:\forall x.(B \Rightarrow C)) \text{ proves } C \text{ from } H}
\]

In addition to these constructors there are also the following axioms, i.e. forms with no subgoals.

hypothesis_rule

\[
x \text{ proves } A \text{ from } H, x:A, H'
\]

The rule constructor is simply \( x \).
law_of_excluded_middle

magic(A) proves A \vdash \neg A from H.

A proof expression is built inductively using the constructors starting from the axiom and observing the correctness restrictions. These restrictions can be thought of as type restrictions on the formation of proof expressions. We give an example.

<table>
<thead>
<tr>
<th>formula</th>
<th>justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>show P \Rightarrow (Q \Rightarrow P)</td>
<td>by imin(x:P)</td>
</tr>
<tr>
<td>assume x:P</td>
<td></td>
</tr>
<tr>
<td>show (Q \Rightarrow P)</td>
<td>by imin(y:Q)</td>
</tr>
<tr>
<td>assume y:Q</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>by x:P</td>
</tr>
<tr>
<td>qed</td>
<td></td>
</tr>
<tr>
<td>qed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The proof expression is imin(x:P; imin (y:Q; x:P): Q\Rightarrow P). We can think of x and y as labels for the assumptions.

Here is a more involved example. Again we first sketch the derivation which can be used to check the proof expression for correctness.

show \neg \forall x. \neg B(x) \Rightarrow \exists x. B(x)
by implication introduction, z1
  d1: \exists x. B(x) \vdash \exists x. B(x) by excluded middle
  show \exists x.A(x) by or elimination on d1
    show \exists x.B(x) \Rightarrow \exists x.B(x)
      by implication introduction, w1
    show \neg \exists x.B(x) \Rightarrow \exists x.B(x)
      by implication introduction, w2
    show \exists x.B(x) by false elimination
    show false by implication elimination on z1
    show \forall x. \neg B(x) by all introduction, x:A
  d2: B(x) \vdash \neg B(x) by excluded middle
  show \neg B(x) by or elimination on d2
    show B(x) \Rightarrow \neg B(x) by implication introduction, y1
    show \neg B(x) by false elimination
    show false by implication elimination, w2
    show \exists x.B(x) by somein x,y1
    show \neg B(x) \Rightarrow \neg B(x) by implication introduction, y2
the proof expression is

\[ \text{impin}(z1: \neg \forall x. \neg B(x), \text{orel}(\text{magic}(\exists x.B(x))): \exists x.B(x) \vdash \exists x.B(x)) \]

\[ \text{impin}(w1: \exists x.B(x), w1: \exists x.B(x)) \]
\[ : \exists x.B(x) \Rightarrow \exists x.B(x), \]

\[ \text{impin}(w2: \neg \exists x.B(x), \text{absurd}(\text{impel}(z1: \forall x. \neg B(x) \Rightarrow \text{false}, \text{orel}(\text{music}(B(x))): B(x) \vdash \neg B(x)) \]
\[ : \text{impin}(y1:B(x), \text{absurd}(\text{impel}(w2: \exists x.B(x) \Rightarrow \text{false, somein}(x, y1:B(x)): \neg B(x)) \]
\[ : \neg B(x) \]
\[ : B(x) \Rightarrow \neg B(x), \]

\[ \text{impin}(y2: \neg B(x), y2: \neg B(x)) : \neg B(x) \Rightarrow \neg B(x) \]
\[ : \forall x. \neg B(x) \]
\[ : \neg \exists x.B(x) \]
\[ : \exists x.B(x) \]
\[ : \exists x.B(x) \]
\[ : \neg \exists x.B(x) \Rightarrow \exists x.B(x) \]
\[ : \exists x.B(x) \]

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 expressions \( a \) 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 \( m \) 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 \( s(z) \in m[A/s'] \). 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 the 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 uniformly 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 \( m(A_i(x_1, \ldots, x_1))(s) \) and the appearance of \( s \) in the definition of \( s(x_1) \) will be sensible. For the remainder of this section we assume that we are dealing with such a state. Now we give the meaning of a proof expression with respect to a model and a state.

1. \( m(\text{andin}(a : T_1, b : T_2))(s) = \langle m(a)(s), m(b)(s) \rangle \)
2. \( m(\text{andell}(p : T))(s) = \text{first member of } m(p)(s) \)
3. \( m(\text{andelr}(p : T))(s) = \text{second member of } m(p)(s) \)
4. \( m(\text{orinl}(a : T))(s) = \text{inl}(m(a)(s)) \) into \( m(A\upharpoonright B)(s) \)
5. \( m(\text{orirr}(b : T))(s) = \text{inr}(m(b)(s)) \) into \( m(A\upharpoonright B)(s) \)
6. \( m(\text{orell}(d : T_1, f : T_2, g : T_3))(s) = m(f)(s)(a) \) where \( m(d)(s) = \text{inl}(a) \)
7. \( = m(g)(s)(b) \) where \( m(d)(s) = \text{inr}(b) \)
8. \( m(\text{somein}(a, b : B))(s) = \langle m(a)(s), m(b)(s) \rangle \)
9. \( m(\text{someel}(p : T_1, f : T_2))(s) = m(f)(s) \) (first member of \( m(p)(s) \)) (second member of \( m(p)(s) \))
10. \( m(\text{impin}(x : T_1, b : T_2))(s) = \lambda y : m(T_1)(s). m(b)(s) \) (second member of \( m(p)(s) \))
11. \( m(\text{impel}(f : T_1, a : T_2))(s) = m(f)(s)(m(a)(s)) \)
12. \( m(\text{allin}(x : b : T))(s) = \lambda y : D. m(b)(s[x := y]) \)
13. \( m(\text{allel}(f : T_1, a : T_2))(s) = m(f)(s)(m(a)(s)) \)
14. \( m(\text{magic}(A))(s) = \) an element of \( m(A)(s) + m[\neg A](s) \)

Notice that by the correctness conditions on a proof expression, \( \text{andell}(p) \), for example, is sensible only if \( p \) is a proof expression for a conjunction, \( A \& B \). We will prove below that \( p \) must denote an element of \( m(A \& B)(s) \). Thus \( p \) denotes a pair, say \( <a, b> \), and it makes sense to take its first member, denoted \( 1of(p) \), and its second member, denoted \( 2of(p) \).

The operations \( \text{inl}, \text{inr} \) are injections of a set into its disjunct in a disjoint union, i.e.

\[
\text{inl} : A \to (A + B) \\
\text{inr} : B \to (A + B)
\]

We know by correctness that \( \text{orinl} \) applies only if \( a \) is a proof expression for \( A \) and \( \text{orinl}(a) \) is one for \( A \upharpoonright B \). Similarly for \( \text{orirr} \). So the mappings make sense in clauses 4 and 5.

\[\text{Note, } \lambda y : A.b \text{ is the typed lambda calculus notation for a function from type } A \text{ whose body is the expression } b.\]
In \( \text{orel}(d:T_1, f:T_2, g:T_3) \) we know that \( d \) must be a proof expression for a disjunction, so \( T_1 \) is \( A \lor B \). Thus \( m(d)(s) \) will be a member of \( m(A \lor B)(s) \) as we show. Thus \( m(d)(s) \) is either \( \text{inl}(a) \) or \( \text{inr}(b) \) for \( a \) in \( m(A)(s) \) and \( b \) in \( m(B)(s) \).

The analysis of \( \text{some}(p:T_1, f:T_2) \) is just as for \( \text{andl} \). We know that \( m(p)(s) \) is a pair consisting of an element of \( D \) and evidence that the element is a witness for an existential quantifier.

In case of \( \text{magic}(A) \), 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 \( x_1:A_1, x_2:A_2, \ldots, x_n:A_n(x_1, \ldots, x_{n-1}) \) is a cascaded enumeration of the free variables of \( a \) and \( A \) with their bindings, and if \( m \) is any model and \( s \) any state assigning \( s(x_i) \) to \( m(A_i; x_i, \ldots, x_{i-1})(s) \), then \( m(a) \in m(A)(s) \).

**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 \( \text{magic}(C) \) for a formula \( C \). If \( a \) is a variable, then by hypothesis \( m(a)(s) = s(a) = s(x_i) \in m(A_i; x_i, \ldots, x_{i-1})(s) \). If \( a \) is \( \text{magic}(C) \), the \( A \) is \( C \lor \neg C \) and \( m(a)(s) \in m(C)(s) + m(\neg C)(s) \).

Now consider the induction case. We assume that the result holds for any subexpression of \( b \) of \( a \), in any state \( s \) assigning values to all free variables.

**induction hypothesis:**

\( \text{assume } m(b)(s) \in m(B)(s) \)

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 \( \text{andl}(b_1:B_1, b_2:B_2) \)

   Then \( A \) must be the conjunction \( B_1 \& B_2 \), and by the induction hypothesis \( m(b_i)(s) \in m(B_i)(s) \) \( i = 1, 2 \). But then the result holds by the definition \( m(\text{andl}(b_1:B_1, b_2:B_2)) = \langle m(b_1)(s), m(b_2)(s) \rangle \in m(B_1 \& B_2)(s) = m(A)(s) \).

2. \( a \) is \( \text{andl}(p:C) \)

   Then \( C \) must be \( A \& B \), and \( p \) must prove \( A \& B \), and we know \( m(p)(s) \in m(A \& B)(s) \). By the definition of evidence, the elements of \( m(A \& B)(s) \) are elements of \( m(A)(s) \times m(B)(s) \). So they are pairs. Thus \( m(\text{andl}(p:C)) = 1^{\text{off}} m(p)(s) \) and \( 1^{\text{off}} m(p)(s) \in m(A)(s) \).

3. \( a \) is \( \text{andlr}(p:C) \)

   This case is just like 2.
4. *a is orinl(b:T)*

Then T is B1C and a proves B1C. By the induction hypothesis we know m(b)(s) ∈ m[B](s). So inl(m(b)(s)) belongs to m[A](s) as required.

5. *a is orinr(b:T)*

This case is like 4.

6. *a is orel(b:C, f:T1, g:T2)*

Then we know that C is B1B2 and b proves B1B2 and f proves B1⇒A and g proves B2⇒A. By the induction hypothesis, m(b)(s) ∈ m[B1B2](s), m(f)(s) ∈ m[B1⇒A](s) and m(g)(s) ∈ m[B2⇒A](s). We know that elements of m[B1B2](s) are either inl(b1:B1) or inr(b2:B2). The cases are similar. Suppose m(b)(s) is inl(b1:B1), that means that b1 ∈ m[B1](s). Thus m(f)(s)(b1) belongs to m[A](s) as required.

7. *a is impin(z:C; b:B)*

Then A is C⇒B. Now consider any new state s′ where s′(z) ∈ m[C](s′). The variables are arranged in a cascade fashion so that this relationship makes sense. Now the induction hypothesis applies to b in this new state. So we know that m(b)(s′) ∈ m[B](s′). Since this relationship holds for all states s′, we can say for any value y of m[C](s) it holds, thus for all y in m(b)(s′(z:=y)). Thus

\[ \lambda y. m[C](s). m(b)(s(z:=y)) \in m[C](s)⇒m[B](s). \]

We can use m[B](s) in place of m[B](s′) because B does not have variables ranging over the evidence for formulas.

8. *a is impel(f:F, b:B)*

Then f proves B⇒A and b proves B. By the induction hypothesis m(b)(s) ∈ m[B](s) and m(f)(s)∈[B⇒A](s) which is the set of functions from m[B](s) into m[A](s). Thus m(f)(s)(m(b)(s)) belongs to m[A](s) as required.

9. *a is allin(x; b:B)*

Then A is ∀x.B. Consider now those states s′ such that s′(x) ∈ D. For these the induction hypothesis holds, so we know

m(b)(s′) ∈ m[B](s′).

This means that for each y in D we have

m(b)(s(x:=y)) ∈ m[B](s(x:=y))

Thus \[ \lambda y:D. m(b)(s(x:=y)) \] is a function of the proper type, in m[∀x.B](s).

10. *a is allel(f:F, b:B)*
Then A is $\exists x. B$. By the induction hypothesis we know that $m(b)(s) \in m[B](s[x := m(t)(s)])$ for any $s$. But then the definition $m[\exists x. B] = y:D[T] \times m[B](s[x := y])$ so in fact somein$(t, b; B)$ belongs to the evidence set as required.

11. $a$ is somein$(t, b; B)$

Then A is $\exists x. B$. By the induction hypothesis we know that $m(b)(s) \in m[B](s[x := m(t)(s)])$ for any $s$. But then by definition $m[\exists x. B] = y:D \times m[B](s[x := y])$ so in fact somein$(t, b; B)$ belongs to the evidence set as required.

12. $a$ is somei$(p; C, f; F)$

For this expression to make sense it must be that $p$ belongs to the evidence set for an existentially quantified formula, say $\exists x. B$. Since $p$ is a subexpression of somei$(p; C, f; F)$ know that $p$ actually denotes an element of such a set. Thus $p$ is a pair, say $\langle t, b \rangle$. We also know that $f$ denotes a member of $m[\forall x. (B \Rightarrow A)](s)$. Now according to the meaning function for proof expressions

$$m[\text{somei}(p; C, f; F)](s) = m(f)(s)(10f(m(p)(s))))(20f(m(p)(s)))),$$

and by the definition of $m(f)$, this is an element of $m(A)(s)$ as is required.

QED

Computational Semantics

With the exception of $\text{magic}(A)$, all proof expressions are given meaning in terms of recursively defined equations. For example,

$$m[\text{andin}(a:A, b:B)] = \langle m(a), m(b) \rangle.$$

The fact that the meaning of all proof expressions not involving $\text{magic}(A)$ is given by computable reductions means in particular that such expressions which denote functions, such as $\text{allin}(x; b; B)$ and $\text{impin}(x; a:A)$, denote in fact computable functions. In the case of number theory, for example, this means that if we prove a statement such as $\forall n. \exists y. y^2 \leq n < (y + 1)^2$ without using the law of excluded middle, then the proof expression for it, say $pf$, can be used to define a function mapping $\mathbb{N}$ to $\mathbb{N}$ which computes the value of $y$, i.e. which finds the integer square root of $n$. In particular the function value is

$$\text{sqrt}(x) = 10f(m(pf)(x)).$$

We do not need the state component because the formula and the proof expression are closed (have no free variables).

Constructive Logic

If the law of excluded middle, $P \lor \neg P$, 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 [6]. Arithmetic based on this logic and the Peano axioms is called Heyting arithmetic after the Intuitionist A.Heyting [14]. These topics are treated thoroughly in Kleene [15], Dummett [12] and Troelstra [23]. Analysis built on such a logic extended to higher order is sometimes called constructive analysis, see Bishop [3]. These topics are discussed in Troelstra [23] and Bridges [5].

Programming

The PRL programming systems built at Cornell in the early 1980's [2,18] are based on the idea that formal constructive logic, because of its computational semantics, provides a new kind of very high level programming language. This idea was first explored in Constable [8] and Bishop [4]. It was later developed by Bates and put into practice by Bates and Constable [2]. The semantics of evidence discussed here is quite close to the actual implementation ideas in Nuprl [17].

Acknowledgements

I want to thank Ryan Stansifer and Todd Knoblock for helpful comments and Donette Isenbarger, Elizabeth Maxwell, and Margaret Schimizzi for preparing the manuscript.

References


