Reprinted from

HANDBOOK OF PROOF THEORY

Edited by

SAMUEL R. BUSS

University of California, San Diego

1998

ELSEVIER

AMSTERDAM • LAUSANNE • NEW YORK • OXFORD • SHANNON • SINGAPORE • TOKYO

It is possible to extend the universe hierarchy further, say indexed by ordinal numbers ord. It is possible to postulate closure of $Type$ under various schemes for generating larger universes; Palmgren [1991] considers such matters.

Nuprl has been designed to facilitate index free, or "polymorphic", treatment of U_i. Generally, the user simply writes a universe as U_i and the system keeps track of providing relative level numbers among them in terms of expressions called *level expressions* which allow forming $i+1$ and $max(i,j)$. The theoretical basis for this is in Allen [1987b] and was implemented by Howe and Jackson (see Jackson [1994c]).

3.9. Semantics: PER models

The principal mathematical method that we have used to prove the soundness of Nuprl (and Martin-Löf type theory) has been to interpret equality relations on a type as partial equivalence relations ("pers") over terms — thereby building a variety of *term model* (see Stenlund [1972]). We use a method pioneered by Stuart Allen [1987a,1987b] to define the model inductively.[39] In his thesis Allen compares his models to those of Aczel [1986], Beeson, and Smith [1984]. The modeling techniques also borrow from Tait [1967,1983] in that the membership relation is extended from values to all terms by the pre-evaluation relation; in that regard it follows closely Martin-Löf's informal semantics.[40]

Allen's method has been remarkably potent in all of our work. Mendler [1988] used the technique to model the recursive types defined in Mendler, Constable and Panangaden [1986]), and Smith [1984] used it to model our bar types for partial objects Constable and Smith [1993]. Harper [1992] gave one of the most accessible accounts; I draw heavily on the accounts of Allen, Mendler, and Harper to explain the method.

The first step is to fix the collection of terms. The next step is to equip the terms with an evaluation relation, written now as $t \downarrow t'$. Allen [1987b] gives an abstract account of the syntax of terms and the properties of evaluation. We follow Mendler and Harper in supplying less detail.

Assume that on *closed* terms t evaluation satisfies E1 and E2:

E1. if $t \downarrow t'$ and $t \downarrow t''$ then $t' = t''$, so $\downarrow$ is *deterministic*, and

E2. if $t \downarrow t'$ then $t' \downarrow t'$, so $\downarrow$ is *idempotent.*

If $t \downarrow t'$ then we call t' a (*canonical*) *value.*

Our task now is to specify those terms which are intended to be expressions for mathematical objects and to specify those terms which are expressions for types. We carry this out for the types built from $\mathbf{N}$ using products and dependent functions. We *distinguish* these as two tasks. The first one is to consider *membership*, and the

[39] In his introduction Allen says "The principal content of this thesis is a careful development of ... a semantic reinterpretation [of type theory] with the intention of making the bulk of type-theoretic practice ... independent of its original type-theoretic and constructive basis. ... Moreover, in the unfamiliar domain of intuitionistic type theory, the reinterpretation can serve as a *staff made of familiar mathematical material.*"

[40] We say that if $t \downarrow t'$ and $t' \in T$, then $t \in T$. This is the *preevaluation* relation of t' to t.

second is to determine *type expressions.* We look at membership first since it is more basic.

According to Martin-Löf, to specify a type is to say what its members are, i.e., which terms are members, and to say what equality means on those terms which are members. Equality will be an equivalence relation, E, on some collection of terms. Considered over the entire collection, the relation need only be partial. The *field* of the relation (those elements in the relation to themselves, xEx) are the members of the type. These relations are called *partial equivalence relations* or per for short.

The built-in notion of computation places an additional requirement on the pers, namely, they must respect evaluation. That is, if $t \downarrow t'$ and tEr, then $t'Er$. We can say this succinctly by defining *Kleene equality* on terms, $t \simeq t'$ means that if $t \downarrow s$ or $t' \downarrow s$, then $t \downarrow s$ and $t' \downarrow s$, i. e. if either term has a value, then both have that same value. So we require that $t \simeq t'$ and tEr implies $t'Er$. We next introduce notation for this notion of type, starting with the idea that types themselves are mathematical objects with an equality defined on them.

Let us see how this notion of type membership looks for the natural numbers, i. e. for the type $\mathbf{N}$. Define the relation *Neq* on terms inductively by

0 *Neq* 0
a *Neq* b implies $suc(a)$ *Neq* $suc(b)$
a' *Neq* b' and $a \downarrow a'$ and $b \downarrow b'$ implies a *Neq* b.

Neq is a partial equivalence relation which determines a minimal notation for numbers on which we can compute by primitive recursion (lazily). That is, we know what elements are zero, and for nonzero numbers, we can find the predecessor.

Next, we define a membership per for the Cartesian product of two types A and B with α and β as the membership relations. Let α^* denote the pre-evaluation relation of a relation α, that is $a\alpha b$ iff there are a', b' such that $a'\alpha b'$ and $a \downarrow a', b \downarrow b'$. Define $\alpha \otimes \beta$ as $\{\langle pair(a;b), pair(a';b')\rangle \mid a\alpha a' \ \&\ b\beta b'\}^*$.

We can see that if α and β are value respecting pers, then so is $\alpha \otimes \beta$. It clearly defines membership in a Cartesian product according to our account of products.

Finally, we need a membership condition for the dependent function space constructor, $fun(A; x.\ B)$. This is a bit more complex because for each element a of A, $B[a/x]$ is a type. So we need to consider a family of membership relations indexed by a type. The members of the function type will be lambda terms, $\lambda(x.\ b)$. Let α be a value respecting per and for each a such that $a\alpha a$, let $\Phi(a)$ be a value respecting per. Define $\Pi\alpha\Phi$ as the following partial equivalence relation:

$$\{\langle \lambda(x.\ b), \lambda(x'.\ b')\rangle \mid \forall a, a'.\ a\alpha a' \Rightarrow b[a/x]\ \Phi(a)\ b'[a'/x']\}^*.$$

In order for this per to define type membership for the function space, we require that whenever $a\alpha a'$, then $\Phi(a) = \Phi(a')$. We have in mind that these membership conditions are put together inductively. This is made explicit by the following inductive definition of a relation $\boldsymbol{K}$ on pers.

$\boldsymbol{K}(Neq)$
$\boldsymbol{K}(\alpha \otimes \beta)$ if $\boldsymbol{K}(\alpha)$ and $\boldsymbol{K}(\beta)$
$\boldsymbol{K}(\Pi\alpha\Phi)$ if $\boldsymbol{K}(\alpha)$ and $\forall a, a'.\ a\alpha a' \Rightarrow \Phi(a) = \Phi(a')$ & $\boldsymbol{K}(\Phi(a))$.

We can prove inductively that all the pers in $\boldsymbol{K}$ are value respecting and all define type membership. $\boldsymbol{K}$ provides a per semantics for the small type theory based on $\mathbf{N}$, products and dependent functions. Notice equality on pers is *extensional.*

Type expressions. The inductively defined set $\boldsymbol{K}$ determines a collection of membership pers which represent types, but it does not relate these to the terms used to name types, e. g. terms such as $N, N \times N$, $fun\,(N; x.\ decide(s; u.\ N; v.\ N \times N))$ and so forth. We establish this relationship next by modifying the definition of $\boldsymbol{K}$ to include names for types. Let $\boldsymbol{M}$ be the following inductively defined binary relation.

N	$\boldsymbol{M}$	*Neq*
$A \times B$	$\boldsymbol{M}$	$\alpha \otimes \beta$ if $A\boldsymbol{M}\alpha$ and $B\boldsymbol{M}\beta$
$fun(A; x.\ B)$	$\boldsymbol{M}$	$\Pi\alpha\Phi$ if $A\boldsymbol{M}\alpha$ and $\forall a, a'.\ a\alpha a' \Rightarrow \Phi(a) = \Phi(a')$ and $B[a/x]\boldsymbol{M}\Phi(a)$

This is an ordinary inductive definition of a binary relation. Also, it is easy to see that $A\boldsymbol{M}\alpha$ implies $\boldsymbol{K}\alpha$. The only membership pers described by $\boldsymbol{M}$ are those whose constituents are also described by $\boldsymbol{M}$. Moreover, all the membership pers are represented by terms, i. e. are related to terms by $\boldsymbol{M}$. This is critical for the $\Pi\alpha\Phi$ pers because it guarantees that Φ is represented by a term. Here are three critical facts about $\boldsymbol{M}$.

Fact 1 $A\boldsymbol{M}\alpha \Rightarrow \boldsymbol{K}(\alpha)$
Fact 2 $A\boldsymbol{M}\alpha$ and $A\boldsymbol{M}\alpha' \Rightarrow \alpha = \alpha'$
Fact 3 $A\boldsymbol{M}\alpha$ and $A \simeq A' \Rightarrow A'\boldsymbol{M}\alpha$.

These facts can be proved by $\boldsymbol{M}$ induction. Fact 1 means that all member pers are value respecting, and Fact 3 means that the type names are value respecting as well.

Pers for intensional type equality. We now want to define a per on type expressions which represents type equality and is value respecting. There is already a sensible equality that arises from $\boldsymbol{M}$, namely, $A = A'$ if $A\boldsymbol{M}\alpha, A'\boldsymbol{M}\alpha'$ and $\alpha = \alpha'$. This is an extensional equality. We want to model the structural equality of section 3.7, thus $A \times B = A' \times B'$ iff $A = A'$ and $B = B'$. Here is the appropriate definition of a binary relation $\boldsymbol{E}$ on terms.

$N\boldsymbol{E}N$
$A \times B\boldsymbol{E}A' \times B'$ if $A\boldsymbol{E}A'$ and $B\boldsymbol{E}B'$
$fun(A; xB)\boldsymbol{E}\ fun(A'; x'.\ B')$ if $A\boldsymbol{E}A'$ and $\exists\alpha\ A\boldsymbol{M}\alpha$ and $A'\boldsymbol{M}\alpha$ and $\forall a, a'.\ a\alpha a' \Rightarrow B[a/x]\boldsymbol{E}\ B'[a'/x']$
$A\boldsymbol{E}A'$ if $B\boldsymbol{E}B'$ and $A \downarrow B$ and $A' \downarrow B$.

We say that A is an intensional type expressions of model $\boldsymbol{M}$ iff $\exists\alpha(A\boldsymbol{M}\alpha$ and $A\boldsymbol{E}A)$. Clearly, the per representing type equality is value respecting. The relations $\boldsymbol{E}$ and $\boldsymbol{M}$ provide a model of our type theory fragment. The methods extend to Martin-Löf's '82 theory and to Nuprl 3 as Allen has shown.

We now summarize the approach described above, starting from $\boldsymbol{E}$ and following Harper's method of using least fixed points to present the inductive relations.

Summary of per semantics. Here is a summary of the per semantics along the lines developed by Harper [1992]. Let $\mathcal{T}$ be the collection of terms. We define on $\mathcal{T}$ a *partial equivalence relation* E intended to denote type equality. If aEa then we say that a is a type. If aEa' then a and a' are equal types. Let $|E| = \{t : \mathcal{T} \mid tEt\}$, called the *field* of E. Let $\mathcal{T}/E$ be the set of equivalence classes of terms; say $[t]_E = \{x : \mathcal{T} \mid xEt\}$. Let PER denote the set of all partial equivalence relations on $\mathcal{T}$.

Associated with each type is a membership equality, corresponding to $a = b$ in A; thus for each $a \in |E|$, there is a partial equivalence relation, $\boldsymbol{L}(a)$. So $\boldsymbol{L} \in \mathcal{T}/E \to$ PER.

We require of E and each $\boldsymbol{L}(a)$ that they respect evaluation, i.e. if $a_1 E a_2$ and $a_1 \downarrow a_1'$ and $a_2 \downarrow a_2'$, then $a_1' E a_2'$. Likewise for $\boldsymbol{L}(a)$ in place of E.

Now consider how we might define E and $\boldsymbol{L}$ mutually recursively to build a model of the type theory. For the sake of simplicity, we start with an extensional notion of type equality, as above. Call it Ext. We define Ext and $\boldsymbol{L}$ mutually recursively.

$$a\ Ext\ b \quad \text{iff} \quad \forall x, y.\ ((x\boldsymbol{L}(a)y) \Leftrightarrow (x\boldsymbol{L}(b)y))$$

$$s\boldsymbol{L}(a_1 \times a_2)t \quad \text{iff} \quad \exists s_1, s_2, t_1, t_2.\ s \downarrow pair(s_1; s_2)\ \&\ t \downarrow pair(t_1; t_2)\ \&\ s_1\boldsymbol{L}(a_1)t_1\ \&\ s_2\boldsymbol{L}(a_2)t_2.$$

$$f\boldsymbol{L}(fun(a_1; x.\ a_2))f' \quad \text{iff} \quad \exists x, b, x', b'.\ f \downarrow \lambda(x.\ b)\ \&\ f' \downarrow \lambda(x'.\ b')\ \&\ \forall y, y' : |Ext(a_1)|.\ (y\boldsymbol{L}(a_1)y' \Rightarrow b[y/x]\boldsymbol{L}(a_2[y/x])b'[y'/x']).$$

This is a mutually recursive definition of Ext and $\boldsymbol{L}$, and it reflects our intuitive understanding, but the definition is not a standard positive (hence monotone) inductive definition because of the negative occurrence of $y\boldsymbol{L}(a_1)y'$ in the clause defining equal functions.

Allen calls these "half-positive" definitions; his method of using $\boldsymbol{K}$ and $\boldsymbol{M}$ as above shows how to replace this nonstandard definition with a standard positive induction which can be interpreted in either classical or constructive settings (for example, in ZF or IZF or in a theory of inductive definitions, see Troelstra [1973] and Feferman [1970]).

Definition. A *type system* τ is a pair $\langle E, \boldsymbol{L}\rangle$ where E is a value respecting per on $\mathcal{T}$ and for each $a \in |E|$, $\boldsymbol{L}(a)$ is a value respecting per. Given type systems $\tau = \langle E, \boldsymbol{L}\rangle$ and $\tau' = \langle E', \boldsymbol{L}'\rangle$, define $\tau \sqsubseteq \tau'$ iff $E \sqsubseteq E'$ and $\forall a : |E|.\ \boldsymbol{L}(a) = \boldsymbol{L}'(a)$; that is $\tau \sqsubseteq \tau'$ iff τ' has possibly more types, and on the types in τ it has the same equality.

Let TS be the collection of all type systems over $\mathcal{T}$ with evaluation $\downarrow$. It is easy to see that TS under $\sqsubseteq$ is a complete partially ordered set, a *cpo*. The relation

$\tau \sqsubseteq \tau'$ is a partial order on TS, and there is a least type system in this ordering, namely $\langle\phi, \phi\rangle$ where ϕ is the empty set. A non-empty subset D of TS is *directed* iff every pair of elements in D, say τ, τ', has an upper bound in D. Given any directed set D of type systems, say τ_i for $i \in I$, it has a least upper bound $\hat{\tau}$ where $\hat{E} = \cup E_i$ and $\boldsymbol{L}_\omega(a) = \boldsymbol{L}_i(a)$ if any $\boldsymbol{L}_i(a)$ is defined. If $\boldsymbol{L}_i(a)$ is defined, then since τ_i, τ_j for $i \neq j$ has an upper bound in D, say τ_k, we know that $\boldsymbol{L}_j(a) = \boldsymbol{L}_i(a)$ for $a \in E_i \cup E_j$, so the type system τ_a is well defined. Also τ_ω is least since if $\tau_i \sqsubseteq \tau'$ for all i, then $E_i \sqsubseteq E'$ for all i, so $\cup E_i \sqsubseteq E'$.

Theorem. *For any cpo D with order $\sqsubseteq$, if $F \in D \to D$ and F is monotone, i.e. $x \sqsubseteq y \Rightarrow F(x) \sqsubseteq F(y)$ for all x, y in D, then there exists a least fixed point of F in D, i.e. an element x_0 such that (i) $F(x_0) = x_0$, (ii) for all z such that $F(z) = z$ and $x_0 \sqsubseteq z$.*

We now define an operation $T \in TS \to TS$ which is monotone and whose least fixed point, $\hat{\tau}$, is a type system which models our rules.

Definition. Let $T(\langle E, \boldsymbol{L}\rangle) = \langle F^*, \boldsymbol{M}\rangle$ where

$$\begin{aligned} F = \{&\langle N, N\rangle\} \\ &\cup\{\langle a \times b, a' \times b'\rangle \mid aEa'\ \&\ bEb'\} \\ &\cup\{\langle fun(a; x.\ b), fun(a'; x'.\ b')\rangle \mid aEa'\ \& \\ &\qquad \forall y, y'.\ y\boldsymbol{L}(a)y' \Rightarrow b[y/x]Eb'[y'/x']\} \end{aligned}$$

$$\boldsymbol{M}(a) = \begin{cases} Neq \text{ if } a = N \\ \boldsymbol{L}(a_1) \otimes \boldsymbol{L}(a_2) \text{ if } a = a_1 \times a_2 \\ \Pi(\boldsymbol{L}(a_1), \lambda(x.\ \boldsymbol{L}(b))) \text{ if } a = fun(a_1; x.\ b)\ \& \\ a_1 \in |E|\ \&\ \forall y : |\boldsymbol{L}(a_1)|.\ b[y/x] \in |E| \end{cases}$$

Theorem. *T is monotone in $\sqsubseteq$ on TS.*

3.10. Using type systems to model type theories

Allen's techniques enable us to model a variety of type theories. Let us designate some models for the theories discussed earlier. We'll fix the terms and evaluation relation to include those of the richest theory; so the terms are: $\mathbf{0}$, $\mathbf{1}$, $\bullet$, $\mathbf{N}$, 0, $suc(t)$, $prd(s)$, $add(s; t)$, $mult(s; t)$, $exp(s; t)$, $R(n; t; v.b; u, v, i.h)$, $s \times t$, $pair(s; t)$, $prod(s; x.t)$, $fun(s; x.t)$, $\lambda(x.t)$, $list(t)$, $(s.t)$, $L(s; a; v.b; h, t, v, i.g)$, $s + t$, $inl(s)$, $inr(t)$, and $decide(p; u.s; v.t)$.

There is also the evaluation relation $s\ evals\_to\ t$ which we abbreviate as $s \downarrow t$. We consider various mappings $\mathbf{T}_l : TS \to TS$ where l is a label such as N, G, ML, Nu, etc. The most elementary "theory" we will examine is a subtheory of arithmetic involving only equalities over $\mathbf{N}$ built from $0, suc(t), prd(s)$, and $add(s; t)$. This is modeled by $\mathbf{T}_N$. The input to $\mathbf{T}_N$ is any pair $\langle E, \boldsymbol{L}\rangle$ and the output is $\langle F, \boldsymbol{M}\rangle$ where the only type name is N, so NFN, thus $N \in |F|$, and the only type equality

is $\boldsymbol{M}(N)$ which is defined inductively. We have that $s\boldsymbol{M}(N)t$ is the least relation *Neq* such that

$$s\ Neq\ t\ \textit{iff}\ (s\downarrow 0\ \&\ t\downarrow 0\ v\ \exists s',t'.\ s\downarrow suc(s')\ \&.\ t\downarrow suc(t')s'Neq\ t')\,.$$

The map T_N takes any $\langle E, \boldsymbol{L}\rangle$ to this $\langle F, \boldsymbol{M}\rangle$, so its least fixed point, $\mu(\boldsymbol{T}_N)$ is just $\langle F, \boldsymbol{M}\rangle$. This is the model for *successor arithmetic.* We see that in this model, N is a type, that $s = t$ in N iff s evaluates to a canonical natural number and t evaluates to the same canonical number.

The rules can be confirmed as follows. First, notice that evaluation is deterministic and idempotent on the terms. As we observed, the general equality rules hold in any type system (because $\boldsymbol{M}(N)$ is in equivalence relation on canonical numbers). This follows by showing inductively that $0, suc(0), suc(suc(0)), \ldots$ are in the relation $\boldsymbol{M}(N)$, i.e. in the field of the relation. The fact that $\boldsymbol{M}(N)$ respects evaluation validates the last equality rule.

$$\mu(T_N) \models 0 = 0 \text{ in } N$$
$$\mu(T_N) \models s = t \text{ in } N \text{ implies } \mu(T_N) \models suc(s) = suc(t) \text{ in } N.$$

The typing rule for successor is also confirmed by induction on *Neq*; namely, if $s'\boldsymbol{L}(N)t'$, then since $suc(s') \downarrow suc(s')$ and $suc(t') \downarrow suc(t')$, then we have $suc(s)\boldsymbol{M}(N)suc(t)$ as required for the typing rule.

In the case of N, the model $\mu(T_N)$, and the informal semantics are essentially the same. So the theory fragment for N can *stand on its own* with respect to the model. Even a set theoretic semantics for N will have the same essential ingredient of an inductive characterization. For instance, Frege's definition was that

$$N == \{x \mid \forall X.\ (0 \in X\ \&\ \forall y.\ (y \in X \Rightarrow s(y) \in X)) \Rightarrow x \in X\}$$

where $s(x)$ is $\{z \mid \exists u.\ (u \in z\ \&\ z - \{u\} = x)\}$. In ZF we can use the postulated infinite set, inf, and form $\omega == \{i : inf \mid \forall x.\ nat\_like(x) \Rightarrow i \in x\}$ where $nat\_like(x)$ *iff* $(0 \in x\ \&\ \forall y.\ (y \in x \Rightarrow suc(y) \in x))$ for $Suc(y) = y \cup \{y\}$. In both of these definitions, the inductive nature of $\mathbf{N}$ is expressed. But Frege's theory and ZF allow very general ways of using this inductive character. So far we have only used it for specifying the canonical values.

The same approach can be used to define a model for the type theory with cartesian products. In this case we denote the operator on type systems as ${T_N}^2$. Given ${T_N}^2(\langle E, \boldsymbol{L}\rangle) = \langle F, \boldsymbol{M}\rangle$, if $S \in |E|$ and $T \in |E|$ then $S \times T \in |F|$, and $\boldsymbol{M}(S \times T)$ is $\boldsymbol{L}(S) \otimes \boldsymbol{L}(T)$. For this system, ${T_N}^2$ is continuous, i. e. if $\tau_0 = \langle \phi, \phi\rangle$ and ${T_N}^2(\tau_i) = \tau_{i+1}$, then $\mu({T_N}^2) = \tau_\omega$.

In $\mu({T_N}^2)$ all the rules for the fragment of section 3.2 are true. Again the theory is so close to the semantics that it stands on its own. Notice that in confirming the rule for typing pairs, we rely on the fact that $\mu({T_N}^2)$ is a fixed point.

$$\mu({T_N}^2) \models s = s' \text{ in } S \text{ and } \mu({T_N}^2) \models (t = t' \text{ in } T) \text{ imply}$$
$$\mu({T_N}^2) \models pair(s;t) = pair(s';t') \text{ in } S \times T.$$

Note, this fact would not be true in any fixed τ_i since $S \times T$ might be defined only in τ_{i+1}.

To provide a semantics for $fun(A; x.\ B)$ and $prod(A; x.\ B)$ we use the map T_{ML} defined in section 3.9. The model is $\mu(T_{ML})$. To prove the rules correct, we recall the meaning of sequents such as $x \in A \vdash B$ type and $x \in A \vdash s = t$ in T.

$$\mu(T_{ML}) \models (x \in A \vdash B\ type) \text{ implies } \mu(T_{ML}) \models fun(A; x.B)\ type$$
$$\mu(T_{ML}) \models (x \in A \vdash b \in B) \text{ implies } \mu(T_{ML}) \models \lambda(x.b) \text{ in } fun(A; x.B)$$

Modeling hypothetical judgments. The meaning of $x \in A \vdash b \in B$ is that A is a type and for any two elements, a, a' of A, $B[a/x]$ is a type and $B[a/x] = B[a'/x]$ (i.e. B is *type functional* in A), and moreover, $b[a/x] \in B[a/x]$ and $b[a/x] = b[a'/x]$ in $B[a/x]$. We have extended this notion to multiple hypotheses inductively to define $x_1 \in A_1, \ldots, x_n \in A_n \vdash b \in B$. This definition can be carried over to type systems.

3.11. A semantics of proofs

The discussion of proofs as objects and Heyting semantics in section 2 suggested treating proofs as objects and propositions as the types they inhabit. True propositions are those inhabited by proofs. But there were several questions left open in section 2.14 about the details of carrying out this idea.

The type theory of this section can answer these questions, and in so doing it provides a *semantics of proofs.* The basic idea is to consider a proposition as the type of all of its proofs and to take proof expressions to denote objects of these types. Based on Heyting's semantics we have a good idea of how to assign a type to compound propositions in terms of types assigned to the components. For atomic propositions there are several possibilities, but the simple one will turn out to provide a good semantics. The idea is to consider only those atomic propositions which can plausibly have *atomic proofs* and to denote the canonical atomic proofs by the term *axiom.* We will assign types to the compound propositions in such a way that the canonical elements will represent what we will call *canonical proofs.* Moreover, the reduction relation on the objects assigned to proof expressions will correspond to meaningful reductions on proofs. Proofs corresponding to noncanonical objects will be called noncanonical proofs. The correspondence will guarantee that noncanonical proofs p' of a proposition P will reduce to canonical proofs of P.

We now define the correspondence between propositions and types and between proofs and objects. Sometimes this correspondence is called the Curry-Howard isomorphism.

Curry-Howard isomorphism. For the sake of this definition, if P is a proposition, we let $[\![P]\!]$ be the corresponding type, and if p is a proof expression, we let $[\![p]\!]$ be the corresponding element of $[\![P]\!]$. We proceed to define $[\![\ ]\!]$ inductively on the structure of proposition P from section 2.5.

1. We consider only atomic propositions of the form $a = b$ in A. The type $[\![a = b \text{ in } A]\!]$ will have the atomic proof object *axiom* if the proposition is