2 Constructive Type Theory Background

2.1 Implemented Type Theories

Four of the proof assistants mentioned above, Agda [12], Coq [10], Nuprl [2], and MetaPRL [28] implement versions of constructive type theories and generalize work of logicians on intuitionistic type theory [33, 34] and constructive set theory [1, 23]. From the beginning Nuprl implemented Constructive Type Theory (CTT) [18], a significant enrichment of Martin-Löf's *extensional type theory* from 1982 [33], call it ML82. CTT provides a precise foundational semantics [3] that justified the addition of several new types to constructive type theory, including recursive types, quotient types, refinement types, and *partial types* (in which elements can diverge, as in programming language types).

One of the most powerful type constructors added to Nuprl after 2000 is the *intersection type* and the intersection of a family of types $\bigcap_{i:A} B(i)$. These are the elements that belong to all of the types $B(i)$. This constructor becomes a *uniform universal quantifier* in logic and is the key to defining a powerful new concept of *uniform validity* used to give a new *completeness theorem* for first-order logic [17]. Applications show this to be one of the most useful types for specification. This type is used to formalize the Smullyan result on Boolean evaluation. The other new types such as *dependent intersection* [30], *objects* [31], and *union* are also interesting, but we do not have occasion to expand on them in this article.

The *partial types* [19] allow Nuprl to write *partial recursive functions* over any data type, thus making it a universal programming language, unlike ML82, Agda and Coq which only support total types and total functions on them. CTT begins with total types T and defines the corresponding partial types $\overline{T}$ as including possibly diverging terms with the property that if they converge, they belong to the type T. Every partial type is inhabited by any term that is known to diverge, such as $fix(\lambda(x.x))$. We give this term the short name $\bot$. We can take fix to be the $\mathbf{Y}$ combinator,

$$\mathbf{Y} = \lambda f(\lambda x(f(xx)))(\lambda x(f(xx))).$$

It has the typing rule needed here, and we use it later in the article as well.

Like the $\mathbf{Y}$ combinator, the fix operator is used to define recursive functions. For example the recursive multiplication function

$$mult(x, y) = if\ y = 0\ then\ 0\ else\ mult(x, y - 1) + x$$

is written as

$$fix(\lambda(mult.\lambda(x, y.if\ y = 0\ then\ 0\ else\ mult(x, y - 1) + x))).$$

To evaluate the above term, we substitute the entire fix expression for *mult* in the definition. This is a very clear definition of recursion in the style of the **Y** combinator.

The simplest form of the typing rule we require later is this. If f has type $\overline{T} \rightarrow \overline{T}$, then $fix(f)$ has type $\overline{T}$. A common example is a functional F of type

$$(\overline{T} \rightarrow \overline{T}) \rightarrow (\overline{T} \rightarrow \overline{T}).$$

The least fixed point of this functional is a partial function in the type $(\overline{T} \rightarrow \overline{T})$. It is given by the fix operator. In constructive type theory the functionals and functions are effectively computable.

3 Validating Hoare Logic using Partial Types

We start this section by looking briefly at how partial types can naturally express many basic ideas of computability theory [19]. For instance, the halting problem can be stated in a particularly clear way and it's unsolvability given an elegant short proof, the kind Smullyan produced in so many of his writings. The type $\mathbb{B}$ is the type of Booleans.

Unsolvability results using partial types Here is a statement of the unsolvability of the halting problem in terms of partial types and a proof of the result.

Given any non-empty type T, there is no function $h : \overline{T} \rightarrow \mathbb{B}$ such that $\forall x : \overline{T}.(h(x) = tt$ iff $x \downarrow)$, e.g. no internal computable h solves the halting problem on $\overline{T}$.

Consider the special case of $T = N$ for concreteness; the proof works verbatim for any non-empty type T. Note that in constructive type theory, all functions are effectively computable. So when we say that no halting detector exists, we are saying that no internally computable function can solve the halting problem. Since the Nuprl type theory includes all partial recursive functions and is thus universal, the result we prove is a faithful version of the halting problem as it is widely known.

Theorem $\neg\exists h : \overline{\mathbb{N}} \rightarrow \mathbb{N}.\forall x : \overline{\mathbb{N}}.(h(x) = 0$ iff $x \downarrow)$

Proof

Suppose such an $h : \overline{\mathbb{N}} \rightarrow \mathbb{N}$ could be constructed in this theory, then consider the function $\lambda(x.zero(h(x); \bot; 0)$, where the *zero* function tests its first argument for having the value zero and returns $\bot$ in that case and 0 otherwise. Its type is $\overline{\mathbb{N}} \rightarrow \overline{\mathbb{N}}$. So $fix(\lambda x.zero(h(x); \bot; 0))$ has type $\overline{\mathbb{N}}$. Let d be this value in $\overline{\mathbb{N}}$.