Dependent Types

Dependent Product

$x:A \times B(x)$ $\quad$ <a,b> with a in A, b in B(a)

Dependent Records

$\{x_1:A_1;\ x_2:A_1(x_1);x_3:A(x_1,x_2);\ldots;x_n:A_n(x_1,\ldots,x_{n-1})\}$

functions r from Labels $\{x_1,\ldots,x_n\}$ to values
where $r(x_1) \varepsilon\ A_1$, $r(x_2) \varepsilon A_2(r(x_1))$, etc.

Dependent Functions (special case of dep records)

$x:A_1 \rightarrow A_2(x)$ $\qquad\qquad$ λ(x.exp) functions

Propositions as Types

The dependent type constructors and the standard types are sufficient to give a semantics for logic different from truth semantics.

The mechanism for doing this goes by many names. One is propositions as types. In this approach, proofs of a proposition are considered elements of the type corresponding to the propositions.

Proofs as Terms

Another way of looking at this principle that N.G. de Bruijn, one of its proponents and inventors, favored was to introduce proofs as terms in the logic, rather than in the metalogic, and treat them as elements of the types that correspond to the propositions.

Semantics of Evidence

My way of looking at these “PAT” principles is as an abstract semantics of evidence. Like de Bruijn’s approach, this one also applies to classical logics as well as constructive logics.

For the constructive logics, the evidence is always computable in the Church/Turing sense or in the Brouwer sense. Classically, we need oracles and a special proof term – I like to call it magic.

Evidence Semantics is Meaningful for Classical and Constructive Logics

For constructive (and intuitionistic) logics this evidence semantics is essentially the Brouwer, Heyting, Kolmogorov semantics (BHK) from 1907 to 1935.

We now know it is meaningful for classical logics as well if we introduce an object that can be used as evidence for the law of excluded middle,

(P v ¬P). I call that element magic(P).

Importance of Propositions as Types

If we adopt an evidence semantics, then proof terms denote evidence. They can be incorporated into the logic as terms, and if the logic is constructive, these terms are part of a computation system, so they can be evaluated.

This principle shows that in a sense, the dependent type system is a first limit or “fixed point” for programming languages and logics – classical or constructive.

Proofs as Programs

For a constructive logic, all proof terms can be considered as programs or data that reveal the implicit computational content and make it explicit.

The proof assistants based on computational type theory or other constructive theories can evaluate these proof terms and thus extract the computational content. For classical logic only some proof terms are computable.

Relationship to Truth Semantics

We can relate the semantics provided by treating propositions as types to the usual truth semantics made precise by Tarski.

Standard Tarski Semantics

The standard semantics for a logical formula *A* is that A is true about a model $\mathcal{M}$ if the truth value of *A,* ***tr****(A) = true* in $\mathcal{M}$. This is called the Tarski semantics.

For example, ***tr****(A&B) = true* exactly when

tr*(A) = true* and ***tr****(B) = true*

Also ***tr****(A → B)* is **tr**(¬*A* v *B*) as well as

tr(*All x.B(x)*) is tr(*B(a)*) all *a* in $\mathcal{M}$.

Semantics of Evidence

The evidence semantics for a formula *A* in a model $\mathscr{M}$ is the set of objects a that constitute evidence for *A*. These objects use elements of the universe $U_{\mathscr{M}}$ along with other elements.

$$[A] = \{a \mid a \text{ is evidence for } A \text{ in } \mathscr{M}\}$$

For example **[**0=0**]** is “axiomatic” say, {*true*}

Evidence Semantics

The evidence for 0 < 1 could be "axiomatic" or if we define x < y to mean x+z = y for

z > 0 then the evidence for 0 < 1 is

$[0<1]_{\mathcal{M}}$ = {<z,w>} where z>0 and w in $[x+z=y]_{\mathcal{M}}$

If A is false, then $[A]_{\mathcal{M}}$ is empty.

Propositional Evidence

Suppose that we have evidence types **[**A**]** for the atomic propositions A. Here is how to construct evidence for compound formulas in a model $\mathscr{M}$.

[A & B**]** == **[**A**]** x **[**B**]**

[A v B**]** == **[**A**]** + **[**B**]**

[A => B**]** == **[**A**]** → **[**B**]**

[false**]** == ϕ (the empty set)

[¬A**]** == [A] → ϕ

Evidence for Quantified Propositions

The evidence types for quantified formulas use the dependent types over the universe $U_{\mathcal{M}}$ of the model $\mathcal{M}$:

$$[\text{All } x.\ B(x)] \quad == \quad x{:}\ U_{\mathcal{M}} \rightarrow [B(x)]$$

$$[\text{Exists } y.\ B(y)] == \quad y{:}\ U_{\mathcal{M}} \times [B(y)]$$

Relationship to Tarski Semantics

It is easy to relate evidence semantics to Tarski's semantics for first-order logic and show that a formula A is true in a model $\mathcal{M}$ iff and only if there is evidence for A. Moreover, if pf is a proof of A, then we can define an evidence semantics for pf, say [pf], such that:

$$pf \mid\!- A \quad \text{iff} \quad [pf]_{\mathcal{M}} \in [A]_{\mathcal{M}}$$

$$\mid\!=_{\mathcal{M}} A \quad \text{iff} \quad a \in [A]_{\mathcal{M}}$$

The Curry Howard Isomorphism

Many computer scientists like to call the “PAT” semantics the Curry-Howard isomorphism, even though it is not an isomorphism nor was it invented by either Curry or Howard.

The book on this topic, Lectures on the Curry-Howard Isomorphism by Sørensen and Urzyczyn, lists 23 contributors to the principle, the main ones being Brouwer, Heyting, Kolmogorov, Kleene, deBruijn, Curry, Howard, Girard, and Martin-Löf, starting by 1907.