Reflective Semantics of Constructive Type Theory (Preliminary Report)

Scott F. Smith
The Johns Hopkins University*

August 30, 1991

Abstract

It is well-known that the proof theory of many sufficiently powerful logics may be represented internally by Gödelization. Here we show that for Constructive Type Theory, it is furthermore possible to define a semantics of the types in the type theory itself, and that this procedure results in new reasoning principles for type theory. Paradoxes are avoided by stratifying the definition in layers.

1 Introduction

Given a sufficiently powerful logical theory L such as Peano Arithmethic, it is well-known that the proof theory of L may be expressed internally via Gödelization. This is accomplished by defining a metafunction $\lceil A \rceil$ that encodes formulas A as data, and a predicate $Provable_L(\lceil A \rceil)$ which is true just when formula A is provable. This gives L knowledge of its own proof theory, but it doesn't know that it knows it: the embedded proof theory could just as well be for some different logic. What is needed then are principles of self-knowledge that connect the provability predicate with the actual proof theory. Feferman [Fef62] for one has studied adding such principles; the principle most relevant to this work is

> Given a theory L_k, extend it to give a theory L_{k+1} with added reflection axiom $\vdash_{L_{k+1}} Provable_{L_k}(\lceil A \rceil) \Rightarrow A$.

In other words, if we prove that we can prove it, we can prove it. Note that this is not a true self-knowledge principle, because L_k does not have self-knowledge of the reflection axiom. Given a theory L_1, this principle induces a hierarchy of theories $L_1, L_2, L_3, \ldots$, which we call the reflected proof hierarchy.

Logicians have studied this hierarchy to understand its proof-theoreric strength; here we are not directly interested in this issue. We are interested in how computer scientists have found it applicable for automated theorem proving systems. Suppose there was an assumption A from which we wished to prove B. If the theorem-proving system knew of a way of always proving B from A, not by having assumption $A \Rightarrow B$ but by observing some syntactic property of A, the system would still have to construct each step of the proof. Given a reflected proof system, this construction may be avoided. Suppose we were to prove the theorem

$$\forall \lceil A \rceil, \lceil B \rceil. \text{“}\lceil A \rceil \text{ is of appropriate syntactic form”} \ \& \ Provable_L(\lceil A \rceil) \Rightarrow Provable_L(\lceil B \rceil)$$

*email scott@cs.jhu.edu, phone (410) 516-5299, fax (410) 516-6134.

Now, if a proof of B is desired and a proof of A is given, if A is of the appropriate form the above principle allows us to immediately conclude $Provable_L(\lceil B \rceil)$, whence B by the reflection axiom. Some systems incorporating this general paradigm are [ACHA90, BM81, DS79].

Cornell researchers [ACHA90] have formulated such a reflected proof hierarchy for the Nuprl type theory [CAB+86]. One advantage of studing reflected proof in such a context is there is a programming language built in to Nuprl, so it is possible to have the condition "$\lceil A \rceil$ is of appropriate syntactic form" be a decidible property that may actually be computed inside the Nuprl computation system, meaning the proof of that property also need not be constructed. Since formal proofs can get very large, this is an important method for shortening proof length.

Our goals are similar, but the method is different. Instead of reflecting proof, we reflect the *semantics*. Nuprl and other constructive type theories (CTT's) [Mar82] may be interpreted by inductively defining the types and their members [All87]. We show here that this inductive definition is actually expressible inside CTT. As in proof reflection there is a hierarchy of reflective semantic interpretations. The stratification is by the universe levels of the CTT. It is then possible to give an internal interpretation of proofs, leading to the *internal k-soundness* theorem. The hierarchy of internal k-soundness proofs then can be taken as an internal proof of soundness and thus consistency of the entire theory, contradicting Gödel's incompleteness theorem all but in name.

An axiom of self-awareness also may be added, and addition of the axiom does not lead to a hierarchy of proof theories, for the proof theory is not being reflected; the only hierarchy is based on universe levels.

2 Type Theory

This section is a short survey of constructive type theory (CTT) necessary for basic comprehension of this paper; for fuller descriptions and examples, see [CAB+86, Tho91]. The CTT used herein, called simply "CTT," is a variant of standard CTTs like Nuprl and Martin Löf's theories in that types do not come with equality $a = b \in A$; instead we just have $a \in A$, and equivalences are expressed via type-free equality $a \sim b$. This is type theory more in the spirit of Feferman class theory [Fef75].

We also wish to extend CTT to include a mechanism for recursive type definition. Recursive types for Nuprl have been defined by Mendler [CAB+86], and Dybjer has defined these types for Martin-Löf's theories [Dyb87]. CTT with recursive types added will be denoted CTTR. The internal presentation of the semantics of CTT is not possible without recursive types.

A type theory has as its language a collection of untyped *terms*. Some of these terms represent types, others computations; the two, perhaps surprisingly, are not separated. The terms of CTT include numbers and basic operations $0, 1, 2, \ldots$, $pred(a)$, $succ(a)$, $if\_zero(a; b; c)$; pairing and projection $\langle a, b \rangle$, $a.1$, $a.2$; injection and decision terms $inl(a)$, $inr(b)$, $decide(a; b; c)$; and functions and application $\lambda x.a$, $a(b)$. This language is thus a small functional programming language. On untyped terms we write $a \sim b$ meaning a is operationally equivalent to b, a notion we will not elaborate on here; see for instance [Plo75, Smi91a].

Letters a–d, A–D will range over terms, and v–z, V–Z will range over variables. Although terms and types are formally of the same sort, we use capitol letters to denote types and small letters to denote terms. Notions of *bound* and *free* variables, *open* and *closed* terms and captureless substitution of b for x in a $a[b/x]$ are standard.

2.1 Types

The expressiveness of type theory is largely due to the diversity of types that are definable. Here we list the types of CTT and CTTR.

(i) N is a type of natural numbers.

(ii) $a \sim b$ is a type that is inhabited (by placeholder 0) just when a and b are indistinguishable as computations.

(iii) $a\ in\ A$ is a type that internalizes the assertion $a \in A$: $0 \in (a\ in\ A)$ just when $a \in A$.

(iv) $x{:}A \times B(x)$ is a dependent product type (the *type* $B(x)$ depends on the *member* of the type A) whose inhabitants are pairs $\langle a, b\rangle$, with $a \in A$ and $b \in B(a)$.

(v) $x{:}A \to B(x)$ is a dependent function type whose inhabitants are functions $\lambda x.b$, where for all $a \in A$, $(\lambda x.b)(a) \in B(a)$.

(vi) $\mathrm{U}_1, \mathrm{U}_2, \ldots, \mathrm{U}_k, \ldots$ are universes or large types, types that have types as members. The levels are constructed in sequence: U_1 has as members all types closed under the type forming operators; U_2 has as members all types closed under the same operators, plus the type U_1; *et cetera.*

(vii) CTTR has parameterized recursive types in addition to all the types mentioned up to now. $rec(X; x{:}A.B(X)(x))(a)$ is a parameterized recursive type. In simplified non-parametric form $rec(X; B(X))$, it denotes the solution to a recursive type equation of the form $X = B(X)$. A parameter x is added to give solutions to more general type equations $X(x) = B(X)(x)$, with initial parameter $a \in A$.

2.2 Formulas as types

One of the defining features of constructive type theory is it is a theory of realizability. Formulas are viewed as types for which the members are the realizers of the formula. To prove the formula, show that when it is viewed as a type, the type is inhabited, i.e. has some member. Formulas are defined in terms of types as follows.

$$
\begin{array}{ll}
A \Rightarrow B & \stackrel{\text{def}}{=} A \to B \\
A \,\&\, B & \stackrel{\text{def}}{=} A \times B \\
A \vee B & \stackrel{\text{def}}{=} A + B \stackrel{\text{def}}{=} x{:}\mathrm{N} \times \mathit{if\_zero}(x; A; B) \\
\neg A & \stackrel{\text{def}}{=} A \to \mathit{false} \\
\mathit{false} & \stackrel{\text{def}}{=} 0 \sim 1 \\
\forall x{:}A.\ B & \stackrel{\text{def}}{=} x{:}A \to B \\
\exists x{:}A.\ B & \stackrel{\text{def}}{=} x{:}A \times B
\end{array}
$$

The type universes U_k also serve as universes of formulas, and the type $A \to \mathrm{U}_k$ has as members predicates on the type A.

2.3 Refinement proof and extraction

An assumption list Γ is of the form $x_1 \in A_1, x_2 \in A_2, \ldots x_n \in A_n$. One form of assertion may be made:

$$\Gamma \vdash a \in A$$

which asserts under assumptions Γ, term a inhabits type A.

The rules of CTT may be presented in goal-directed or *refinement-style* fashion: a rule is applied to a goal, and this gives subgoals which when proven realize the goal. Proofs are thus trees with nodes being goals and children of a node being its subgoals. The leaves of the tree are goals with no subgoals. A sample rule, the introduction rule for (non-dependent) products, is

$$\begin{array}{l} \vdash \langle a, b\rangle \in A \times B \\ \quad \vdash a \in A \\ \quad \vdash b \in B \end{array}$$

where the first line is the goal and the indented lines are the subgoals. A full set of rules for CTT is given in appendix A.

3 Semantics of CTT

CTT and CTTR may be proven sound by giving an inductive definition of the types and their inhabitants, using the method of Allen [All87]. Types are defined by induction on their structure. In the presence of dependent types it is impossible to first define the types and then define membership predicates for the types, because for a term $x{:}A \to B(x)$ to be a type, $B(a)$ must be a type for all members a of type A. This means the *members* of A must be defined before the *type* $x{:}A \to B(x)$ is considered well-formed. The solution Allen developed is to simultaneously define types and their inhabitants. This inductive definition can be viewed as a term model of CTT.

The main technical result of this paper is to show that Allen's method of inductive definition may be expressed inside CTTR itself, giving an internal definition of truth. We first give the inductive definition of CTT and show it has reasonable properties. The definition is then expressed in CTTR. Inductive definition of recursive types requires considerable extra notation, so we will settle for a summary at the end of the paper of the steps necessary to inductively define CTTR. Mendler [Men87] has developed techniques for giving an inductive definition of recursive types.

The development of the semantics of CTT now proceeds in three phases: untyped equality on terms is defined, the inductive definition of types is given, and the theory is proven sound with respect to this definition.

First, an evaluator for untyped closed terms and equivalence thereupon is defined. See [Smi91b] for complete definitions.

DEFINITION 3.1 For closed CTT terms a and b, $a \mapsto b$ iff a computes under call-by-name syntactic reduction to b.

Equality $a \sim b$ is defined to mean a and b are indistinguishable when placed in any context. $c[-]$ is defined to be a context, i.e. a term with a distinguished hole "$-$" in which another term may be placed. $c[a]$ denotes $c[-]$ with a placed in the hole, possibly capturing free variables of a.

DEFINITION 3.2 $a \sim b$ iff for all closing contexts $c[-]$. $c[a] \mapsto a'$ iff $c[b] \mapsto b'$.

A collection of properties may be proven about this equivalence that in turn allows the $\sim$ rules of CTT to be justified; see [Smi91b].

Given the collection of terms with equivalence relation, the types and their inhabitants may be simultaneously defined. Open terms are considered only when interpreting hypothetical assertions, so until that point all terms may be implicitly taken to be closed.

DEFINITION 3.3 A *type interpretation* is a two-place predicate $\tau(A, \epsilon)$ where A is a term, and ϵ is a one-place predicate on terms.

For $\tau(A, \epsilon)$ to be true means in type interpretation τ, A is a type with its members specified by ϵ. The following definitions make this explicit.

DEFINITION 3.4 (i) A Type$_\tau$ iff $\tau(A, \epsilon)$ for some ϵ

(ii) $a \in_\tau A$ iff $\tau(A, \epsilon)$ for some ϵ and $\epsilon(a)$.

A Type$_\tau$ thus means A is a type in interpretation τ, and $a \in_\tau A$ means a is a member of the type A.

Recall we have both small types and a hierarchy of large types U_k. The universes are predicative, so $\mathrm{U}_k \in \mathrm{U}_k$ fails. This is important because it means larger universes may be defined in terms of the (small) type constructors and the smaller universes. The definition of the full theory thus proceeds a universe at a time: ν_k is the type interpretation for universe U_k.

To give a well-formed inductive type definition, a monotonic operator is defined, and the least fixed point is then taken. The proof of monotonicity of the operator will be omitted.

DEFINITION 3.5 ν_k is the least fixed-point of the following monotonic operator Ψ_k on type interpretations:

$\Psi_k(\tau) \stackrel{\text{def}}{=} \tau'$, where $\tau'(T, \epsilon)$ is true if and only if

EITHER $T \sim \mathrm{N}$, in which case
for all t, $\epsilon(t)$ iff $t \sim n$, where n is $0, 1, 2, \ldots$
OR $T \sim (a \sim b)$, in which case
for all t, $\epsilon(t)$ iff $t \sim 0$ and $a \sim b$
OR $T \sim a \ in \ A$, in which case A Type$_\tau$ and
for all t, $\epsilon(t)$ iff $t \sim 0$ and $a \in_\tau A$
OR $T \sim x{:}A \to B(x)$, in which case
A Type$_\tau$ and for all $a \in_\tau A$, $B(a)$ Type$_\tau$, and
for all t, $\epsilon(t)$ iff $t \sim \lambda x.b$ and for all $a \in_\tau A$, $t(a) \in_\tau B(a)$
OR $T \sim x{:}A \times B(x)$, in which case
A Type$_\tau$ and for all $a \in_\tau A$, $B(a)$ Type$_\tau$, and
for all t, $\epsilon(t)$ iff $t \sim \langle a, b \rangle$ and $a \in_\tau A$ and $b \in_\tau B(a)$
OR $T \sim U_{k'}$, in which case
$k' < k$ and for all t, $\epsilon(t)$ iff $\nu_{k'}(t, \epsilon')$ for some ϵ'.

This completes the definition is each universe level, and thus the entire CTT. Some abbreviations are now made: $a \in_k A \stackrel{\text{def}}{=} a \in_{\nu_k} A$, $a \in A \stackrel{\text{def}}{=} a \in_k A$ for some k, and A Type$_k \stackrel{\text{def}}{=} A$ Type$_{\nu_k}$, A Type $\stackrel{\text{def}}{=} A$ Type$_k$ for some k.

Some straightforward lemmas about typehood and membership are now proven. They can be taken as defining what it means for the different forms of term to be types, and for what it means to be an inhabitant of the different types.

LEMMA 3.6 Type formation is characterized by the following properties.

(i) N Type, $a \sim b$ Type.

(ii) $a\ in\ A$ Type iff A Type.

(iii) $x{:}A \to B(x)$ Type iff A Type and for all $a \in A$, $B(a)$ Type.

(iv) $x{:}A \times B(x)$ Type iff A Type and for all $a \in A$, $B(a)$ Type.

(v) if $A \in \mathrm{U}_k$ then A Type.

(vi) if A Type then $A \in \mathrm{U}_k$ for some k.

(vii) if $a \in A$ then A Type.

LEMMA 3.7 Type membership is characterized by the following properties.

(i) $c \in a \sim b$ iff $c \sim 0$ and $a \sim b$.

(ii) $c \in \mathrm{N}$ iff $c \sim 0, 1, 2, \ldots$.

(iii) $c \in (a\ in\ A)$ iff $(a\ in\ A)$ Type and $c \sim 0$ and $a \in A$.

(iv) $c \in x{:}A \to B(x)$ iff $x{:}A \to B(x)$ Type and $c \sim \lambda x.c(x)$ and for all $a \in A$, $c(a) \in B(c)$.

(v) $c \in x{:}A \times B(x)$ iff $x{:}A \times B(x)$ Type and $c \sim \langle c.1, c.2\rangle$ and $c.1 \in A$ and $c.2 \in B(c.1)$.

These two collections of lemmas closely correspond to the type formation and introduction/elimination rules, respectively. From the lemmas, it is then direct to show

THEOREM 3.8 (SOUNDNESS) If $\vdash a \in A$ in CTT, then $a \in A$, meaning the interpretation is sound.

COROLLARY 3.9 (INTUITIONISTIC CONSISTENCY) There is no proof of $\vdash 0 \in (0 \sim 1)$ in CTT.

4 Reflective semantics of CTT

The previous section gave a semantics of CTT; in this section we show how this semantics may be phrased inside CTTR. The key insights are that parameterized recursive types are powerful enough that general inductive definitions may be expressed, and that the type definitions ν_k may be expressed in U_{k+1}, meaning each universe is defined inside the next one.

A *Gödelization* of the CTT terms consists of a type *Term* encoding all terms of the theory, together with operations to construct, destruct, and compare terms. To convert terms to and from Gödelized form, we use metafunctions $\lceil a \rceil$ and $\lfloor a \rfloor$, respectively. We also will use the convention that a metavariable with a hat ($\hat{a}$) means the variable is a Gödelized term, so $\hat{a} \in \mathit{Term}$. The proof theory (the rules of appendix A) may also be Gödelized as follows.

(i) *Sequent* is a type of sequents, lists of pairs of terms.

(ii) *Ptree* is a type of primitive proof trees.

(iii) *Proof* is a type of proof trees that are well-formed, i.e. the provability of the children of a node imply the provability of the node by one of the rules.

(iv) $\mathit{Provable}(\lceil \Gamma \vdash a \in A \rceil)$ iff $\lceil \Gamma \vdash a \in A \rceil$ is the top node of a *Proof* tree.

A rigorous Gödelization of the terms and proofs of CTT in CTT may be found in [ACHA90]; details will not be given here.

Since here we are here concerned with truth and not just proof, the evaluator and equivalence defined in the previous section also need to be expressed in CTT. Define an n-step evaluator $\mathit{Eval} \in \mathit{Term} \times \mathrm{N} \to \mathit{Term}$, and an untyped term equality predicate $\mathit{Equal} \in \mathit{Term} \times \mathit{Term} \to \mathrm{U}_1$. For this paper, we take these as given: they amount to expressing the definitions of $\mapsto$ and $\sim$ inside CTT. Since the proofs in [Smi91a] are constructive, this internalization should be routine.

The internalized equality should properly reflect the external equality.

LEMMA 4.1 (i) If for some t, $t \in \mathit{Equal}(\lceil a \rceil, \lceil b \rceil)$, then $a \sim b$.

(ii) If for some t, $t \in \neg \mathit{Equal}(\lceil a \rceil, \lceil b \rceil)$, then $a \not\sim b$.

4.1 The inductive definition

We will define a family of predicates $CTT_k \in \mathit{Term} \times (\mathit{Term} \to \mathrm{U}_k) \to \mathrm{U}_{k+1}$ for $k = 1, 2, 3, \ldots$. Each CTT_k is intended to be an internal expression of the type definition ν_k. Then, it is possible to prove a family of theorems in CTT that collectively express its own soundness and thus consistency.

The predicates CTT_k are now defined inductively using parameterized recursive types. The definition below thus is only expressible in CTTR; see section 5 for a sketch of how predicates $CTTR_k$ may be defined.

DEFINITION 4.2

$$
\begin{aligned}
CTT_k \stackrel{\mathrm{def}}{=} \lambda\langle \hat{A}, M_{\hat{A}} \rangle.\mathit{rec}(X; \langle \hat{A}, M_{\hat{A}} \rangle{:}\mathit{Term} \times \mathit{Term} \to \mathrm{U}_k.\\
\quad \mathit{Equal}(\hat{A}, \lceil \mathrm{N} \rceil) \;\&\; \forall \hat{a}{:}\mathit{Term}.\, M_{\hat{A}}(\hat{a}) \iff \exists \hat{n}{:}\mathit{Term}.\, \mathit{Equal}(\hat{a}, \hat{n}) \;\&\; \mathit{Natnum}(\hat{n})\\
\qquad \vee\\
\quad \exists \hat{b}, \hat{c}{:}\mathit{Term}.\, \mathit{Equal}(\hat{A}, \lceil \lfloor \hat{b} \rfloor \sim \lfloor \hat{c} \rfloor \rceil) \;\&\;\\
\qquad \forall \hat{a}{:}\mathit{Term}.\, M_{\hat{A}}(\hat{a}) \iff \mathit{Equal}(\hat{a}, \lceil 0 \rceil) \;\&\; \mathit{Equal}(\hat{b}, \hat{c})\\
\qquad \vee\\
\quad \exists \hat{b}, \hat{B}{:}\mathit{Term}.\, \mathit{Equal}(\hat{A}, \lceil \lfloor \hat{b} \rfloor \ \mathit{in}\ \lfloor \hat{B} \rfloor \rceil) \;\&\; \exists M_{\hat{B}}{:}\mathit{Term} \to \mathrm{U}_k.\, X(\langle \hat{B}, M_{\hat{B}} \rangle) \;\&\;\\
\qquad \forall \hat{a}{:}\mathit{Term}.\, M_{\hat{A}}(\hat{a}) \iff \mathit{Equal}(\hat{a}, \lceil 0 \rceil) \;\&\; M_{\hat{B}}(\hat{b})\\
\qquad \vee\\
\quad \exists \hat{B}{:}\mathit{Term}.\, \exists \hat{C}{:}\mathit{Term} \to \mathit{Term}.\, \mathit{Equal}(\hat{A}, \lceil x{:}\lfloor \hat{B} \rfloor \to \lfloor \hat{C} \rfloor(x) \rceil) \;\&\; \exists M_{\hat{B}}{:}\mathit{Term} \to \mathrm{U}_k.\, X(\langle \hat{B}, M_{\hat{B}} \rangle) \;\&\;\\
\qquad \exists M_{\hat{C}}{:}\mathit{Term} \to \mathit{Term} \to \mathrm{U}_k.\, \forall \hat{b}{:}\mathit{Term}.\, M_{\hat{B}}(\hat{b}) \Rightarrow X(\langle \hat{C}(\hat{b}), M_{\hat{C}}(\hat{b}) \rangle) \;\&\;\\
\qquad \forall \hat{a}{:}\mathit{Term}.\, M_{\hat{A}}(\hat{a}) \iff \mathit{Equal}(\hat{a}, \lceil \lambda x.\lfloor \hat{a} \rfloor(x) \rceil) \;\&\; \forall \hat{b}{:}\mathit{Term}.\, M_{\hat{B}}(\hat{b}) \Rightarrow M_{\hat{C}}(\hat{b})(\lceil \lfloor \hat{a} \rfloor(\lfloor \hat{b} \rfloor) \rceil)\\
\qquad \vee\\
\quad \exists \hat{B}{:}\mathit{Term}.\, \exists \hat{C}{:}\mathit{Term} \to \mathit{Term}.\, \mathit{Equal}(\hat{A}, \lceil x{:}\lfloor \hat{B} \rfloor \times \lfloor \hat{C} \rfloor(x) \rceil) \;\&\; \exists M_{\hat{B}}{:}\mathit{Term} \to \mathrm{U}_k.\, X(\langle \hat{B}, M_{\hat{B}} \rangle) \;\&\;\\
\qquad \exists M_{\hat{C}}{:}\mathit{Term} \to \mathit{Term} \to \mathrm{U}_k.\, \forall \hat{b}{:}\mathit{Term}.\, M_{\hat{B}}(\hat{b}) \Rightarrow X(\langle \hat{C}(\hat{b}), M_{\hat{C}}(\hat{b}) \rangle) \;\&\;\\
\qquad \forall \hat{a}{:}\mathit{Term}.\, M_{\hat{A}}(\hat{a}) \iff \mathit{Equal}(\hat{a}, \lceil \langle \lfloor \hat{a} \rfloor.1, \lfloor \hat{a} \rfloor.2 \rangle \rceil) \;\&\; M_{\hat{B}}(\lceil \lfloor \hat{a} \rfloor.1 \rceil) \;\&\; M_{\hat{C}}(\lceil \lfloor \hat{a} \rfloor.1 \rceil)(\lceil \lfloor \hat{a} \rfloor.2 \rceil)\\
\qquad \vee\\
\quad \mathit{Equal}(\hat{A}, \lceil \mathrm{U}_{k-1} \rceil) \;\&\; \forall \hat{B}{:}\mathit{Term}.\, M_{\hat{A}}(\hat{B}) \iff \exists M_{\hat{B}}{:}\mathit{Term} \to \mathrm{U}_{k-1}.\, CTT_{k-1}(\hat{B}, M_{\hat{B}})\\
\qquad \vee\\
\qquad \vdots\\
\qquad \vee\\
\quad \mathit{Equal}(\hat{A}, \lceil \mathrm{U}_1 \rceil) \;\&\; \forall \hat{B}{:}\mathit{Term}.\, M_{\hat{A}}(\hat{B}) \iff \exists M_{\hat{B}}{:}\mathit{Term} \to \mathrm{U}_1.\, CTT_1(\hat{B}, M_{\hat{B}})\,)(\langle \hat{A}, M_{\hat{A}} \rangle)
\end{aligned}
$$

LEMMA 4.3 $CTT_k \in \mathrm{U}_{k+1}$.

This completes the definition; now we establish its reasonableness. A first step is the proof of soundness of the previous section may be internalized. First, we define the notion of a proof occurring entirely within some maximum universe.

DEFINITION 4.4 For arbitrary k, $Provable_k(t)$ iff $Provable(t)$ and all universes appearing in the proof are of level at most k.

Since all proofs have some maximum universe level, if $Provable(t)$ then $Provable_k(t)$ for some k.

All k-provable statements can then internally be shown sound.

THEOREM 4.5 (INTERNAL k-SOUNDNESS) For any natural number k, the following theorem may be proved in CTTR: $\vdash \forall \hat{t}, \hat{T}{:}Term.Provable_k(\ulcorner \vdash \lfloor \hat{t} \rfloor \in \lfloor \hat{T} \rfloor \urcorner) \Rightarrow \exists M{:}Term \to \mathrm{U}_k.\ CTT_k(\hat{T}, M)\ \&\ M(\hat{t})$.

PROOF. This is just a matter of repeating the arguments of the previous section inside type theory. The most interesting facet of that procedure is proofs by induction on the definition of ν_k now proceed by the (Rec induction) rule on CTT_k definitions.
QED.

COROLLARY 4.6 (INTERNAL k-CONSISTENCY) The consistency of proofs at any particular level k may then be proven, i.e. $\vdash \neg Provable_k(\ulcorner 0 \in (0 \sim 1) \urcorner)$.

The interpretation of $CTT_k(\ulcorner A \urcorner, M)$ in the external semantics ν_k may be connected to the external semantics itself, and this proves useful.

THEOREM 4.7 (INTERNAL-EXTERNAL) If $t \in_{k+1} \exists M{:}Term \to \mathrm{U}_k.\ CTT_k(\ulcorner A \urcorner, M)\ \&\ M(\ulcorner a \urcorner)$ then $\nu_k(A, \epsilon)$ and $\epsilon(a)$.

This means internal truth implies external truth. Rules may be added to the theory to reflect this fact.

CTT is extended to give the theory CTTI by adding the rule (In), which we present for the simplified case that the hypothesis list is empty.

$$\text{(In)} \qquad \frac{\vdash a \in A \qquad \vdash t \in \exists M{:}Term \to \mathrm{U}_k.\ CTT_k(\ulcorner A \urcorner, M)\ \&\ M(\ulcorner a \urcorner)}{}$$

With the (In) rule, proving a term is in a type is accomplished by showing the term to be in the membership predicate for the type in the internal semantics. This rule is justified to be sound by theorem 4.7.

5 Semantics of Recursive Types

Recursive types present more difficulties. For $rec(X; x{:}A.B(X)(x))(a)$ to be a type, the formation rule states the body $B(X)(x)$ must be a type for all types X, but this means a new type is defined by quantifying over all types, obviously circular. The solution is to use Girard's candidat de réductibilité method [Gir71] and interpret X as an arbitrary predicate, not a type. This breaks the circularity, but at the expense of needing environments to bind all such X to some predicate. Mendler [Men87] gives a detailed semantics for CTT with parameterized recursive types. The reflective semantics for the full CTTR type theory then involves encoding Mendler's method inside CTTR.

5.1 Uses of reflective semantics

We sketch some application of these ideas through a simple example. Many types are obviously well-formed, but in an automated implementation of a type theory such as Nuprl, the complete proofs of well-formedness must be given. Since a significant amount of processing time is spent proving types are well-formed, it would be useful to be able to immediately recognize some simple types and avoid constructing their proofs of well-formedness.

Suppose the function $SimpleType(\hat{A})$ was defined such that $SimpleType(\hat{A}) \sim true$ just when $\hat{A}$ meets some criteria of being a simple well-formed type. Then, we may prove a theorem

$$\vdash \forall \hat{A}{:}Term.\ SimpleType(\hat{A}) \sim true \Rightarrow \exists M{:}Term \to \mathrm{U}_1.\ CTTR_2(\lceil \lfloor \hat{A} \rfloor \in \mathrm{U}_1 \rceil, M)$$

meaning all such simple types may be shown, by the internal semantics, to be types. Then, any time it is necessary to show a type well-formed, $\vdash A \in \mathrm{U}_1$, we may try the following procedure:

(i) Compute $SimpleType(\lceil A \rceil)$; fail if the result is not *true*.

(ii) apply the (In) rule to give the subgoal $\vdash CTTR_2(\lceil A \in \mathrm{U}_1 \rceil, M)$.

(iii) using the above theorem and modus ponens, the proof is complete.

A The rules

The rules will be presented refinement-style, giving a goal sequent, followed by subgoal sequents which if proven imply the truth of the goal.

We confine various technical conventions for the presentation of the rules to this paragraph. The hypothesis list is always increasing going from goal to subgoals, even though subgoals do not list goal hypotheses. In the hypothesis list, x_i may occur free in any A_{i+j} for positive j. The free variables in the conclusion are no more than the x_i. α-conversion is an unmentioned rule. To improve readability, the assertion $0 \in a \sim b$ will be abbreviated $a \sim b$, likewise for $0 \in (a\ in\ A)$. Since there is at most one inhabiting object, 0, it need not be mentioned. Also, in hypothesis lists, $x \in a \sim b$ will be abbreviated $a \sim b$, since x is known to be 0. Abbreviate $a \sim b \to 0 \sim 1$ as $a \not\sim b$.

A.1 Computation

The type $a \sim b$ asserts a and b are equivalent computations. In accordance with the principle of propositions-as-types, this type is inhabited (by the placeholder 0) just when it is true.

(Computation) $\quad \Gamma \vdash t \sim t'$

where t and t' are one of the following pairs:

t	t'
$(\lambda x.b)(a)$	$b[a/x]$
$\langle a, b \rangle.1$	a
$\langle a, b \rangle.2$	b
$succ(n)$	n', where n' is one larger than n
$pred(n)$	n', where n' is one smaller than n or 0 if $n = 0$
$if\_zero(0; a; b)$	a.

(Computation) $\quad \Gamma \vdash if\_zero(a; b; c) \sim c$
$\vdash a \not\sim 0$

(Contradict) $\Gamma, a \sim b \vdash c \in C$

where a and b are different values by inspection.

(Sim refl) $\Gamma \vdash a \sim a$

(Sim sym) $\Gamma, a \sim b \vdash b \sim a$

(Sim trans) $\Gamma, a \sim b, b \sim c \vdash a \sim c$

A.2 Membership

The expression $a\ in\ A$ reifies the assertion $a \in A$ as a type; $0 \in a\ in\ A$ just when $a \in A$. The inhabitant 0 is implicit in the rules below.

(Member intro) $\Gamma \vdash a\ in\ A$
$\vdash a \in A$

(Member elim) $\Gamma \vdash a \in A$
$\vdash a\ in\ A$

A.3 Natural number

N is a type of natural numbers.

(N intro) $\Gamma \vdash a \in \mathrm{N}$

Where a is one of 0, 1, 2,....

(N succ) $\Gamma \vdash succ(a) \in \mathrm{N}$
$\vdash a \in \mathrm{N}$

There is a symmetric rule (N pred).

(N induction) $\Gamma, a \in \mathrm{N} \vdash b(a) \in B(a)$
$a \sim 0 \vdash b(a) \in B(a)$
$a \not\sim 0, x \in \mathrm{N}, x \not\sim 0, b(pred(x)) \in B(pred(x)) \vdash b(x) \in B(x)$

A.4 Dependent function

Dependent functions $x{:}A \to B(x)$ are also commonly notated $\Pi x{:}A.B(x)$. We let $A \to B$ abbreviate $x{:}A \to B$ where x is not free in B.

(Func intro) $\Gamma \vdash \lambda x.b \in x{:}A \to B(x)$
$x \in A \vdash b \in B(x)$
$\vdash A \in \mathrm{U}_k$

(Func elim) $\Gamma \vdash c \in C$
$b \sim \lambda x.b(x), b(a) \in B(a) \vdash c \in C$
$\vdash a \in A$
$\vdash b \in x{:}A \to B(x)$

where x is not free in b.

A.5 Dependent product

Dependent products $x{:}A \times B(x)$ are also commonly notated $\Sigma x{:}A.B(x)$, but "dependent product" is a more apt computational description. These are types of pairs for which the type of the second component of the pair depends on the value of the first component. Let $A \times B$ abbreviate $x{:}A \times B$ where x is not free in B.

(Prod intro) $\Gamma \vdash \langle a, b \rangle \in x{:}A \times B(x)$
$\vdash a \in A$
$\vdash b \in B(a)$
$x \in A \vdash B(x) \in \mathrm{U}_k$

The third subgoal assures that $x{:}A \times B(x)$ is a sensible type.

(Prod elim) $\Gamma \vdash c \in C$
$a \sim \langle a.1, a.2 \rangle, a.1 \in A, a.2 \in B(a.1) \vdash c \in C$
$\vdash a \in x{:}A \times B(x)$

A.6 Universe

(U form) $\Gamma \vdash \mathrm{U}_k \in \mathrm{U}_{k+1}$

(U cumulativity) $\Gamma \vdash A \in \mathrm{U}_{k+1}$
$\vdash A \in \mathrm{U}_k$

($\sim$ form) $\Gamma \vdash a \sim b \in \mathrm{U}_k$

(Member form) $\Gamma \vdash (a\ in\ A) \in \mathrm{U}_k$
$\vdash A \in \mathrm{U}_k$

(N form) $\Gamma \vdash \mathrm{N} \in \mathrm{U}_k$

(Func form) $\Gamma \vdash x{:}A \to B(x) \in \mathrm{U}_k$
$\vdash A \in \mathrm{U}_k$
$x \in A \vdash B \in \mathrm{U}_k$

(Prod form) $\Gamma \vdash x{:}A \times B(x) \in \mathrm{U}_k$
$\vdash A \in \mathrm{U}_k$
$x \in A \vdash B(x) \in \mathrm{U}_k$

A.7 Miscellaneous

(Cut) $\Gamma \vdash a \in A$
$\vdash b \in B$
$x \in B \vdash a \in A$

(Hypothesis) $\Gamma \vdash x \in A$
where $x \in A$ occurs in Γ.

(Subst) $\Gamma \vdash c[a] \in C[a]$
$\vdash a \sim b$
$\vdash c[b] \in C[b]$

(Prop member) $\Gamma, x \in A \vdash x \sim 0$
where A is $b\ in\ B$, or $a \sim b$.

A.8 Recursive Types and CTTR

CTTR is CTT extended to have parameterized recursive types. $rec(X; x{:}A.B(X)(x))(a)$ denotes solutions to parameterized recursive type equations $X(x) \stackrel{\text{def}}{=} B(X)(x)$, where x is a parameter and has initial value a. Parameterized recursive types for CTT are presented in [CAB+86], chapter 12, and another version appears in [Men87]. Define $R_1 \subseteq_A R_2$ for $R_{1/2} \in A \to U_k$ as $\forall x{:}A.\, \forall y{:}R_1(x).\, y \in R_2(x)$.

(Rec intro)
$$\Gamma \vdash b \in rec(X; x{:}A.B(X)(x))(a)$$
$$\vdash b \in B(rec(X; x{:}A.B(X)(x)))(a)$$
$$\vdash rec(X; x{:}A.B(X)(x))(a) \in \mathrm{U}_k$$

(Rec elim)
$$\Gamma, y \in rec(X; x{:}A.B(X)(x))(a) \vdash c \in C$$
$$y \in B(rec(X; x{:}A.B(X)(x)))(a) \vdash c \in C$$

(Rec induction)
$$\Gamma, x \in rec(X; x{:}A.B(X)(x)))(a) \vdash c(x)(a) \in C(x)(a)$$
$$Z \in A \to \mathrm{U}_k, y \in A, z \in (\forall y{:}A.\, x \in Z(y) \to c(x)(y)\ in\ C(x)(y)), x \in B(Z)(y)$$
$$\vdash c(x)(y)\ in\ C(x)(y)$$

(Rec form)
$$\Gamma \vdash rec(X; x{:}A.B(X)(x)) \in A \to \mathrm{U}_k$$
$$X \in A \to \mathrm{U}_k, x \in A \vdash B(X)(x) \in \mathrm{U}_k$$
$$X_1 \in A \to \mathrm{U}_k, X_2 \in A \to \mathrm{U}_k, y \in X_1 \subseteq_A X_2 \vdash e \in B(X_1) \subseteq_A B(X_2)$$

The second subgoal assures the body $B(X)(x)$ to be monotonic, making the least fixed point sensible.

References

[ACHA90] S. F. Allen, R. L. Constable, D. Howe, and W. Aitken. The semantics of reflected proof. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 95–105, 1990.

[All87] S. F. Allen. A non-type-theoretic semantics for type-theoretic language. Technical Report 87-866, Department of Computer Science, Cornell University, September 1987. Ph.D. Thesis.

[BM81] R. S. Boyer and J S. Moore. Metafunctions: Proving them correct and using them efficiently as new proof procedures. In R. S. Boyer and J Strother Moore, editors, *The Correctness Problem in Computer Science*, chapter 3. Academic Press, 1981.

[CAB+86] R. L. Constable, S. F. Allen, H. Bromley, W. R. Cleveland, J. Cremer, R. Harper, D. Howe, T. Knoblock, N. P. Mendler, P. Panangaden, J. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[DS79] M. Davis and J. T. Schwartz. Metamathematical extensibility for theorem verifiers and proof-checkers. *Computers and Mathematics with Applications*, 5:217–230, 1979.

[Dyb87] P. Dybjer. Inductively defined sets in Martin-Löf's set theory. Technical report, Department of Computer Science, University of Göteburg/Chalmers, 1987.

[Fef62] S. Feferman. Transfinite recursive progressions of axiomatic theories. *Journal of Symbolic Logic*, 27:259–316, 1962.

[Fef75] S. Feferman. A language and axioms for explicit mathematics. In J. N. Crossley, editor, *Algebra and Logic*, volume 450 of *Lecture notes in Mathematics*, pages 87–139. Springer-Verlag, 1975.

[Gir71] J.-Y. Girard. Une extension de l'interprétation de Gödel à l'Analyse, et son application à l'Élimination des coupures dans l'Analyse et la Théorie des types. In J. E. Fenstad, editor, *Second Scandinavian Logic Symposium*, pages 63–92, Amsterdam, 1971. North-Holland.

[Mar82] P. Martin-Löf. Constructive mathematics and computer programming. In *Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, Amsterdam, 1982. North Holland.

[Men87] P. F. Mendler. Inductive definition in type theory. Technical Report 87-870, Department of Computer Science, Cornell University, September 1987. Ph.D. Thesis.

[Plo75] G. Plotkin. Call-by-name, call-by-value, and the λ-calculus. *Theoretical Computer Science*, pages 125–159, 1975.

[Smi91a] S. F. Smith. From operational to denotational semantics. In *MFPS 1991*, Lecture notes in Computer Science, 1991. (To appear).

[Smi91b] S. F. Smith. Partial computations in constructive type theory. Submitted to *Journal of Logic and Computation*, 1991.

[Tho91] S. Thompson. *Type Theory and Functional Programming.* Addison-Wesley, 1991.