Strong Normalization for the Calculus of Constructions

Chris Casinghino

December 2010

1 Introduction

The calculus of constructions (CC) is a core theory for dependently typed programming and higher-order constructive logic. Originally introduced in Coquand's 1985 thesis [4], CC has inspired 25 years of research in programming languages and type theory. Today, extensions of CC form the basis of languages like Coq [17] and Agda [15, 16],

The popularity of CC can be attributed to the combination of its expressiveness and its pleasant metatheoretic properties. Among these properties, one of the most important is *strong normalization*, which means that there are no infinite reduction sequences from well-typed expressions. This result has two important consequences. First, it implies that CC is consistent as a logic. This makes it an attractive target for the formalization of mathematics. Second, it implies that there is an algorithm to check whether two expressions are β-convertible. Thus, type checking is decidable and CC provides a practical basis for programming languages.

The strong normalization theorem has traditionally been considered difficult to prove [5, 2]. Coquand's original proof was found to have at least two errors, but a number of later papers give different, correct proofs [5]. In subsequent years, many authors considered how to extend this result for additional programming constructs like inductive datatypes with recursion, a predicative hierarchy of universes, and large eliminations [18, 9, 11]. Many of these proofs are even more challenging, and several span entire theses.

This document reviews three proofs of strong normalization for CC. Each paper we have chosen proves the theorem by constructing a model of the system in a different domain, and each contributes something novel to the theory of CC and its extensions. The technical details of the models are often complicated and intimidating. Rather than comprehensively verifying and reproducing the proofs, we have focused on painting a clear picture of the beautiful and fascinating mathematical structures that underpin them.

The first proof, originally presented by Geuvers and Nederhof [8] and subsequently popularized by Barendregt [3], models CC in the simpler theory of F_ω. It demonstrates that the strong normalization theorems for CC and F_ω are equivalent by giving a reduction-preserving translation from the former to the latter. The second, by Geuvers [7], models CC's types with sets of expressions. The paper demonstrates how the model may be extended to cope with several popular language features, aiming for flexibility. The last proof, from Melliès and Werner [13], uses realizability semantics to consider a large class of type theories, known as the pure type systems, which include CC. The authors' goal is to prove strong normalization for any pure type system that enjoys a particular kind of realizability model.

Though each paper has a unique focus and models CC in a different semantic system, the overall structures are very similar. After unifying the syntax, the correspondences between certain parts of the proofs are quite striking. Readers are encouraged, for example, to compare the interpretation functions defined in Sections 6.2 and 6.3 with those in Section 5.2. The similarities between the papers speak to the fundamental underlying structure of CC, while their differences illustrate how design choices can push the proof towards varying goals.

The paper is structured as follows: In Sections 2 and 3 we review the definition and basic metatheory of pure type systems and the calculus of constructions. We present the high-level structure of a strong normalization argument in Section 4, then the proofs of Geuvers and Nederhof [8], Geuvers [7] and Melliès and Werner [13] in Sections 5, 6 and 7, respectively. We compare the proofs and conclude in Section 8.

2 Pure Type Systems

The calculus of constructions is one example of a *pure type system* (PTS). This very general notion, introduced by Berardi and popularized by Barendregt [3], consists of a parameterized lambda calculus which can be instantiated to a variety of well-known type systems. For example, the simply-typed lambda calculus, System F, System F_ω and CC are all pure type systems. The PTS generalization is convenient because it allows us to simultaneously study the properties of several systems.

A PTS is specified by three parameters. First, the collection of *sorts* s is given by a set $\mathcal{S}$. The typing hierarchy among these sorts is given by a collection of *axioms* $\mathcal{A} \subseteq \mathcal{S}^2$. Finally, the way product types may be formed is specified by the set of *rules* $\mathcal{R} \subseteq \mathcal{S}^3$. Figure 1 gives the complete definition of the system.

Choosing to explain CC as a PTS settles several questions of presentation. The terms, types and kinds are collapsed into one grammar. Some authors choose to separate these levels syntactically for easier identification, but we find this version more economical and it is more closely aligned with the three papers under consideration. For the same reasons, we have used β-conversion in the Conv rule instead of using a separate judgemental equality (as is done, for example, in [2]). Here, $=_\beta$ is the symmetric, transitive, reflexive closure of $\rightsquigarrow$. We do not consider η-conversion.

$$
\begin{array}{ll}
A,\ B,\ a,\ b & ::= s \mid x \mid (x:A) \to B \mid \lambda x:A.b \mid a\ b \\
\Gamma & ::= \cdot \mid \Gamma, x\!:\!A
\end{array}
$$

$\boxed{a \rightsquigarrow b}$

$$\frac{}{(\lambda x:A.b)\ a \rightsquigarrow [a/x]b}\ \textsc{SBeta}$$

$$\frac{A \rightsquigarrow A'}{(x:A) \to B \rightsquigarrow (x:A') \to B}\ \textsc{SPi1} \qquad \frac{A \rightsquigarrow A'}{\lambda x:A.b \rightsquigarrow \lambda x:A'.b}\ \textsc{SLam1} \qquad \frac{a \rightsquigarrow a'}{a\ b \rightsquigarrow a'\ b}\ \textsc{SApp1}$$

$$\frac{B \rightsquigarrow B'}{(x:A) \to B \rightsquigarrow (x:A) \to B'}\ \textsc{SPi2} \qquad \frac{b \rightsquigarrow b'}{\lambda x:A.b \rightsquigarrow \lambda x:A.b'}\ \textsc{SLam2} \qquad \frac{b \rightsquigarrow b'}{a\ b \rightsquigarrow a\ b'}\ \textsc{SApp2}$$

$\boxed{a \rightsquigarrow^* b}$

$$\frac{}{a \rightsquigarrow^* a}\ \textsc{MSRefl} \qquad \frac{\begin{array}{l} a_1 \rightsquigarrow a_2 \\ a_2 \rightsquigarrow^* a_3 \end{array}}{a_1 \rightsquigarrow^* a_3}\ \textsc{MSStep}$$

$\boxed{\Gamma \vdash a : A}$

$$\frac{\begin{array}{l} \vdash \Gamma \\ (s_1, s_2) \in \mathcal{A} \end{array}}{\Gamma \vdash s_1 : s_2}\ \textsc{TSort} \qquad \frac{\begin{array}{l} \vdash \Gamma \\ (x:A) \in \Gamma \end{array}}{\Gamma \vdash x : A}\ \textsc{TVar} \qquad \frac{\begin{array}{l} \Gamma \vdash a : A \qquad \Gamma \vdash B : s \\ A =_\beta B \end{array}}{\Gamma \vdash a : B}\ \textsc{TConv}$$

$$\frac{\begin{array}{l} \Gamma \vdash A : s_1 \\ \Gamma, x\!:\!A \vdash B : s_2 \\ (s_1, s_2, s_3) \in \mathcal{R} \end{array}}{\Gamma \vdash (x:A) \to B : s_3}\ \textsc{TPi} \qquad \frac{\begin{array}{l} \Gamma \vdash (x:A) \to B : s \\ \Gamma, x\!:\!A \vdash b : B \end{array}}{\Gamma \vdash \lambda x:A.b : (x:A) \to B}\ \textsc{TLam} \qquad \frac{\begin{array}{l} \Gamma \vdash a : (x:A) \to B \\ \Gamma \vdash b : A \end{array}}{\Gamma \vdash a\ b : [b/x]B}\ \textsc{TApp}$$

$\boxed{\vdash \Gamma}$

$$\frac{}{\vdash \cdot}\ \textsc{CNil} \qquad \frac{\begin{array}{l} x \notin \mathsf{dom}(\Gamma) \\ \Gamma \vdash A : s \end{array}}{\vdash \Gamma, x\!:\!A}\ \textsc{CCons}$$

Figure 1: Definition of Pure Type Systems

This context also permits a clean and compartmentalized explanation of CC's features. In most of the systems we consider, the sorts and axioms are given by the sets:

$$\mathcal{S} = \{*, \Box\} \qquad \mathcal{A} = \{(*, \Box)\}$$

Intuitively, $*$ classifies types and $\Box$ classifies kinds. The lone axiom says that $*$ is itself a kind. The rule $(*, *, *)$ permits standard function types, whose domain and codomain are both types. The system with only this rule is the simply-typed lambda calculus:

$$\mathcal{R} = \{(*, *, *)\}$$

The rule $(\Box, *, *)$ permits functions whose domain is a kind. For example, when the domain is $*$ these are functions which takes types as arguments (i.e., polymorphism). Thus, adding this rule yields System F:

$$\mathcal{R} = \{(*, *, *),\ (\Box, *, *)\}$$

The rule $(\Box, \Box, \Box)$ effectively duplicates STLC at the type level. It allows functions that take and return types. Adding it yields System F_ω, which has type-level computation:

$$\mathcal{R} = \{(*, *, *),\ (\Box, *, *),\ (\Box, \Box, \Box)\}$$

CC adds *dependent types* to System F_ω. The rule $(*, \Box, \Box)$ permits types to depend on terms by allowing functions which take terms as arguments but return types. Thus, the complete specification of CC is:

$$\mathcal{S} = \{*, \Box\} \qquad \mathcal{A} = \{(*, \Box)\} \qquad \mathcal{R} = \{(*, *, *),\ (\Box, *, *),\ (\Box, \Box, \Box),\ (*, \Box, \Box)\}$$

3 Simple Metatheory

For completeness, we review a few basic metatheoretic results. We will write $\Gamma \vdash a : A$ for the typing judgement of an arbitrary PTS or when it is clear what system we are discussing, and otherwise will label the turnstile as in $\Gamma \vdash_{CC} a : A$ for CC's typing relation in particular.

The first result, confluence, can be proven using the standard Tait–Martin-Löf technique [12, 11].

Theorem 3.1 (Confluence)**.** *If $a \rightsquigarrow^* a_1$ and $a \rightsquigarrow^* a_2$ then there is a b such that $a_1 \rightsquigarrow^* b$ and $a_2 \rightsquigarrow^* b$.*

The second property, preservation, is proved by induction on typing derivations, using a substitution lemma.

Theorem 3.2 (Preservation)**.** *If $\Gamma \vdash a : A$ and $a \rightsquigarrow b$ then $\Gamma \vdash b : A$.*

The last property is not usually considered for less expressive lambda calculi because they are presented with separate syntax for terms, types and kinds. The theorem says that CC expressions can still be classified in this way with the typing judgement. It is proved by a straightforward induction on typing derivations.

Theorem 3.3 (Classification)**.** If $\Gamma \vdash_{CC} A : B$, then exactly one of the following holds:

- B is $\Box$. In this case, we call A a *kind*.
- $\Gamma \vdash_{CC} B : \Box$. In this case, we call A a Γ-*constructor*.
- $\Gamma \vdash_{CC} B : *$. In this case, we call A a Γ-*term*.

When B is $*$, we will call A a Γ-*type*. This is a special case of the second bullet above. In this document we use the word "expression" to refer to any element of CC's grammar and reserve the word "term" for the subclass of expressions identified here.

Notice that we need a context to distinguish between constructors and terms, but can identify kinds without one. The ambiguity comes from variables, and some authors avoid it by splitting them into two syntactic classes (typically x, y, z for term variables and α, β for type variables). Distinguishing the variables in this way forces duplication or subtle inaccuracy when discussing binders at different levels. For that reason, we prefer to mix the variables and use a context to identify the terms and constructors.

Finally, we define the central notion considered below:

Definition 3.4. An expression is called *strongly normalizing* if there are no infinite $\rightsquigarrow$ reduction sequences beginning at it. We write SN for the collection of all such expressions.

4 Structure of the proofs

The three proofs we consider each model CC in a different domain, but they share a similar overall structure. In this section we describe the technique at a high level.

Step 1: Define the interpretations

Each proof begins by defining two interpretations. A "type" interpretation, usually written $[\![A]\!]$, captures the static meaning of types, kinds and sorts. For example, in the second proof we will model types as sets of expressions so that $[\![A]\!]$ contains all the terms of type A. Then a "term" interpretation is defined to capture the run-time behavior of terms, types and kinds. This is usually written $[a]$. In the example where types are interpreted as sets of expressions, the term interpretation might pick a canonical inhabitant with the right reduction behavior from the set.

Step 2: Relate the interpretations

After defining the term and type interpretations, we prove a theorem that relates them. For example, in the second proof we will show that if $\Gamma \vdash a : A$, then $[a] \in [\![A]\!]$. This theorem is usually called “soundness”.

Step 3: Declare success

After proving the soundness theorem we observe that one of the interpretations has some important property. This property will mean that strong normalization is a direct consequence of the soundness theorem. In the running example, $[\![A]\!]$ will turn out to contain only strongly normalizing expressions. Then, since $[a] \in [\![A]\!]$ and $[a]$ models a's run-time behavior, $a \in \mathsf{SN}$.

A clarification about the interpretations

Though we have called $[\![\cdot]\!]$ the “type” interpretation and $[\cdot]$ the “term” interpretation, we do not mean that the former is only defined on types and the later on terms, in the sense of the classification theorem. Rather, $[\![\cdot]\!]$ is meant to model the static meaning of any expression that can be used to classify other expressions. In each proof it will be defined on all constructors, kinds and sorts of CC. Correspondingly, $[\cdot]$ is meant to model the dynamic behavior of any expression which can take reduction steps. It will be defined on the terms, constructors and kinds of CC.

5 Modeling CC in $\mathbf{F}_\omega$

The first proof we consider translates CC expressions to System F_ω in a way that preserves reduction. System F_ω is known to be strongly normalizing (see [6] for a detailed proof), so the correctness of this translation will imply that CC is strongly normalizing as well. The idea to prove strong normalization of an expressive type theory by translation to a better-understood system has been used in a variety of contexts. For example, Harper et al. [10] demonstrated that LF is strongly normalizing by giving a reduction-preserving translation to the simply typed lambda calculus. This technique was originally applied to CC by Geuvers and Nederhof [8], and their proof is reproduced in Barendregt [3].

While this development does not have the same focus on extensibility or generality as the later two, it has at least two advantages. First, the proof is modular. The other two proofs we will see are monolithic in that they must explain the unique features of CC while recapitulating and extending a complicated semantic argument. Here we may focus on the ways in which CC extends F_ω and can rely on the somewhat simpler semantics of that system. Second, the translation itself is simple and can be verified in Peano arithmetic. Thus, this technique demonstrates that the proof-theoretic complexity of CC's strong normalization argument is no greater than that of F_ω.

5.1 Intuition for the translation

The calculus of constructions extends System F_ω with dependency in the form of the rule $(*, \Box, \Box)$. This rule permits type-level abstractions which create types but take terms as arguments. The difficulty comes from modeling these functions in System F_ω without erasing any possible reduction sequences.

To do this, we will translate expressions in two distinct ways. The "type" translation $[\![\cdot]\!]$ erases the dependencies to create F_ω types from CC types. The "term" translation $[\cdot]$ keeps the dependencies to avoid erasing any possible reductions, but lowers type functions to the level of terms. The soundness theorem of our translation will state

$$\text{if } \Gamma \vdash_{CC} a : A \ \text{ then } \ [\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} [a] : [\![A]\!].$$

We follow Geuvers and Nederhof [8] in exhibiting how these translations handle several examples before specifying them in full detail. Consider a simple example of dependency where F is a dependent type function, A a type and a a term, so that:

$$\frac{\Gamma \vdash_{CC} F : A \to * \qquad \Gamma \vdash_{CC} a : A}{\Gamma \vdash_{CC} F\ a : *}$$

The subderivation which checks the type of F will need to make use of rule $(*, \Box, \Box)$. We must somehow erase this use of dependency so that, in F_ω:

$$\frac{[\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} [F] : [\![A \to *]\!] \qquad [\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} [a] : [\![A]\!]}{[\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} [F\ a] : [\![*]\!]}$$

To solve this, we take $[\![A \to *]\!] = [\![A]\!] \to 0$ where $0 : *$ is a fixed type variable that is added to the context by $[\![\Gamma]\!]$. We set $[\![*]\!] = 0$ and $[F\ a] = [F]\,[a]$. Now when checking the translated F we have a term-level function rather than one which returns a type.

For our second example, suppose A and B are types and a is a term of type A. When translating the application $(\lambda x : A.B)\, a$, we must erase A to an F_ω type using the type translation $[\![A]\!]$. However, this admits the possibility that by erasing dependency we will delete redexes. This is solved by inserting an extra redex which does nothing but provide a spot to hold A's translation as a term. That is, for some fresh variable y

$$[(\lambda x : A.B)\ a] = (\lambda y : 0.\lambda x : [\![A]\!].[B])\,[A]\,[a].$$

The situation for polymorphism is similar. Consider a constructor F with kind $(x : *) \to x \to x$ (for example, the polymorphic identity function) and a type $A : *$. In translating the term $F\,A$, we must preserve A's static meaning as a type without erasing any possible reduction sequences. The solution is to use both translations, again:

$$\begin{array}{ll} [\![(x : *) \to x \to x]\!] & = (x : *) \to 0 \to x \to x \\ [F\ A] & = [F]\,[\![A]\!]\,[A] \end{array}$$

The theme of these examples is that the two translations accomplish different tasks. The type translation $[\![\cdot]\!]$ erases dependencies to make F_ω types out of CC types. The term translation $[\cdot]$ preserves reduction behavior but lowers CC types to F_ω terms in order to accommodate the weaker type system. We translate parts of expressions twice so that we can achieve both goals.

5.2 The translation of types and contexts

Now we give the complete definition of the translation functions. We begin by owning up to a slight simplification in the last section. To distinguish term variables from type variables, the translations must be indexed by contexts. Thus, the translation for types becomes $[\![\cdot]\!]_\Gamma$, and the translation for terms becomes $[\cdot]_\Gamma$. The translation for contexts, $[\![\cdot]\!]$, remains unindexed.

In addition to these functions we define V which translates CC sorts and kinds to F_ω kinds:

$$\begin{aligned}
V(\Box) &= * \\
V(*) &= * \\
V((x:A) \to B) &= \begin{cases} V(A) \to V(B) & \text{if } A \text{ is a kind} \\ V(B) & \text{otherwise} \end{cases}
\end{aligned}$$

This function is not indexed by a context because CC kinds may be distinguished without one, by the classification theorem. The reason for the case split in the last clause is that we are erasing dependency.

The translation of types from CC to F_ω follows the examples from the previous section. The domain of $[\![\cdot]\!]_\Gamma$ is the sorts, kinds, and Γ-constructors of CC. We pick a unique type variable 0 and assume it is never used in an input to this function.

$$\begin{aligned}
[\![\Box]\!]_\Gamma &= 0 \\
[\![*]\!]_\Gamma &= 0 \\
[\![x]\!]_\Gamma &= x \\
[\![(x:A) \to B]\!]_\Gamma &= \begin{cases} (x : V(A)) \to [\![A]\!]_\Gamma \to [\![B]\!]_{\Gamma,x:A} & \text{if } A \text{ is a kind} \\ (x : [\![A]\!]_\Gamma) \to [\![B]\!]_{\Gamma,x:A} & \text{if } A \text{ is a } \Gamma\text{-type} \end{cases} \\
[\![(\lambda x : A.B)]\!]_\Gamma &= \begin{cases} \lambda x : V(A).[\![B]\!]_{\Gamma,x:A} & \text{if } A \text{ is a kind} \\ [\![B]\!]_\Gamma & \text{if } A \text{ is a } \Gamma\text{-type} \end{cases} \\
[\![A\ B]\!]_\Gamma &= \begin{cases} [\![A]\!]_\Gamma\ [\![B]\!]_\Gamma & \text{if } B \text{ is a } \Gamma\text{-constructor} \\ [\![A]\!]_\Gamma & \text{if } B \text{ is a } \Gamma\text{-term} \end{cases}
\end{aligned}$$

This function inserts duplication in product types as we discussed in the examples section. Otherwise, it is straightforward with the intuition that we are erasing dependency. The cases of the type translation that deal with functions take into account the level of the function's domain (just as we saw with V). This distinction is justified by the classification lemma and is reflected in the substitution lemma for the translation:

Lemma 5.1 ($[\![\cdot]\!]_\Gamma$ respects substitution)**.** Suppose A is a kind or Γ-constructor in CC. When $x : B \in \Gamma$ and $\Gamma \vdash_{CC} b : B$, we have:

- $[\![[b/x]A]\!]_\Gamma = [[\![b]\!]_\Gamma/x][\![A]\!]_\Gamma$, if B is a kind.
- $[\![[b/x]A]\!]_\Gamma = [\![A]\!]_\Gamma$, if B is a Γ-type.

This lemma can be shown by induction on the typing derivation. It follows that the translation of types preserves β-conversion:

Lemma 5.2 ($[\![\cdot]\!]_\Gamma$ preserves $=_\beta$)**.** Suppose A and A' are kinds or Γ-constructors in CC such that $A =_\beta A'$. Then $[\![A]\!]_\Gamma =_\beta [\![A']\!]_\Gamma$.

Before we can state that the results of $[\![\cdot]\!]_\Gamma$ are classified by V, we must extend the translation to contexts. As mentioned, $[\![\cdot]\!]$ will add a type variable $0 : *$ to the context. There are two additional changes. First, a variable $z : (x : *) \to *$ will be added to help provide a canonical inhabitant for each type. Second, for each kind variable x which appears in Γ, the translation will add another variable $w^x : x$. This last change simply ensures that contexts match up with the translation of product types, where we add an extra argument in the case of kinds as discussed above.

We define the translation of contexts in two parts. First, a function $[\![x : A]\!]_\Gamma$ maps each context binding to one or two translated bindings:

$$[\![x : A]\!]_\Gamma = \begin{cases} x : [\![A]\!]_\Gamma, w^x : x & \text{if } A \text{ is a } \Gamma\text{-kind} \\ x : [\![A]\!]_\Gamma & \text{if } A \text{ is a } \Gamma\text{-type} \end{cases}$$

The translation of a context simply maps this last function onto each binding and adds 0 and z to the front, as mentioned. Suppose $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, then:

$$[\![\Gamma]\!] = 0 : *,\ z : (x : *) \to *,\ [\![x_1 : A_1]\!]_\Gamma,\ \ldots,\ [\![x_n : A_n]\!]_\Gamma$$

Now the soundness of the translation of types follows straightforwardly by induction on typing derivations

Lemma 5.3 (Soundness of $[\![\cdot]\!]_\Gamma$)**.** Suppose A is a sort, kind or Γ-type of CC such that $\Gamma \vdash_{CC} A : B$. Then $[\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} [\![A]\!]_\Gamma : V(B)$.

5.3 The translation of terms

As mentioned in the last section, the translation of contexts permits the construction of a canonical inhabitant of each type or kind in F_ω. In particular, for any expression B such that $[\![\Gamma]\!] \vdash_{F_\omega} B : s$, we will define a term c^B of type B in the same context. That is, $[\![\Gamma]\!] \vdash_{F_\omega} c^B : B$. If $s = *$, then we may use the term z to construct c^B:

$$c^B = z\, B \text{ when } B \text{ is a type}$$

Otherwise, B is a kind and we define:

$$\begin{aligned} c^* &= 0 \\ c^{(x:A)\to B} &= \lambda x : A.c^B \end{aligned}$$

The evaluation behavior of these canonical inhabitants is not very important. The chief purpose of c is to help in the term translation of product types. The problem is that when $(x : A) \to B$ is a valid CC type, its translation $(x : [A]_\Gamma) \to [B]_\Gamma$ is not necessarily well-typed in F_ω. The translation $[\![\cdot]\!]_\Gamma$ handles this by erasing dependency, but $[\cdot]_\Gamma$ must retain all the possible reductions which begin at $(x : A) \to B$. Instead of translating it as a product, we use c to construct a function whose application to A and B is well-typed. In particular, $c^{0\to0\to0}\, [A]_\Gamma\, [B]_\Gamma$ will be a valid F_ω expression. Since $[\cdot]_\Gamma$ does not erase the terms from A and B, this retains all the possible reduction sequences.

We now present the full translation of terms:

$$\begin{aligned}
[*]_\Gamma &= c^0 \\
[x]_\Gamma &= \begin{cases} w^x & \text{if } x \text{ is a } \Gamma\text{-type} \\ x & \text{if } x \text{ is a } \Gamma\text{-term} \end{cases} \\
[(x : A) \to B]_\Gamma &= \begin{cases} c^{0\to0\to0}\, [A]_\Gamma\, ([c^{V(A)}/x][c^{[\![A]\!]_\Gamma}/w^x][B]_{\Gamma,x A}) & \text{if } A \text{ is a kind} \\ c^{0\to0\to0}\, [A]_\Gamma\, ([c^{[\![A]\!]_\Gamma}/x][B]_{\Gamma,x A}) & \text{if } A \text{ is a } \Gamma\text{-type} \end{cases} \\
[\lambda x : A.b]_\Gamma &= \begin{cases} (\lambda y : 0.\lambda x : V(A).\lambda w^x : [\![A]\!]_\Gamma.[b]_{\Gamma,x A})\, [A]_\Gamma & \text{if } A \text{ is a kind, picking } y \text{ fresh} \\ (\lambda y : 0.\lambda x : [\![A]\!]_\Gamma.[b]_{\Gamma,x A})\, [A]_\Gamma & \text{if } A \text{ is a } \Gamma\text{-type, picking } y \text{ fresh} \end{cases} \\
[A\, B]_\Gamma &= \begin{cases} [A]_\Gamma\, [\![B]\!]_\Gamma\, [B]_\Gamma & \text{if } B \text{ is a } \Gamma\text{-type} \\ [A]_\Gamma\, [B]_\Gamma & \text{if } B \text{ is a } \Gamma\text{-term} \end{cases}
\end{aligned}$$

Theorem 5.4 (Soundness of $[\cdot]_\Gamma$)**.** If $\Gamma \vdash_{CC} a : A$ then $[\![\Gamma]\!] \vdash_{F_\omega} [a]_\Gamma : [\![A]\!]_\Gamma$.

As we have seen with previous soundness theorems, this proof is not conceptually surprising but requires a certain amount of book keeping. We show only one interesting case:

Proof. We go by induction on the structure of the derivation $\mathcal{D}$ of $\Gamma \vdash_{CC} a : A$.

Case: $\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash_{CC} (x : A) \to B : s} \qquad \overset{\mathcal{D}_2}{\Gamma, x{:}A \vdash_{CC} b : B}}{\Gamma \vdash_{CC} \lambda x : A.b : (x : A) \to B}$ TLam

Inversion on $\mathcal{D}_1$ yields a subderivation showing either that A is a kind that A is a Γ-type in CC. We will consider each possibility individually. Note that because $[\![\square]\!]_\Gamma = [\![*]\!]_\Gamma = 0$, in either case we have an induction hypothesis:

$$\mathrm{IH}_A : [\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} [A]_\Gamma : 0$$

- Suppose first that $\Gamma \vdash_{CC} A : *$. Unfolding the definitions of the translations, we see that we must show:

  $$[\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} (\lambda y : 0.\lambda x : [\![A]\!]_\Gamma.[b]_{\Gamma,x:A})\,[A]_\Gamma : (x : [\![A]\!]_\Gamma) \to [\![B]\!]_{\Gamma,x:A}$$

  Here y is some variable which doesn't occur in Γ, A, B or b. By IH_A and the TApp rule, it will be enough to show:

  $$[\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} \lambda y : 0.\lambda x : [\![A]\!]_\Gamma.[b]_{\Gamma,x:A} : (y : 0) \to (x : [\![A]\!]_\Gamma) \to [\![B]\!]_{\Gamma,x:A}$$

  Recall that $0 : *$ will appear in $[\![\Gamma]\!]$. By applying soundness for $[\![\cdot]\!]_\Gamma$ to the subderivations of $\mathcal{D}_2$, we find that $[\![A]\!]_\Gamma$ and $[\![B]\!]_{\Gamma,x:A}$ are also valid F_ω types in the contexts Γ and $\Gamma, x : A$, respectively. So by two applications of TPi and a standard weakening lemma for F_ω, we have:

  $$[\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} (y : 0) \to (x : [\![A]\!]_\Gamma) \to [\![B]\!]_{\Gamma,x:A} : *$$

  Therefore, by rule TLam, it will be enough to show:

  $$[\![\Gamma]\!], y{:}0 \vdash_{\mathrm{F}_\omega} \lambda x : [\![A]\!]_\Gamma.[b]_{\Gamma,x:A} : (x : [\![A]\!]_\Gamma) \to [\![B]\!]_{\Gamma,x:A}$$

  We have already observed that $(x : [\![A]\!]_\Gamma) \to [\![B]\!]_{\Gamma,x:A}$ is a valid F_ω type in this context. Thus, by another application of TLam, it is sufficient to show:

  $$[\![\Gamma]\!], y{:}0, x{:}[\![A]\!]_\Gamma \vdash_{\mathrm{F}_\omega} [b]_{\Gamma,x:A} : [\![B]\!]_{\Gamma,x:A}$$

  Observe that the IH for D_2 is close to this (after slightly unfolding the interpretation of the context):

  $$[\![\Gamma]\!], x{:}[\![A]\!]_\Gamma \vdash_{\mathrm{F}_\omega} [b]_{\Gamma,x:A} : [\![B]\!]_{\Gamma,x:A}$$

  And the result follows by a weakening lemma.

- Suppose instead that $\Gamma \vdash_{CC} A : \square$. After unfolding the translations, we must show:

  $$[\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} (\lambda y : 0.\lambda x : V(A).\lambda w^x : [\![A]\!]_\Gamma.[b]_{\Gamma,x:A})\,[A]_\Gamma : (x : V(A)) \to [\![A]\!]_\Gamma \to [\![B]\!]_{\Gamma,x:A}$$

  Here y is some fresh variable. By IH_A and rule TApp, it is enough to show:

  $$[\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} \lambda y : 0.\lambda x : V(A).\lambda w^x : [\![A]\!]_\Gamma.[b]_{\Gamma,x:A} : (y : 0) \to (x : V(A)) \to [\![A]\!]_\Gamma \to [\![B]\!]_{\Gamma,x:A}$$

As before, 0 is a $[\![\Gamma]\!]$-type and soundness for $[\![\cdot]\!]_\Gamma$ implies that $[\![A]\!]_\Gamma$ and $[\![B]\!]_{\Gamma,xA}$ are valid types in F_ω as well. The definition of V ensures that $V(A)$ is an F_ω kind. So by several applications of TPi and weakening for F_ω, we have:

$$[\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} (y : 0) \to (x : V(A)) \to [\![A]\!]_\Gamma \to [\![B]\!]_{\Gamma,xA} : *$$

Thus, by three applications of TLam, it is enough to show:

$$[\![\Gamma]\!], y\!:\!0, x\!:\!V(A), w^x\!:\![\![A]\!]_\Gamma \vdash_{\mathrm{F}_\omega} [b]_{\Gamma,xA} : [\![B]\!]_{\Gamma,xA}$$

This follows from the IH for $\mathcal{D}_2$, the observation that $[\![\Gamma, x\!:\!A]\!] = [\![\Gamma]\!], x\!:\!V(A), w^x\!:\! [\![A]\!]_\Gamma$, and weakening for F_ω. □

The soundness of the term translation demonstrates that it preserves the static semantics of CC expressions. We must also show that it preserves their reduction behavior. A lemma describing the way this function interacts with substitutions is needed. The duplication in the first case below mirrors the duplication we have discussed in the translation.

Lemma 5.5 (Substitution for $[\cdot]_\Gamma$)**.** Suppose $\Gamma \vdash_{CC} a : A$ and $(x : B) \in \Gamma$.

- If B is a kind and b is a Γ-type in CC, then

$$[[b/x]a]_\Gamma = [[\![b]\!]_\Gamma/x][[b]_\Gamma/w^x][a]_\Gamma$$

- If B is a Γ-type and b is a Γ-term in CC, then

$$[[b/x]a]_\Gamma = [[b]_\Gamma/x][a]_\Gamma$$

5.4 Strong Normalization

The final step in this proof is to relate reductions from CC expressions with reductions from their translations. The following result says that the term translation does not drop any reduction steps.

Theorem 5.6 ($[\cdot]_\Gamma$ preserves reduction)**.** Suppose $\Gamma \vdash_{CC} a : A$.

$$a \leadsto a' \quad \Rightarrow \quad [a]_\Gamma \leadsto^*_{\neq 0} [a']_\Gamma$$

Here, $\leadsto^*_{\neq 0}$ denotes reduction in at least one step.

Proof. The proof is by induction on the derivation that $a \leadsto a'$. The case of beta reduction uses Lemma 5.5. Each congruence case follows quickly by using an inversion lemma on the typing assumption and applying the induction hypothesis. □

Strong normalization for CC now follows quickly, using the same result for F_ω.

Theorem 5.7 (Strong normalization)**.** If $\Gamma \vdash_{CC} a : A$, then $a \in \mathsf{SN}$.

Proof. Assume for a contradiction that there is an infinite reduction sequence starting at a:

$$a \rightsquigarrow a_1 \rightsquigarrow a_2 \rightsquigarrow \dots$$

By preservation, $\Gamma \vdash_{CC} a_n : A$ for each n. Thus, by Lemma 5.6, there is another infinite sequence of reductions:

$$[a]_\Gamma \rightsquigarrow^*_{\neq 0} [a_1]_\Gamma \rightsquigarrow^*_{\neq 0} [a_2]_\Gamma \rightsquigarrow^*_{\neq 0} \cdots$$

But by the soundness of the term interpretation, we have $[\![\Gamma]\!] \vdash_{\mathrm{F}_\omega} [a]_\Gamma : [\![A]\!]_\Gamma$. This is a contradiction because the well-typed terms of F_ω are strongly normalizing. □

6 Modeling types as sets of expressions

The second proof we consider, from Geuvers [7], will be the most familiar to readers acquainted with the Girard–Tait method of reducibility candidates or saturated sets. The paper places a special emphasis on making the proof easy to extend with additional programming language constructs. To this end, only the metatheory we have introduced so far is required.[1] Several examples of extensions are included, and we consider some after the development for CC itself.

6.1 Basic definitions

We begin with a few definitions and results relating to reduction. Intuition for these ideas is important to understanding the main proof, so we discuss them in some detail.

Definition 6.1. Any expression of the form $x\ a_1\ \dots\ a_n$ is called a *base expression*. The set of base expressions is denoted BASE. Note that variables are base expressions (i.e., $n = 0$ is allowed).

Definition 6.2. With some expressions we associate another expression, called a *key redex*.

- The expression $(\lambda x : A.b)\ a$ is its own key redex.
- If A has a key redex, then $A\,B$ has the same key redex.

We denote by $\mathrm{red_k}(A)$ the expression obtained by reducing A's key redex, when it has one. Note that base expressions don't have key redexes. The intuition behind key redexes is that they can not be avoided. Reducing an expression without reducing its key redex leaves the redex in place. This intuition and the importance of key reduction is captured by the

[1] In fact, Geuvers requires a little less: he claims preservation isn't necessary. He still relies on substitution and a strong inversion lemma, though, so our presentation does not deviate too far from his proof.

following two lemmas. They are not difficult to prove, but they rely on a few other simple properties of beta reduction.

Lemma 6.3. Suppose a has a key redex and $a \rightsquigarrow b$ without reducing that redex. Then b has a key redex, and $\mathrm{red_k}(a) \rightsquigarrow^* \mathrm{red_k}(b)$.

It is helpful to visualize this lemma:

$$\begin{array}{ccc} a & \rightsquigarrow & b \\ \wr & & \wr \\ \mathrm{red_k}(a) & \rightsquigarrow^* & \mathrm{red_k}(b) \end{array}$$

Lemma 6.4. Suppose $a, b \in \mathsf{SN}$ and $\mathrm{red_k}(a\ b) \in \mathsf{SN}$. Then $a\ b \in \mathsf{SN}$.

Proof. Suppose for a contradiction that there is an infinite reduction sequence starting at $a\ b$. Since we know a and b are in SN, this means the application must beta-reduce in some finite number of steps. That is, the infinite sequence has a prefix of the form:

$$a\ b \rightsquigarrow^* (\lambda x : A.a')\ b' \rightsquigarrow [b'/x]a' \rightsquigarrow \ldots$$

Note that this last step reduces a key redex. Thus, by multiple applications of lemma 6.3, we have $\mathrm{red_k}(a\ b) \rightsquigarrow^* [b'/x]a'$. This is a contradiction, since $\mathrm{red_k}(a\ b) \in \mathsf{SN}$ but we found an infinite reduction sequence starting at it. $\square$

Saturated sets and their closure properties are the key technical device in the interpretation. Originally introduced by Tait, they are closely related to Girard's *candidates of reducibility* (for detailed comparisons, see [11] and [6]). The idea is pervasive, and we will see it again in the second proof.

Definition 6.5. A set of expressions S is called a *saturated set* if the following three conditions hold:

- $S \subseteq \mathsf{SN}$
- $(\mathsf{SN} \cap \mathsf{BASE}) \subseteq S$
- If $A \in \mathsf{SN}$ and $\mathrm{red_k}(A) \in S$ then $A \in S$.

The third condition states that saturated sets are "closed under the expansion of key redexes". We write SAT for the collection of all saturated sets. Note that $\mathsf{SN} \in \mathsf{SAT}$ and that every saturated set is non-empty.

Lemma 6.6. If S is a non-empty collection of saturated sets, then $\bigcap S \in \mathsf{SAT}$.

Definition 6.7. If S_1 and S_2 are sets of expressions, define:

$$\Pi(S_1,\, S_2) := \{a \mid \forall b \in S_1, a\ b \in S_2\}$$

It helps to have some intuition for this last definition: An expression a is in $\Pi(S_1,\ S_2)$ if whenever a is applied to an expression in S_1, you get an expression in S_2. Thus, when these sets model types, $\Pi(S_1,\ S_2)$ will contain the functions from the first type to the second. The next lemma, that $\Pi(\cdot,\ \cdot)$ preserves saturation, involves the most intricate reasoning about reduction that appears in this proof.

Lemma 6.8. If $S_1, S_2 \in \mathsf{SAT}$, then $\Pi(S_1,\ S_2) \in \mathsf{SAT}$.

Proof. There are three conditions to verify:

- ($\Pi(S_1,\ S_2) \subseteq \mathsf{SN}$) Suppose $a \in \Pi(S_1,\ S_2)$. Saturated sets are non-empty, so let $b \in S_1$ be given. Then $a\ b \in S_2$, and so $a\ b \in \mathsf{SN}$. Thus, $a \in \mathsf{SN}$.
- ($(\mathsf{SN} \cap \mathsf{BASE}) \subseteq \Pi(S_1,\ S_2)$) Let $a \in \mathsf{SN} \cap \mathsf{BASE}$ be given. For any $b \in S_1$, since $b \in \mathsf{SN}$, $a\ b \in \mathsf{SN} \cap \mathsf{BASE}$. Thus $a\ b \in S_2$. So, $a \in \Pi(S_1,\ S_2)$.
- ($\Pi(S_1,\ S_2)$ is closed under key redex expansion) Suppose $a \in \mathsf{SN}$ and $\mathrm{red_k}(a) \in \Pi(S_1,\ S_2)$. We must show $a \in \Pi(S_1,\ S_2)$, so let $b \in S_1$ be given. We have $\mathrm{red_k}(a)\ b \in S_2$, and must show $a\ b \in S_2$. But $\mathrm{red_k}(a)\ b = \mathrm{red_k}(a\ b)$. Since S_2 is closed under expansion of key redexes, it is enough to show that $a\ b \in \mathsf{SN}$. This follows immediately by lemma 6.4. □

6.2 Interpreting kinds

The interpretation of types comes in two steps. First we define a function $V(\,\cdot\,)$ on the sorts and kinds of CC. This is, roughly, the type of the main interpretation: if B is a kind or type such that $\Gamma \vdash B : A$, then B's interpretation will be an element of the set $V(A)$.

$$
\begin{aligned}
V(\Box) &= \mathsf{SAT} \\
V(*) &= \mathsf{SAT} \\
V((x : A) \to B) &= \begin{cases} \{f \mid f : V(A) \to V(B)\} & \text{when } A \text{ is a kind} \\ V(B) & \text{otherwise} \end{cases}
\end{aligned}
$$

By $\{f \mid f : V(A) \to V(B)\}$, we mean the collection of all (set-theoretic) functions from $V(A)$ to $V(B)$.

Lemma 6.9. If A is a kind, then $V(A)$ is non-empty.

As an example, consider the type $* \to *$. Notice that $V(* \to *)$ is the collection of all functions from saturated sets to saturated sets. So, when we interpret an expression with this type (say $\lambda x : *.x$), we will expect to get a function that takes collections of expressions to other collections of expressions. This makes sense, since it is a function from types to types.

The observant reader will notice that this definition of V mirrors the one from Section 5.2. Just as there, it indicates that we will ignore dependency in the interpretation of types. This will work because of the limited ways in which CC may use terms in types. For example, CC lacks large eliminations: even though we can encode natural numbers, we can not define types by pattern matching on them.

6.3 Interpreting types

Because our interpretation is not restricted to closed types, we begin by defining an environment that interprets the variables. Later, we will consider another similar environment for terms.

Definition 6.10. Given a context Γ such that $\vdash \Gamma$, a *constructor environment* σ for Γ is a function that maps the type variables of Γ to appropriate sets according to V. It should satisfy the condition:

$$\text{if } (x : A) \in \Gamma \wedge A \text{ is a kind, then } \sigma(x) \in V(A)$$

We'll write $\sigma \models \Gamma$ for this relation and $\sigma[x \mapsto S]$ for the constructor environment which maps x to S and otherwise agrees with σ.

Finally, we define the interpretation of types $[\![A]\!]_\sigma$ when A is a sort, kind or Γ-type:

$$\begin{aligned}
[\![\Box]\!]_\sigma &= \mathsf{SN} \\
[\![*]\!]_\sigma &= \mathsf{SN} \\
[\![x]\!]_\sigma &= \sigma(x) \\
[\![(x : A) \to B]\!]_\sigma &= \begin{cases} \Pi([\![A]\!]_\sigma, \bigcap\limits_{S \in V(A)} [\![B]\!]_{\sigma[x \mapsto S]}) & \text{when } A \text{ is a kind} \\ \Pi([\![A]\!]_\sigma, [\![B]\!]_\sigma) & \text{otherwise} \end{cases} \\
[\![\lambda x : A.b]\!]_\sigma &= \begin{cases} S \in V(A) \mapsto [\![b]\!]_{\sigma[x \mapsto S]} & \text{when } A \text{ is a kind} \\ [\![b]\!]_\sigma & \text{otherwise} \end{cases} \\
[\![a\ b]\!]_\sigma &= \begin{cases} [\![a]\!]_\sigma\ [\![b]\!]_\sigma & \text{when } b \text{ is a } \Gamma\text{-constructor} \\ [\![a]\!]_\sigma & \text{otherwise} \end{cases}
\end{aligned}$$

By $S \in V(A) \mapsto [\![b]\!]_{\sigma[x \mapsto S]}$, we mean the set-theoretic function that maps each S in $V(A)$ to $[\![b]\!]_{\sigma[x \mapsto S]}$.

This type interpretation is very similar to the one from Section 5.2. Many of the lemmas we will need also mirror results from that section. For example, compare the substitution lemma below with Lemma 5.1.

Lemma 6.11 ($[\![\cdot]\!]$ respects substitution)**.** Suppose $\sigma \models \Gamma$ and A is a kind or Γ-constructor. When $(x : B) \in \Gamma$ and $\Gamma \vdash b : B$, we have:

- $[\![[b/x]A]\!]_\sigma = [\![A]\!]_{\sigma[x \mapsto [\![b]\!]_\sigma]}$, if B is a kind.
- $[\![[b/x]A]\!]_\sigma = [\![A]\!]_\sigma$, if B is a Γ-type.

From this it follows that beta-convertible types have the same interpretation.

Lemma 6.12 ($[\![\cdot]\!]$ respects $=_\beta$)**.** Suppose $\sigma \models \Gamma$ and A_1, A_2 are kinds or Γ-constructors such that $A_1 =_\beta A_2$. Then $[\![A_1]\!]_\sigma = [\![A_2]\!]_\sigma$.

As promised, the range of the interpretation is classified by the function V. The proof is by induction on typing derivations. In the conversion case, Lemma 6.12 is used.

Lemma 6.13 (Soundness of $[\![\cdot]\!]$)**.** If $\sigma \models \Gamma$ and A is a kind or Γ-constructor such that $\Gamma \vdash A : B$, then $[\![A]\!]_\sigma \in V(B)$.

An important consequence of this lemma is that the interpretation of a type is always a saturated set and thus contains only strongly normalizing expressions.

6.4 From the interpretation to Strong Normalization

The key fact about the function $[\![\cdot]\!]$ is that every CC expression is in the interpretation of its type. Before we can prove this, we need a notion of environment for terms corresponding to σ for types.

Definition 6.14. We call a mapping on variables ρ a *term environment for* Γ *with respect to* σ when $\sigma \models \Gamma$ and:

$$\text{if } (x : A) \in \Gamma \text{ then } \rho(x) \in [\![A]\!]_\sigma$$

We write $\sigma \models \rho : \Gamma$ for this relation and $[A]_\rho$ for the expression created by simultaneously replacing the variables of A with their mappings in ρ. We write $\rho[x \mapsto a]$ for the term environment that sends x to a and otherwise agrees with ρ.

We show only the trickiest case of the key theorem—the complete proof may be found in the appendix. Though there are a number of details to keep track of, all the cleverness is in the definition of the interpretation; the result here is straightforward by induction.

Theorem 6.15 (Soundness of $[\cdot]$)**.** Suppose $\Gamma \vdash a : A$ and $\sigma \models \rho : \Gamma$. Then $[a]_\rho \in [\![A]\!]_\sigma$.

Proof. By induction on the derivation $\mathcal{D}$ of $\Gamma \vdash a : A$.

Case: $$\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ \Gamma \vdash A : s\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ \Gamma, x : A \vdash b : B\end{matrix}}{\Gamma \vdash \lambda x : A.b : (x : A) \to B}\ \textsc{TLam}$$

The IH for $\mathcal{D}_1$ gives us $[A]_\rho \in \mathsf{SN}$. Since x is a bound variable, we may pick it to be fresh for the domain and range of ρ. There are two subcases: s is either $*$ or $\Box$.

– Suppose s is $*$. Then we must show $\lambda x : [A]_\rho.[b]_\rho \in \Pi(\llbracket A \rrbracket_\sigma, \llbracket B \rrbracket_\sigma)$. So let $a \in \llbracket A \rrbracket_\sigma$ be given, and observe it is enough to show $(\lambda x : [A]_\rho.[b]_\rho)\ a \in \llbracket B \rrbracket_\sigma$.

  We have $\sigma \models \rho[x \mapsto a] : \Gamma, x : A$. Thus, the IH for $\mathcal{D}_2$ gives us $[b]_{\rho[x \mapsto a]} \in \llbracket B \rrbracket_\sigma$. But we also know

$$(\lambda x : [A]_\rho.[b]_\rho)\ a \rightsquigarrow [a/x][b]_\rho = [b]_{\rho[x \mapsto a]},$$

  and this step contracts a key redex. So, it suffices to show that $(\lambda x : [A]_\rho.[b]_\rho)\ a \in \mathsf{SN}$. This follows by lemma 6.4, using the classification lemma and lemma 6.13 to show the pieces of the application are in SN.

– Suppose instead that s is $\Box$. We must show $\lambda x : [A]_\rho.[b]_\rho \in \Pi(\llbracket A \rrbracket_\sigma, \bigcap_{S \in V(A)} \llbracket B \rrbracket_{\sigma[x \mapsto S]})$.

  Let an expression $a \in \llbracket A \rrbracket_\sigma$ and a saturated set $S \in V(A)$ be given. It is enough to show $(\lambda x : [A]_\rho.[b]_\rho)\ a \in \llbracket B \rrbracket_{\sigma[x \mapsto S]}$.

  Because $\sigma[x \mapsto S] \models \rho[x \mapsto a] : \Gamma, x : A$, the IH for $\mathcal{D}_2$ gives us that $[b]_{\rho[x \mapsto a]} \in \llbracket B \rrbracket_{\sigma[x \mapsto S]}$. As in the previous case, we can observe that

$$(\lambda x : [A]_\rho.[b]_\rho)\ a \rightsquigarrow [a/x][b]_\rho = [b]_{\rho[x \mapsto a]}.$$

  This step contracts a key redex, and by reasoning as in the last case we find $(\lambda x : [A]_\rho.[b]_\rho)\ a \in \llbracket B \rrbracket_{\sigma[x \mapsto S]}$ as desired. □

The last result quickly implies strong normalization:

Theorem 6.16. Suppose $\Gamma \vdash a : A$. Then $a \in \mathsf{SN}$.

Proof. For each kind A, let S_A be some canonical inhabitant of $V(A)$ (by lemma 6.9, these exist). Define a constructor environment σ such that, if $x : A \in \Gamma$ and A is a kind, then $\sigma(x) \mapsto S_A$. Define a term environment ρ such that each variable of Γ maps to itself.

To see $\sigma \models \rho : \Gamma$, observe first that $\vdash \Gamma$ (by induction on the typing derivation). Thus each type assigned by Γ itself has type $*$ or $\Box$. So their interpretations are saturated sets (lemma 6.13), which contain all the variables.

Thus, by the soundness of the interpretation, $a = [a]_\rho \in \llbracket A \rrbracket_\sigma$. But by the classification lemma and lemma 6.13, $\llbracket A \rrbracket_\sigma$ is a saturated set. So $a \in \mathsf{SN}$. □

6.5 Extensions

We conclude the presentation of this development by describing how it changes to accommodate several extensions to CC. We sketch each addition at a high level to give a sense of the proof's flexibility. Adding small Σ-types and W-types is encouragingly straightforward. Unfortunately, changes at the kind level turn out to be considerably more complicated.

Small Σ-types

Small Σ-types classify dependent pairs where the first component is a term. We extend the syntax of CC with four new constructs

$$A, B, a, b := \ldots \mid \Sigma x : A.B \mid (a, b) \mid \mathsf{proj}_1\ a \mid \mathsf{proj}_2\ a$$

and we add straightforward corresponding typing rules:

$$\frac{\Gamma \vdash A : * \qquad \Gamma, x{:}A \vdash B : s}{\Gamma \vdash \Sigma x : A.B : s}\ \textsc{TSigma} \qquad \frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : [a/x]B \qquad \Gamma \vdash \Sigma x : A.B : s}{\Gamma \vdash (a, b) : \Sigma x : A.B}\ \textsc{TPair}$$

$$\frac{\Gamma \vdash a : \Sigma x : A.B}{\Gamma \vdash \mathsf{proj}_1\ a : A}\ \textsc{TProj1} \qquad \frac{\Gamma \vdash a : \Sigma x : A.B}{\Gamma \vdash \mathsf{proj}_2\ a : [\mathsf{proj}_1\ a/x]B}\ \textsc{TProj2}$$

The reduction judgement must also change. The obvious congruence rules are needed for each construct, and there are two reduction rules to handle the case where the projection operations meet pairs:

$$\frac{}{\mathsf{proj}_1\ (a, b) \rightsquigarrow a}\ \textsc{SProj1} \qquad \frac{}{\mathsf{proj}_2\ (a, b) \rightsquigarrow b}\ \textsc{SProj2}$$

We make some simple changes to the definitions of base expressions and key reduction. These ensure that certain pair constructions will always appear in our interpretations. In particular, we extend Definition 6.1 with the following clause:

- If $a \in \mathsf{BASE}$ then $\mathsf{proj}_1\ a \in \mathsf{BASE}$ and $\mathsf{proj}_2\ a \in \mathsf{BASE}$.

And we extend Definition 6.2 with the following clause:

- If a has a key redex, then $\mathsf{proj}_1\ a$ and $\mathsf{proj}_2\ a$ have the same key redex.

The definition of saturated sets remains the same, and we define a new construction $X \otimes Y$ that is a saturated set whenever X and Y are:

$$X \otimes Y := \{a \mid \mathsf{proj}_1\ a \in X \wedge \mathsf{proj}_2\ a \in Y\}$$

This construction is used to extend the interpretation of types for dependent sums. Here, since we know x is a term variable, we do not need to extend σ in the interpretation of B (just as in the interpretation for product types and functions).

$$[\![\Sigma x : A.B]\!]_\sigma = [\![A]\!]_\sigma \otimes [\![B]\!]_\sigma$$

This new clause doesn't significantly alter the proofs of Lemmas 6.11, 6.12 and 6.13. Similarly, a quick inspection of the four new typing rules reveals that the soundness of the term interpretation (Theorem 6.15) follows directly by induction in these cases.

W-types

W-types add well-founded trees and recursion to the calculus of constructions. They are common in the literature as a small change that adds much of the expressive power of simple datatypes. We do not present their details, but a comprehensive introduction may be found in [14].

Extending the proof to support W-types is only a little harder than the previous example. Once again, the definitions of the base expressions and key reduction each get an extra clause. The main difficulty comes in defining a new construction on saturated sets to model the $\mathsf{W}x : A.B$ type constructor. The typing rule for this constructor is:

$$\frac{\Gamma \vdash A : * \qquad \Gamma, x\!:\!A \vdash B : *}{\Gamma \vdash \mathsf{W}x : A.B : *}\ \text{TW}$$

The type $\mathsf{W}x : A.B$ classifies well-founded trees where A describes the ways a tree may be formed and B describes the contents of the tree for each possible A. Unsurprisingly, the interpretation of this type involves a least fixed point over a particular monotone operator on saturated sets. Geuvers demonstrates that a suitable class of operators on saturated sets has least fixed points (an interesting exercise in set theory, but somewhat outside the scope of the current project).

After proving this property of saturated sets, the rest of the proof hardly changes. An extra case is added to the interpretation of types which uses a fixed point to interpret $\mathsf{W}x : A.B$. The cases involving the new typing rules for W-types are then straightforward by induction.

Large Σ-types

Large Σ-types classify dependent pairs where the first component is a constructor. Adding them to small Σ-types involves only one additional typing rule:

$$\frac{\Gamma \vdash A : \Box \qquad \Gamma, x\!:\!A \vdash B : s}{\Gamma \vdash \Sigma x : A.B : \Box}\ \textsc{TSigmaL}$$

This addition is more complicated than small product types because the $\otimes$ construction on saturated sets is no longer sufficient. Defining

$$[\![\Sigma x : A.B]\!]_\sigma = [\![A]\!]_\sigma \otimes [\![B]\!]_\sigma$$

is incorrect when A is a kind, because σ must contain interpretations for each of the type variables in B on the right-hand side.

This requires significant changes to the kind and type interpretations. Currently, $[\![A]\!]_\sigma$ is a saturated set when A is a kind. Instead, $[\![A]\!]_\sigma$ will be function from elements of $V(A)$ to SAT. Lemma 6.13 and Theorem 6.15 change as follows:

Lemma (Soundness of $[\![\cdot]\!]$)**.** Suppose $\sigma \models \Gamma$ and A is a kind or Γ-constructor such that $\Gamma \vdash A : B$.

- If A is a Γ-constructor, then $[\![A]\!]_\sigma \in V(B)$.
- If A is a kind, then $[\![A]\!]_\sigma \in \{f \mid f : V(A) \to \mathsf{SAT}\}$.

Theorem (Soundness of $[\cdot]$)**.** Suppose $\Gamma \vdash a : A$ and $\sigma \models \rho : \Gamma$.

- If A is a Γ-type, $[a]_\rho \in [\![A]\!]_\sigma$.
- If A is a kind, $[a]_\rho \in [\![A]\!]_\sigma([\![a]\!]_\sigma)$.

To illustrate these changes, we show the modified interpretations for the kind $\Sigma x : A.B$ when both A and B are kinds. In this case, we define

$$V(\Sigma x : A.B) = V(A) \times V(B)$$

where $\times$ is the standard set-theoretic product operator. The type interpretation of this kind uses the new function argument to fill in the gap we observed before:

$$[\![\Sigma x : A.B]\!]_\sigma = (X, Y) \in V(A) \times V(B) \mapsto [\![A]\!]_\sigma(X) \otimes [\![B]\!]_{\sigma[x \mapsto X]}(Y)$$

The other cases of $[\![\cdot]\!]$ that handle kinds must also be updated, but we omit the details. The proofs of every result involving the kind and type interpretations must be redone, but they are not harder.

7 Modeling pure type systems with realizability semantics

Note to the reader: The proof presented in this section is the most complicated of the three. The model it uses is substantially more complex than the previous two, and there are several technical problems with the paper under consideration. This section is included for completeness and to document some of the issues we encountered in reproducing the results. Casual readers are encouraged to skip the details.

Melliès and Werner [13] consider the question of strong normalization for a subset of the pure type systems. They define a "realizability" semantics parameterized in the same way as a PTS and show that, when such a model exists, the corresponding PTS is strongly normalizing. Their proof identifies four specific properties that the model must satisfy in order to guarantee strong normalization, and the paper exhibits suitable models for several systems. The idea of using realizability semantics to model CC was originally introduced by Altenkirch [1].

In this section we present their development. The results are particularly interesting in that the authors consider pure type systems which are more expressive than CC. For example, ECC (the extended calculus of constructions) adds an infinite hierarchy of predicative sorts to CC. Proving strong normalization for this system has traditionally been somewhat harder [11].

This proof is considerably more involved than the one in the previous section. In particular, we must define realizability models and lift many of the ideas already explored in the context of CC to this new domain, suitably generalized to work with any pure type system. The situation is additionally complicated because some of the theorems and proofs given in the paper are false or inadequate. We still believe the technique is worth presenting because of its promised generality and because it seems possible the problems here could be repaired. We will focus on giving intuition for the model constructions and avoid getting caught up in the proofs.

We begin by introducing *labeled* pure type systems with tight reduction (Section 7.1). The basic structures used in the interpretation are defined in Section 7.2, and in this context Section 7.3 illustrates one of the paper's errors. Section 7.4 identifies the four key properties that must hold of a model for the strong normalization proof to apply, and examples of suitable constructions are given for System F (Section 7.5) and CC (Section 7.7). The interpretation of expressions into these models is given in Section 7.8. Finally, Section 7.9 discusses the paper's attempt to prove strong normalization when a satisfactory model exists and to lift the result back to a PTS without the extra labels.

7.1 Labeled pure type systems and tight reduction

The proof we consider here is primarily concerned with *labeled* pure type systems which have more type annotations and a restricted reduction relation. In particular, the syntactic forms for function abstraction ($\lambda_{(x:A)\to B}.b$) and application ($\mathsf{app}_{(x:A)\to B}(b,a)$) are now labeled with the complete type of the function involved. Additionally, the rule for beta reduction has been modified to demand that the annotations match:

$$\frac{}{\mathsf{app}_{(x:A)\to B}((\lambda_{(x:A)\to B}.b),a) \rightsquigarrow_t [a/x]b}\ \textsc{TSBeta}$$

This new rule is known as *tight reduction*. The restrictions give us more information about types from the syntax itself, which can help to avoid potential circularity in the proof.

The complete specification of the modified system can be found in Figure 2. The only other significant change is in the conversion rule, which demands that one type is actually reducible to the other. This ensures that conversions take place in a path through the set of well-typed expressions. After proving that the expressions of this system are strongly normalizing, it will be relatively simple to lift the result back to a standard PTS by adding annotations.

In what follows we consider an arbitrary labeled PTS and let $\mathcal{T}$ stand for the set of its expressions. Many of our definitions from the previous section easily adapt to the new domain. The base expressions are now those with the form:

$$\mathsf{app}_{(y_n:A_n)\to B_n}(\ldots(\mathsf{app}_{(y_1:A_1)\to B_1}(x,a_1),\ldots),a_n)$$

The definitions of key redexes, saturated sets and the product construction $\Pi(S_1, S_2)$ on saturated sets remain the same. Syntactically, key reductions now look like this:

$$\begin{array}{ll} & \mathsf{app}_{(x_n:A_n)\to B_n}(\ldots(\mathsf{app}_{(x:A)\to B}(\lambda_{(x:A)\to B}.b, a), a_1), \ldots), a_n) \\ \leadsto_t & \mathsf{app}_{(x_n:A_n)\to B_n}(\ldots(\mathsf{app}_{(x_1:A_1)\to B_1}([a/x]b, a_1), \ldots), a_n) \end{array}$$

As before, we'll write $a = \mathrm{red_k}(b)$ when a is the labeled expression that results from reducing b's key redex.

7.2 Realizability constructions

We now define the basic constructions that will be used in the interpretation of a PTS.

Definition 7.1. A Λ-*set* is a pair $(X_0, \models)$ where X_0 is any set and $\cdot \models \cdot \subseteq \mathcal{T} \times X_0$ is a relation between the set of labeled expressions and X_0.

We call the elements of X_0 the *carriers*. When $\alpha \in X_0$, the *realizers* of α are the expressions A such that $A \models \alpha$. When X is a Λ-set we write X_0 for its first component, $\models_X$ for its second, and $\alpha \sqsubset X$ for $\alpha \in X_0$.

Roughly speaking, a type B will be modeled as a Λ-set whose realizers include the terms of type B. We can think of Λ-sets as sets of expressions with some extra structure provided by the carriers. Like sets of expressions, Λ-sets can be saturated:

Definition 7.2. A Λ-set X is *saturated* if:

- every realizer is strongly normalizing,
- there is a carrier that is realized by every element of $\mathsf{BASE} \cap \mathsf{SN}$, and
- if $\alpha \sqsubset X$, the realizers of α are closed under the expansion of key redexes. That is, if $a \models \alpha$, $a = \mathrm{red_k}(b)$ and $b \in \mathsf{SN}$, then $b \models \alpha$.

It is not hard to see that if X is a saturated Λ-set, X's realizers form a saturated set. We also identify a class of isomorphisms between Λ-sets:

Definition 7.3. Let X and Y be two Λ-sets. A Λ-*iso* p from X to Y is a bijective function $p : X_0 \to Y_0$ such that $a \models_X \alpha$ iff $a \models_Y p(\alpha)$.

As we suggested earlier, types will be modeled by Λ-sets. Unsurprisingly, then, sorts will be modeled by collections of Λ-sets. We introduce some additional structure in these collections to deal with the circularity in some pure type systems. In the definitions to follow we have some fixed set $\mathfrak{E}$ which will index a family of equivalence relations. We will instantiate $\mathfrak{E}$ when we give models for particular theories.

Definition 7.4. An $\mathfrak{E}$-*set* $\mathfrak{A}$ is a set of Λ-sets that is paired with two families of equivalence

$$
\begin{array}{lll}
A,\ B,\ a,\ b & ::= & s \mid x \mid (x:A) \to B \mid \lambda_{(x:A)\to B}.b \mid \mathsf{app}_{(x:A)\to B}(a,b) \\
\Gamma & ::= & \cdot \mid \Gamma, x\!:\!A
\end{array}
$$

$\boxed{a \rightsquigarrow_t b}$

$$
\frac{}{\mathsf{app}_{(x:A)\to B}((\lambda_{(x:A)\to B}.b), a) \rightsquigarrow_t [a/x]b} \quad \textsc{TSBeta}
$$

plus contextual rules, including reduction in the type annotations

$\boxed{a \rightsquigarrow_t^* b}$

$$
\frac{}{a \rightsquigarrow_t^* a} \quad \textsc{MTSRefl} \qquad \frac{\begin{array}{c} a_1 \rightsquigarrow_t a_2 \\ a_2 \rightsquigarrow_t^* a_3 \end{array}}{a_1 \rightsquigarrow_t^* a_3} \quad \textsc{MTSStep}
$$

$\boxed{\Gamma \vdash_t a : A}$

$$
\frac{\begin{array}{l} \vdash_t \Gamma \\ (s_1, s_2) \in \mathcal{A} \end{array}}{\Gamma \vdash_t s_1 : s_2} \quad \textsc{TTSort} \qquad \frac{\begin{array}{l} \vdash_t \Gamma \\ (x : A) \in \Gamma \end{array}}{\Gamma \vdash_t x : A} \quad \textsc{TTVar}
$$

$$
\frac{\begin{array}{ccc} \Gamma \vdash_t a : A & & \Gamma \vdash_t B : s \\ A \rightsquigarrow_t^* B & \text{or} & B \rightsquigarrow_t^* A \end{array}}{\Gamma \vdash_t a : B} \quad \textsc{TTConv} \qquad \frac{\begin{array}{l} \Gamma \vdash_t A : s_1 \\ \Gamma, x\!:\!A \vdash_t B : s_2 \\ (s_1, s_2, s_3) \in \mathcal{R} \end{array}}{\Gamma \vdash_t (x : A) \to B : s_3} \quad \textsc{TTPi}
$$

$$
\frac{\begin{array}{l} \Gamma \vdash_t (x : A) \to B : s \\ \Gamma, x\!:\!A \vdash_t b : B \end{array}}{\Gamma \vdash_t \lambda_{(x:A)\to B}.b : (x : A) \to B} \quad \textsc{TTLam} \qquad \frac{\begin{array}{l} \Gamma \vdash_t a : (x : A) \to B \\ \Gamma \vdash_t b : A \end{array}}{\Gamma \vdash_t \mathsf{app}_{(x:A)\to B}(a, b) : [b/x]B} \quad \textsc{TTApp}
$$

$\boxed{\vdash_t \Gamma}$

$$
\frac{}{\vdash_t \cdot} \quad \textsc{TCNil} \qquad \frac{\begin{array}{l} x \notin \mathsf{dom}(\Gamma) \\ \Gamma \vdash_t A : s \end{array}}{\vdash_t \Gamma, x\!:\!A} \quad \textsc{TCCons}
$$

Figure 2: Definition of PTS with tight reduction

relations, indexed by $i \in \mathfrak{E}$:

$$\cdot \cong^i_{\mathfrak{A}} \cdot \subseteq \mathfrak{A}^2 \qquad \text{and} \qquad \cdot \equiv^i_{\mathfrak{A}} \cdot \subseteq (\bigcup_{X \in \mathfrak{A}} X_0)^2$$

Here, $\cong^i_{\mathfrak{A}}$ equates some of the Λ-sets in $\mathfrak{A}$, and $\equiv^i_{\mathfrak{A}}$ equates some of their carriers. As we will see, it would be hard to interpret types formed using CC's rule $(\square, \square, \square)$ inside of set theory without breaking up the $\mathfrak{E}$-set associated with $\square$ into equivalence classes.

We can now lift the notion of products to Λ-sets. When interpreting a type $(x : A) \to B$, we will have a Λ-set X for A and a family of Λ-sets for B, one for each carrier of X. The following construction defines a new corresponding Λ-set.

Definition 7.5 (Λ-set products)**.** Let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be $\mathfrak{E}$-sets. Suppose $X \in \mathfrak{A}_1$ and $Y_\alpha \in \mathfrak{A}_2$ for each $\alpha \sqsubset X$. We define a new Λ-set $\Pi(X, Y)$ by:

$$\Pi(X, Y)_0 \quad := \quad \{f : (\alpha \in X_0) \to (Y_\alpha)_0 \,|\, \forall \alpha, \alpha' \in X_0, \forall i \in \mathfrak{E}, \, \alpha \equiv^i_{\mathfrak{A}_1} \alpha' \Rightarrow f(\alpha) \equiv^i_{\mathfrak{A}_2} f(\alpha')\}$$

$$a \models_{\Pi(X, Y)} f \quad \text{iff} \quad \forall \alpha \in X_0, \forall b \models_X \alpha, \forall A, B \in \mathsf{SN}, \, \mathsf{app}_{(x:A)\to B}(a, b) \models_{Y_\alpha} f(\alpha)$$

The carriers of this new Λ-set are set-theoretic functions which take any carrier $\alpha \sqsubset X$ to a carrier of Y_α. These functions must map carriers related in $\mathfrak{A}_1$ to carriers related in $\mathfrak{A}_2$. Intuitively, a term a realizes such a function f when any application of f is realized by the corresponding applications of a.

Our last definition in this section extends the equivalence relations of two $\mathfrak{E}$-sets to the products between them. The definition is somewhat intricate and can be skipped on a first reading.

Definition 7.6. Let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be $\mathfrak{E}$-sets. Suppose we have $X, X' \in \mathfrak{A}_1$ and two corresponding families of Λ-sets, $Y_\alpha, Y'_{\alpha'} \in \mathfrak{A}_2$ for each $\alpha \in X$ and $\alpha' \in X'$. For each $i \in \mathfrak{E}$, we define two new relations:

- $\Pi(X, Y) \cong^i_{\Pi(\mathfrak{A}_1, \mathfrak{A}_2)} \Pi(X', Y')$ iff:

$$X \cong^i_{\mathfrak{A}_1} X' \qquad \text{and} \qquad \forall (\alpha, \alpha') \in X_0 \times X'_0, \; \alpha \equiv^i_{\mathfrak{A}_1} \alpha' \Rightarrow Y_\alpha \cong^i_{\mathfrak{A}_2} Y'_{\alpha'}$$

- Suppose $\Pi(X, Y) \cong^i_{\Pi(\mathfrak{A}_1, \mathfrak{A}_2)} \Pi(X', Y')$. When $f \sqsubset \Pi(X, Y)$ and $g \sqsubset \Pi(X', Y')$ we define $f \equiv^i_{\Pi(\mathfrak{A}_1, \mathfrak{A}_2)} g$ iff:

$$\forall (\alpha, \alpha') \in X_0 \times X'_0, \; \alpha \equiv^i_{\mathfrak{A}_1} \alpha' \Rightarrow f(\alpha) \equiv^i_{\mathfrak{A}_2} g(\alpha')$$

7.3 A problem

As mentioned in the introduction, there are several problems with this proof. The first comes from the definition of Λ-set products. The authors claim to prove that this operation preserves saturation:

Let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be $\mathfrak{E}$-sets. Suppose $X \in \mathfrak{A}_1$ and $Y_\alpha \in \mathfrak{A}_2$ for each $\alpha \sqsubset X$. If X and each Y_α are saturated Λ-sets, then so is $\Pi(X, Y)$.

While a similar result holds for for saturated sets, this proposition is false. A saturated Λ-set must have a carrier which realizes every element of $\mathsf{BASE} \cap \mathsf{SN}$, but $\Pi(X, Y)$ may have no carriers at all.

As an example, consider three Λ-sets, X, Y_β and Y_γ, such that X has two carriers and the others have one, and each carrier is realized by every strongly normalizing expression. That is:

$$\begin{aligned} X &:= \ (\{\beta,\gamma\},\, \mathsf{SN} \times \{\beta,\gamma\}) \\ Y_\beta &:= \ (\{\beta'\},\, \mathsf{SN} \times \{\beta'\}) \\ Y_\gamma &:= \ (\{\gamma'\},\, \mathsf{SN} \times \{\gamma'\}) \end{aligned}$$

These Λ-sets are saturated. Suppose $\mathfrak{E}$ is a singleton set $\{1\}$, and define two $\mathfrak{E}$-sets:

$$\begin{array}{lcllcl} \mathfrak{A}_1 & := & \{X\} & \qquad \mathfrak{A}_2 & := & \{Y_\beta,\, Y_\gamma\} \\ \cong^1_{\mathfrak{A}_1} & := & \{(X, X)\} & \qquad \cong^1_{\mathfrak{A}_2} & := & \mathfrak{A}_2 \times \mathfrak{A}_2 \\ \equiv^1_{\mathfrak{A}_1} & := & X_0 \times X_0 & \qquad \equiv^1_{\mathfrak{A}_2} & := & \{(\beta',\beta'), (\gamma',\gamma')\} \end{array}$$

Notice in particular that β and γ are related by $\equiv^1_{\mathfrak{A}_1}$ but that β' and γ' are not related by $\equiv^1_{\mathfrak{A}_2}$. Any carrier of $\Pi(X, Y)$ would have to map β to β' and γ to γ', so no carrier can preserve the equivalence relation. Thus, $\Pi(X, Y)$ has no carriers and is not saturated. The (non-)proof given in the paper misses this problem because it neglects to reason carefully about which functions preserve the equivalence relation.

We could fix the example given here by demanding that if two carriers of elements of a $\mathfrak{E}$-set realize the same expressions, they must be related by $\equiv^i_{\mathfrak{A}}$. However, it would still be possible to construct a similar counter example by picking a smaller set of realizers for γ'.

One can imagine more complicated restrictions on Λ-sets and $\mathfrak{E}$-sets which restore this property, but it is not clear how they would influence the rest of the proof. It is also possible that all the specific uses of the product construction later in the paper result in saturated Λ-sets. However, because this regularity property is implicitly relied on in countless places, tracking it completely is beyond the scope of this survey.

7.4 Models

We will now describe the model into which we interpret a labeled PTS. This will be followed by four conditions, parameterized by the sets $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$. The main result of the paper is that when there exists a model satisfying the four conditions instantiated with parameters corresponding to a particular PTS, that system is strongly normalizing.

The four conditions are somewhat involved. However, when using a model where the set $\mathfrak{E}$ is empty, conditions 2-4 are trivially satisfied. This is the case for System F, so we recommend beginning by understanding condition 1 and the model of System F in the next section. Then Section 7.6 explains why this construction does not suffice for CC, which may help motivate the remaining conditions.

For each sort $s \in \mathcal{S}$, a model consists of

- an $\mathfrak{E}$-set $\mathfrak{A}^{\Uparrow}(s)$,
- a saturated Λ-set $\mathfrak{A}_{\Downarrow}(s)$, and
- a bijection $\Uparrow_s\colon \mathfrak{A}_{\Downarrow}(s)_0 \to \mathfrak{A}^{\Uparrow}(s)$.

When $\Gamma \vdash A : s$, we intend for $\mathfrak{A}^{\Uparrow}(s)$ to contain a model of A as a "type" and for $\mathfrak{A}_{\Downarrow}(s)$ to contain a model of A as a "term". The lifting function $\Uparrow_s$ relates these two interpretations: with each carrier of $\mathfrak{A}_{\Downarrow}(s)$ we associate the Λ-set that models its realizers as a type. We will also use the inverse of this function, which we write $\Downarrow_s\colon \mathfrak{A}^{\Uparrow}(s) \to \mathfrak{A}_{\Downarrow}(s)_0$.

The interpretation in Section 7.8 comes in two corresponding levels. Each type classified by a sort s has a *type* interpretation as an element of $\mathfrak{A}^{\Uparrow}(s)$. Every well-typed expression also has a *term* interpretation as a carrier of the Λ-set associated with its type. Finally, each expression will realize its term interpretation. By condition 1.1 below, this will imply the expression is strongly normalizing.

Condition 1: Uniformity of the universe hierarchy

The following three properties ensure that the $\mathfrak{E}$-sets and Λ-sets corresponding to sorts have a regular internal structure and that there are relationships among them corresponding to the sets $\mathcal{A}$ and $\mathcal{R}$.

(1.1) For each sort s, the elements of $\mathfrak{A}^{\Uparrow}(s)$ are saturated Λ-sets and the carriers of $\mathfrak{A}_{\Downarrow}(s)$ are each realized by every strongly normalizing expression.

(1.2) If $(s_1, s_2) \in \mathcal{A}$, then $\mathfrak{A}_{\Downarrow}(s_1) \in \mathfrak{A}^{\Uparrow}(s_2)$.

(1.3) Suppose $(s_1, s_2, s_3) \in \mathcal{R}$ and that we have Λ-sets $X \in \mathfrak{A}^{\Uparrow}(s_1)$ and $Y_\alpha \in \mathfrak{A}^{\Uparrow}(s_2)$ for each $\alpha \in X$. If:
$$\forall \alpha, \alpha' \sqsubset X, \forall i \in \mathfrak{E},\ \alpha \equiv^i_{\mathfrak{A}^{\Uparrow}(s_1)} \alpha' \Rightarrow Y_\alpha \cong^i_{\mathfrak{A}^{\Uparrow}(s_2)} Y_{\alpha'}$$
Then there is a Λ-set $\Pi_{\downarrow}(X, Y) \in \mathfrak{A}^{\Uparrow}(s_3)$ and a Λ-iso $\downarrow_{\Pi(X,Y)}\colon \Pi(X, Y) \to \Pi_{\downarrow}(X, Y)$.

While the first two sub-conditions are straightforward, some intuition is helpful for the third. If $(s_1, s_2, s_3) \in \mathcal{R}$, then the PTS allows function types whose domain is classified by s_1 and range by s_2. These function types can themselves be classified by sort s_3. Correspondingly, the condition says that when we may form a Λ-set product in the model from $\mathfrak{A}^{\Uparrow}(s_1)$ to $\mathfrak{A}^{\Uparrow}(s_2)$, there must be an isomorphic Λ-set in $\mathfrak{A}^{\Uparrow}(s_3)$. As we will see in Section 7.6, the equivalence relation condition on the Λ-set product restricts our attention to certain well-formed constructions to cope with the size mismatch between set-theoretic function spaces and PTS functions.

The remaining three conditions impose regularity constraints on the equivalence relations. These are essentially just sanity checks, and because we are not covering the details of the proofs we will not fully discuss how they are used. Roughly, conditions 2.1 and 3 check that when two Λ-sets or carriers are related by $\cong^i_{\mathfrak{A}}$ or $\equiv^i_{\mathfrak{A}}$, applying various operations to them preserves the relation. Conditions 2.2 and 4 check that when carriers appear in the model of more than one sort, they are treated uniformly.

Condition 2: Uniformity of $\cong^i_{\mathfrak{A}}$ and $\equiv^i_{\mathfrak{A}}$

(2.1) Suppose $X, X' \in \mathfrak{A}^{\Uparrow}(s_1)$ and $\mathfrak{A}_{\Downarrow}(s_1) \in \mathfrak{A}^{\Uparrow}(s_2)$. Then, for any $i \in \mathfrak{E}$:

$$X \cong^i_{\mathfrak{A}^{\Uparrow}(s_1)} X' \qquad \text{iff} \qquad \Downarrow_{s_1}(X) \equiv^i_{\mathfrak{A}^{\Uparrow}(s_2)} \Downarrow_{s_1}(X')$$

(2.2) Suppose $X_1, X_1' \in \mathfrak{A}^{\Uparrow}(s_1)$ and $X_2, X_2' \in \mathfrak{A}^{\Uparrow}(s_2)$ such that α is a carrier of both X_1 and X_2 and α' is a carrier of both X_1' and X_2'. Then, for any $i \in \mathfrak{E}$:

$$\alpha \equiv^i_{\mathfrak{A}^{\Uparrow}(s_1)} \alpha' \qquad \text{iff} \qquad \alpha \equiv^i_{\mathfrak{A}^{\Uparrow}(s_2)} \alpha'$$

Condition 3: Uniformity of $\Pi_{\downarrow}(X, Y)$ and $\downarrow_{\Pi(X,Y)}$

Suppose $(s_1, s_2, s_3) \in \mathcal{R}$. Let Λ-sets $X, X' \in \mathfrak{A}^{\Uparrow}(s_1)$ and families of Λ-sets $Y_\alpha, Y'_{\alpha'} \in \mathfrak{A}^{\Uparrow}(s_2)$ for each $(\alpha, \alpha') \in X_0 \times X_0'$ be given such that both families satisfy the hypothesis of (1.3). For each $i \in \mathfrak{E}$:

(3.1) If $\Pi(X, Y) \cong^i_{\Pi(\mathfrak{A}^{\Uparrow}(s_1), \mathfrak{A}^{\Uparrow}(s_2))} \Pi(X', Y')$ then $\Pi_{\downarrow}(X, Y) \cong^i_{\mathfrak{A}^{\Uparrow}(s_3)} \Pi_{\downarrow}(X', Y')$.

(3.2) Suppose $\Pi(X, Y) \cong^i_{\Pi(\mathfrak{A}^{\Uparrow}(s_1), \mathfrak{A}^{\Uparrow}(s_2))} \Pi(X', Y')$. Then for any $f \sqsubset \Pi(X, Y)$ and $g \sqsubset \Pi(X', Y')$:

$$f \equiv^i_{\Pi(\mathfrak{A}^{\Uparrow}(s_1), \mathfrak{A}^{\Uparrow}(s_2))} g \qquad \text{iff} \qquad \downarrow_{\Pi(X,Y)}(f) \equiv^i_{\mathfrak{A}^{\Uparrow}(s_3)} \downarrow_{\Pi(X',Y')}(g)$$

Condition 4: Uniformity of $\Uparrow_s$ and $\Downarrow_s$

(4.1) If $X \in \mathfrak{A}^{\Uparrow}(s_1)$ and $X \in \mathfrak{A}^{\Uparrow}(s_2)$ then $\Downarrow_{s_1}(X) = \Downarrow_{s_2}(X)$.

(4.2) If $\alpha \sqsubset \mathfrak{A}_{\Downarrow}(s_1)$ and $\alpha \sqsubset \mathfrak{A}_{\Downarrow}(s_2)$ then $\Uparrow_{s_1}(\alpha) = \Uparrow_{s_2}(\alpha)$.

These two conditions indicate that the sort subscripts on the lifting operation and its inverse are only annotations; they do not influence the behavior of the functions.

7.5 A model for System F

We will use two simple Λ-set constructions in the model for System F.

Definition 7.7. A Λ-set X is *degenerate* if $X_0 = \{S\}$ where S is a saturated set and $a \models_X S$ iff $a \in S$. We refer to S as the *underlying set* of X, and write DG for the set of all degenerate Λ-sets.

Definition 7.8. When X is any non-empty set, we define an associated Λ-set $J(X)$ whose carrier set is X. Each element of X is realized by every strongly normalizing expression.

Recall that System F is given by:

$$\mathcal{S} = \{*, \Box\} \qquad \mathcal{A} = \{(*, \Box)\} \qquad \mathcal{R} = \{(*, *, *), (\Box, *, *)\}$$

In the model of System F, we pick $\mathfrak{E} = \emptyset$. So, we will not have to define any of the $\mathfrak{E}$-set relations. We pick

$$\mathfrak{A}^{\Uparrow}(*) = \mathsf{DG} \qquad \mathfrak{A}^{\Uparrow}(\Box) = \{J(\mathsf{DG})\}$$

and, for each sort s, set $\mathfrak{A}_{\Downarrow}(s) = J(\mathfrak{A}^{\Uparrow}(s))$. The bijection $\Uparrow_s\colon \mathfrak{A}_{\Downarrow}(s)_0 \to \mathfrak{A}^{\Uparrow}(s)$ is then simply the identity function.

We must verify that this model has the appropriate properties. Of these, conditions 2-4 are vacuous because $\mathfrak{E}$ is empty. Conditions 1.1 and 1.2 are apparent. It only remains to check 1.3.

Proof. Suppose that $(s_1, s_2, s_3) \in \mathcal{R}$ and we have Λ-sets $X \in \mathfrak{A}^{\Uparrow}(s_1)$ and $Y_\alpha \in \mathfrak{A}^{\Uparrow}(s_2)$ for each $\alpha \sqsubset X$. Because $s_2 = s_3 = *$, each Y_α is a degenerate Λ-set.

The carriers of $\Pi(X, Y)$ are the functions which map each $\alpha \sqsubset X$ to a carrier of Y_α. But because each Y_α has just one carrier, there is only one such function f.

We must pick a degenerate Λ-set for $\Pi_\downarrow(X, Y)$, so pick the one whose underlying set is the realizers of f in $\Pi(X, Y)$.[2] Then the Λ-iso $\downarrow_{\Pi(X,Y)}$ simply maps the only carrier of $\Pi(X, Y)$ to the only carrier of $\Pi_\downarrow(X, Y)$ and trivially satisfies the condition that realizers are preserved. $\Box$

7.6 Does this model work for CC?

It is instructive to consider why the model selected for System F is not sufficient for CC. This will motivate the use of $\mathfrak{E}$-sets and their equivalence relations. Recall that CC is the PTS given by:

$$\mathcal{S} = \{*, \Box\} \qquad \mathcal{A} = \{(*, \Box)\} \qquad \mathcal{R} = \{(*, *, *),\ (*, \Box, \Box),\ (\Box, *, *),\ (\Box, \Box, \Box)\}$$

[2] Here we are implicitly relying on the problematic lemma from Section 7.3. However, expanding the definitions when $s_1 = *$ or $s_1 = \Box$ reveals that the constructions remain saturated in this particular case.

The only difference from System F is the addition of two new rules. We find the problem in satisfying property 1.3 for the rule $(\Box, \Box, \Box)$.

Suppose $X \in \mathfrak{A}^{\Uparrow}(\Box)$ and $Y_\alpha \in \mathfrak{A}^{\Uparrow}(\Box)$ for each $\alpha \sqsubset X$. Referring back to the definition of $\mathfrak{A}^{\Uparrow}(\Box)$ for our model, we see this means X and the Y_αs are all $J(\mathsf{DG})$.

Condition 1.3 demands that we find an element of $\mathfrak{A}^{\Uparrow}(\Box)$ which is isomorphic to $\Pi(X, Y)$. The only such element is $J(\mathsf{DG})$, so we are being asked to show $J(\mathsf{DG})$ is isomorphic to $\Pi(X, Y)$. But the carrier set of $J(\mathsf{DG})$ is DG, and the carrier set of $\Pi(X, Y)$ is all the set theoretic functions from DG to DG. Plainly, there is no bijection between these sets.

This problem is fundamental, since the axiom $(*, \Box)$ means $\mathfrak{A}^{\Uparrow}(\Box)$ must contain $\mathfrak{A}_{\Downarrow}(*)$. To resolve it, we use a non-empty $\mathfrak{E}$. The equivalence relations will break our model of $\Box$ up into an infinite hierarchy of levels. In situations like this where self-reference and set-theoretic size constraints are a problem, we will simply move to a "higher" level.

7.7 A model for CC

We pick $\mathfrak{A}^{\Uparrow}(*) = \mathsf{DG}$ and $\mathfrak{A}_{\Downarrow}(s) = J(\mathfrak{A}^{\Uparrow}(s))$ for each s as we did in the model of System F. Correspondingly, each $\Uparrow_s$ is the identity function. The definition of $\mathfrak{A}^{\Uparrow}(\Box)$ is broken into levels. Define:

- $\mathsf{lev}_1 = \{J(\mathsf{DG})\}$
- For each $n \in \mathbb{N}^+$,
$$\mathsf{lev}_{\leq n} = \bigcup_{1 \leq k \leq n} \mathsf{lev}_k$$
- For each $n \in \mathbb{N}^+$
$$\begin{array}{rcll} \mathsf{lev}_{n+1} & = & \{\Pi(X, Y) \mid X \in \mathfrak{A}^{\Uparrow}(*), & Y_\alpha \in \mathsf{lev}_n \text{ for each } \alpha \sqsubset X\} \\ & \cup & \{\Pi(X, Y) \mid X \in \mathsf{lev}_n, & Y_\alpha \in \mathsf{lev}_{\leq n} \text{ for each } \alpha \sqsubset X\} \\ & \cup & \{\Pi(X, Y) \mid X \in \mathsf{lev}_{\leq n}, & Y_\alpha \in \mathsf{lev}_n \text{ for each } \alpha \sqsubset X\} \end{array}$$

Then $\mathfrak{A}^{\Uparrow}(\Box)$ is the collection of all these levels:

$$\mathfrak{A}^{\Uparrow}(\Box) = \bigcup_{n \in \mathbb{N}^+} \mathsf{lev}_n$$

Notice that all the levels are disjoint. Each level collects the Λ-set products whose domain or range is formed from elements of the previous level.

Finally we define the equivalence relations for our $\mathfrak{E}$-sets. In this model, we make $\mathfrak{E}$ a singleton set $\{1\}$, so there are two relations to define for each sort.

- $\cong^1_{\mathfrak{A}^{\Uparrow}(*)}$ relates every two elements of $\mathfrak{A}^{\Uparrow}(*)$.
- $\equiv^1_{\mathfrak{A}^{\Uparrow}(*)}$ relates every two carriers of every two elements of $\mathfrak{A}^{\Uparrow}(*)$.

- $\cong^1_{\mathfrak{A}^\Uparrow(\Box)}$ relates two elements of $\mathfrak{A}^\Uparrow(\Box)$ iff they are in the same lev_n.
- $\equiv^1_{\mathfrak{A}^\Uparrow(\Box)}$ relates every two carriers of every two elements of $\mathfrak{A}^\Uparrow(\Box)$.

These constructions satisfy conditions 1.1 and 1.2 as in the model for System F.[3] The same proof of condition 1.3 for rules $(*, *, *)$ and $(\Box, *, *)$ also applies. We now consider how this model solves the problem we encountered with rules $(*, \Box, \Box)$ and $(\Box, \Box, \Box)$ in the previous section.

Proof. Suppose $X \in \mathfrak{A}^\Uparrow(s)$ for some s and $Y_\alpha \in \mathfrak{A}^\Uparrow(\Box)$ for each $\alpha \sqsubset X$. Suppose also that:

$$\forall \alpha, \alpha' \sqsubset X, \alpha \equiv^1_{\mathfrak{A}^\Uparrow(s_1)} \alpha' \ \Rightarrow\ Y_\alpha \cong^1_{\mathfrak{A}^\Uparrow(\Box)} Y_{\alpha'}$$

Because the relations $\equiv^1_{\mathfrak{A}^\Uparrow(s)}$ equate any two carriers, this implies that all of the Y_αs are related by $\cong^1_{\mathfrak{A}^\Uparrow(\Box)}$. This means they are all in the same set lev_n for some n. Define the natural number m to be 1 if $s = *$ and otherwise to be the m such that $X \in \mathsf{lev}_m$. Then $\Pi(X, Y) \in \mathsf{lev}_{\max(n,m)+1}$. So we pick $\Pi(X, Y)$ itself for $\Pi_\downarrow(X, Y)$ and the identity function for $\downarrow_{\Pi(X,Y)}$. $\Box$

We check each of the remaining conditions. Of these, 3.1 and 3.2 are somewhat involved and the rest are easy.

(2.1) The only case where $\mathfrak{A}_\Downarrow(s_1) \in \mathfrak{A}^\Uparrow(s_2)$ is when s_1 is $*$ and s_2 is $\Box$. The required condition follows easily since $\cong^1_{\mathfrak{A}^\Uparrow(s_1)}$ relates every two elements of $\mathfrak{A}^\Uparrow(s_1)$ and $\equiv^1_{\mathfrak{A}^\Uparrow(s_2)}$ relates every two carriers of elements of $\mathfrak{A}^\Uparrow(s_2)$.

(2.2) The carriers of elements of $\mathfrak{A}^\Uparrow(*)$ are all expressions. The carriers of elements of $\mathfrak{A}^\Uparrow(\Box)$ are all Λ-sets. Thus we must have $s_1 = s_2$ and the condition is trivial.

(3.1) When $s_2 = s_3 = *$ this condition is trivial because $\cong^i_{\mathfrak{A}^\Uparrow(*)}$ relates every two elements of $\mathfrak{A}^\Uparrow(*)$. Otherwise suppose the Λ-sets $X, X' \in \mathfrak{A}^\Uparrow(s_1)$ and the families of Λ-sets $Y_\alpha, Y'_{\alpha'} \in \mathfrak{A}^\Uparrow(\Box)$ are given as specified, so that:

$$\Pi(X, Y) \cong^1_{\Pi(\mathfrak{A}^\Uparrow(s_1), \mathfrak{A}^\Uparrow(\Box))} \Pi(X', Y')$$

We must show that $\Pi_\downarrow(X, Y)$ and $\Pi_\downarrow(X', Y')$ occupy the same lev_n. Recall that $\Pi_\downarrow(X, Y) = \Pi(X, Y)$ and $\Pi_\downarrow(X', Y') = \Pi(X', Y')$. The result then follows from the definition of $\cong^1_{\Pi(\mathfrak{A}^\Uparrow(s_1), \mathfrak{A}^\Uparrow(s_2))}$, which checks that X and X' are on the same level and each pair Y_α and $Y_{\alpha'}$ are on the same level.

(3.2) Suppose $(s_1, s_2, s_3) \in \mathcal{R}$ and $\Pi(X, Y) \cong^1_{\Pi(\mathfrak{A}^\Uparrow(s_1), \mathfrak{A}^\Uparrow(s_2))} \Pi(X', Y')$. Let $f \sqsubset \Pi(X, Y)$ and $g \sqsubset \Pi(X', Y')$ be given. Since $\equiv^1_{\mathfrak{A}^\Uparrow(s_3)}$ always relates any two carriers of $\mathfrak{A}^\Uparrow(s_3)$, we must show $f \equiv^1_{\Pi(\mathfrak{A}^\Uparrow(s_1), \mathfrak{A}^\Uparrow(s_2))} g$.

[3] This is another implicit use of the mistaken theorem discussed in Section 7.3. However, one can verify that $\mathfrak{A}^\Uparrow(\Box)$ here does contain only saturated Λ-sets. The situation is less clear in the paper's model of ECC, where the relations $\equiv^i_{\mathfrak{A}^\Uparrow(s)}$ are more complicated.

This follows immediately by the definition of $\equiv^1_{\Pi(\mathfrak{A}^{\Uparrow}(s_1),\,\mathfrak{A}^{\Uparrow}(s_2))}$ and the observation that $\equiv^1_{\mathfrak{A}^{\Uparrow}(s_2)}$ also relates any two carriers of $\mathfrak{A}^{\Uparrow}(s_2)$ (and thus any two elements in the in the range of f and g).

(4) The models of the sorts are disjoint, so this condition is trivial.

7.8 The interpretation

The interpretations of types and terms should not be very surprising. As we have mentioned, the interpretation of a type will be a Λ-set and the interpretation of a term will be a carrier. The soundness theorem will say that every well-typed expression realizes its term interpretation.

We define three functions by mutual recursion: $[\![\Gamma]\!]$, $[\![\Gamma \vdash_t a]\!]_{(\gamma)}$ and $[\Gamma \vdash_t a]_{(\gamma)}$. The soundness theorem will show that the first is defined whenever $\vdash_t \Gamma$ and the latter two whenever $\Gamma \vdash_t a : A$ and $\gamma \in [\![\Gamma]\!]$. In general γ will be an n-tuple of pairs associating variables from the context with carriers. We write $\gamma(x)$ for the carrier associated with x, when it exists.

$$[\![\cdot]\!] := \emptyset \tag{1}$$
$$[\![\Gamma, x\!:\!A]\!] := \{(\gamma,(x,\alpha)) \mid \gamma \in [\![\Gamma]\!] \wedge \alpha \sqsubset [\![\Gamma \vdash_t A]\!]_{(\gamma)}\} \tag{2}$$

$$[\![\Gamma \vdash_t s]\!]_{(\gamma)} := \mathfrak{A}_{\Downarrow}(s) \tag{3}$$
$$[\![\Gamma \vdash_t (x:A) \to B]\!]_{(\gamma)} := \Pi_{\downarrow}([\![\Gamma \vdash_t A]\!]_{(\gamma)},\, [\![\Gamma, x\!:\!A \vdash_t B]\!]_{(\gamma,\text{-})}) \tag{4}$$
$$[\![\Gamma \vdash_t A]\!]_{(\gamma)} := \Uparrow_s ([\Gamma \vdash_t A]_{(\gamma)}) \tag{5}$$

$$[\Gamma \vdash_t x]_{(\gamma)} := \gamma(x) \tag{6}$$
$$[\Gamma \vdash_t \lambda_{(x:A)\to B}.b]_{(\gamma)} := \downarrow_{\Pi([\![\Gamma\vdash_t A]\!]_{(\gamma)},\,[\![\Gamma,x A\vdash_t B]\!]_{(\gamma,\text{-})})} ([\Gamma, x\!:\!A \vdash_t b]_{(\gamma,\text{-})}) \tag{7}$$
$$[\Gamma \vdash_t \mathsf{app}_{(x:A)\to B}(a,b)]_{(\gamma)} := (\uparrow_{\Pi([\![\Gamma\vdash_t A]\!]_{(\gamma)},\,[\![\Gamma,x A\vdash_t B]\!]_{(\gamma,\text{-})})} ([\Gamma \vdash_t a]_{(\gamma)}))\ ([\Gamma \vdash_t b]_{(\gamma)}) \tag{8}$$
$$[\Gamma \vdash_t a]_{(\gamma)} := \Downarrow_s ([\![\Gamma \vdash_t a]\!]_{(\gamma)}) \tag{9}$$

There are a few notational infelicities to explain. First, equations 5 and 9 are meant to apply only when the previous clauses in the respective definitions do not. The remark after the definition of condition 4 in Section 7.4 justifies the use of an arbitrary s in these cases. By $\uparrow_{\Pi([\![\Gamma\vdash_t A]\!]_{(\gamma)},\,[\![\Gamma,x A\vdash_t B]\!]_{(\gamma,\text{-})})}$ in equation 8 we mean the inverse of $\downarrow_{\Pi([\![\Gamma\vdash_t A]\!]_{(\gamma)},\,[\![\Gamma,x A\vdash_t B]\!]_{(\gamma,\text{-})})}$. Finally, throughout the definition we wrote $[\![\Gamma, x\!:\!A \vdash_t B]\!]_{(\gamma,\text{-})}$ and $[\Gamma, x\!:\!A \vdash_t b]_{(\gamma,\text{-})}$ for the functions

$$\alpha \sqsubset [\![\Gamma \vdash_t A]\!]_{(\gamma)} \mapsto [\![\Gamma, x\!:\!A \vdash_t B]\!]_{(\gamma,(x,\alpha))}$$

and

$$\alpha \sqsubset [\![\Gamma \vdash_t A]\!]_{(\gamma)} \mapsto [\Gamma, x\!:\!A \vdash_t b]_{(\gamma,(x,\alpha))}$$

respectively. The first can be viewed as an indexed family of Λ-sets suitable for use as the second argument to a Λ-set product construction. The second is a carrier of products formed with the first.

This definition is more intimidating than the interpretation from the previous proof, but none of it is surprising. More, it has the advantage that the connection between the interpretations of types and terms is explicit as part of their definition. We describe each clause to help with the symbolic burdern:

- The definition of $\llbracket\Gamma\rrbracket$ says that when $\gamma \in \llbracket\Gamma\rrbracket$, each x in Γ should be paired with a carrier of the interpretation of its given type. This is essentially the definition of the $\sigma \models \Gamma$ judgement from the last section.

- The definition of $\llbracket\Gamma \vdash_t A\rrbracket_{(\gamma)}$ associates with each well-typed expression a Λ-set in the model (assuming the four conditions are met). Clause 3 says that when A is a sort s, we pick the Λ-set explicitly specified by the model ($\mathfrak{A}_{\Downarrow}(s)$).

  In clause 4, we consider a function type $(x : A) \to B$. The typing rule TTPI says this will be a valid function type in sort s_3 when A is in sort s_1, B is in sort s_2 and $(s_1, s_2, s_3) \in \mathcal{R}$. The Λ-set $\Pi(\llbracket\Gamma \vdash_t A\rrbracket_{(\gamma)}, \llbracket\Gamma, x : A \vdash_t B\rrbracket_{(\gamma,\text{-})})$ models the functions from the interpretation of A to the interpretation of B, but we do not know if it exists anywhere in our model. Luckily, condition 1.3 guarantees that an isomorphic Λ-set lives in $\mathfrak{A}^{\Uparrow}(s_3)$. We pick that one, since this is just where the interpretation of the function type belongs.

  The catch-all clause 5 handles expressions that look more like terms than types. In that case, we use the term interpretation and then lift the resulting carrier to a Λ-set with the provided $\Uparrow$ function. For example, if A is the application of a type function $a\,b$, the term interpretation will interpret a and apply the result to the term interpretation of b. There is a similar clause in the term interpretation that will call back to the type interpretation when it reaches subcomponents that looked more like types.

- The definition of $[\Gamma \vdash_t a]_{(\gamma)}$ associates the carrier of a Λ-set in the model with each well-typed expression. Here, clause 6 handles variables using the provided environment. The final default clause 9 mirrors clause 5 from the previous definition.

  Functions $\lambda_{(x:A)\to B}.b$ are interpreted by clause 7. Here, $[\Gamma, x : A \vdash_t b]_{(\gamma,\text{-})}$ is a function from carriers in the type interpretation of A to carriers in the type interpretation of B. Such functions are themselves carriers of $\Pi(\llbracket\Gamma \vdash_t A\rrbracket_{(\gamma)}, \llbracket\Gamma, x : A \vdash_t B\rrbracket_{(\gamma,\text{-})})$. As we saw in the discussion of clause 4, this Λ-set may not be in our model. So we use the function $\downarrow_{\Pi(\llbracket\Gamma\vdash_t A\rrbracket_{(\gamma)}, \llbracket\Gamma,x{:}A\vdash_t B\rrbracket_{(\gamma,\text{-})})}$, which condition 1.3 guarantees will return a corresponding carrier in the model.

  The case for applications is similar. The soundness theorem will show that, if the application $\mathsf{app}_{(x:A)\to B}(a, b)$ is well-typed, the interpretation of a will be a carrier of $\Pi_{\downarrow}(\llbracket\Gamma \vdash_t A\rrbracket_{(\gamma)}, \llbracket\Gamma, x : A \vdash_t B\rrbracket_{(\gamma,\text{-})})$. We use the provided Λ-iso to convert this to a carrier of $\Pi(\llbracket\Gamma \vdash_t A\rrbracket_{(\gamma)}, \llbracket\Gamma, x\!:\!A \vdash_t B\rrbracket_{(\gamma,\_)})$ so that applying it to the interpretation of b will yield a carrier in the type interpretation of B.

7.9 Strong Normalization

Strong normalization follows from two key theorems. The first says that when an expression is well-typed, its term interpretation is a carrier of the interpretation of its type. The second says that each expression realizes its term interpretation. Since condition 1.1 guarantees that all Λ-sets in the model are saturated, the realizers are all strongly normalizing.

Theorem 7.9 (Soundness)**.** If $\vdash_t \Gamma$, then $\llbracket\Gamma\rrbracket$ is well-defined. If $\Gamma \vdash_t a : A$ and $\gamma \in \llbracket\Gamma\rrbracket$ then:

- $[\Gamma \vdash_t a]_{(\gamma)}$ and $\llbracket\Gamma \vdash_t A\rrbracket_{(\gamma)}$ are well-defined, and $[\Gamma \vdash_t a]_{(\gamma)} \sqsubset \llbracket\Gamma \vdash_t A\rrbracket_{(\gamma)}$.
- If $A = s$, then $\llbracket\Gamma \vdash_t a\rrbracket_{(\gamma)}$ is well-defined and an element of $\mathfrak{A}^{\Uparrow}(s)$.

Theorem 7.10 (Self-realization)**.** Suppose $\Gamma \vdash_t a : A$ and $\Gamma = x_1 : B_1, \ldots, x_n : B_n$. Let $((x_1, \beta_1), \ldots, (x_n, \beta_n)) \in \llbracket\Gamma\rrbracket$ and expressions $b_1, \ldots, b_n$ be given such that for each $1 \leq i \leq n$:

$$b_i \models_{\llbracket x_1:B_1,\ldots,x_{i-1}:B_{i-1}\vdash_t B_i\rrbracket_{((x_1,\beta_1),\ldots,(x_{i-1},\beta_{i-1}))}} \beta_i$$

Then:

$$[b_1/x_1]\ldots[b_n/x_n]a \models_{\llbracket\Gamma\vdash_t A\rrbracket_{(\gamma)}} [\Gamma \vdash_t a]_{(\gamma)}$$

Strong normalization then follows just as it did in the previous proof. In Theorem 7.10, the type interpretation of each B_i has a carrier which is realized by any strongly normalizing base expression, and thus any variable. Pick that realizer for β_i and pick x_i for b_i. Then the substitutions in the conclusion of the theorem do nothing and we find that a realizes its own term interpretation. Since term interpretations are carriers in saturated Λ-sets, a is strongly normalizing.

We do not go into the details of the proofs of these theorems. Indeed, the proofs given seem inadequate. For example, consider the first lemma the paper gives after the interpretation:

$$[\Gamma \vdash_t \mathsf{app}_{(x:A)\to B}((\lambda_{(x:A)\to B}.a), b)]_{(\gamma)} = [\Gamma \vdash_t a]_{(\gamma,(x,b))}$$

The authors do not identify the hypotheses of this lemma. However, observe that the lambda term is interpreted as a function whose domain is the carriers of $\llbracket\Gamma \vdash_t A\rrbracket_{(\gamma)}$. Thus, if $[\Gamma \vdash_t b]_{(\gamma)}$ is not such a carrier, the left-hand side will not be defined while the right-hand side may be (for example, if x doesn't occur in a). The lemma is subsequently used in situations where we only know $\Gamma \vdash_t b : A$, and the soundness theorem itself would be needed to show this is enough. A much more careful proof of the soundness theorem is needed.

All that remains is to relate strong normalization for a labeled PTS to strong normalization for an ordinary PTS. To do this, define an operation $|a|$ on labeled expressions which simply erases the extra annotations to obtain an ordinary PTS expression. Extend this operation to contexts $|\Gamma|$ by applying it to each type. We would like to know that if $\Gamma \vdash a : b$ in an ordinary PTS, then there is a labeled context Γ' and labeled expressions a', b' such that $|\Gamma'| = \Gamma$, $|a'| = a$, $|b'| = b$ and $\Gamma' \vdash_t a' : b'$. Since a' is well-typed in the labeled PTS, all the type annotations on beta redexes must match, and thus that its normalization behavior corresponds to that of a.

The proof of this theorem is mostly straightforward by induction. The only problem comes in the case where the derivation used the conversion rule. Here we must show a relationship between labeled, tight conversion and ordinary PTS conversion. The paper shows that the two agree in the case of well-typed expressions:

Lemma 7.11. Suppose $\Gamma \vdash_t a : A$ and $\Gamma \vdash_t b : B$. If $|a| =_\beta |b|$ then $a =^t_\beta b$.

This is unsurprising, since in well-typed expressions, labeled or not, beta reductions only occur when the function's domain type agrees with the type of its argument. The authors give a detailed proof.

8 Discussion and Conclusion

The original goal of this project was to survey several very different strong normalization proofs for the calculus of constructions. To that end, we picked three attempts that target different structures (sets of expressions, realizability semantics, and F_ω). Each paper's proof was targeted toward different additional goals. The first considered extensions with various datatypes and recursion. The second tried to model a large class of pure type systems. The last gave a relatively straightforward translation to simpler system, demonstrating that the proof-theoretic complexity of CC's strong normalization argument is no greater than that of F_ω.

Despite this, the three proofs are remarkably similar. Each gives a type interpretation $[\![A]\!]$ followed by a term interpretation $[a]$. Then, when $\Gamma \vdash a : A$, a simple relationship between the two translations is demonstrated (for example, $[a] \in [\![A]\!]$). Finally, this relationship is shown to imply that the expression a itself is strongly normalizing. Though the models targeted by these functions are different in each proof, they share a considerable amount of structure. For example, saturated sets of expressions are very useful in both the first and second proofs. They are often used in proofs of strong normalization for F_ω as well.

There are many commonalities even in the specifics of the interpretations. Compare the kind interpretation V from Section 6.2 with V in Section 5.2. Though one targets collections saturated sets and the other F_ω, both cope with the kind structure of CC in the same way. Moreover, examining this similarity yields a cleaner understanding of both proofs. In the context of saturated sets, it is tricky to motivate the definition of the kind interpretation $V((x : A) \to B)$ when A is a type. For example, we might plausibly have picked functions from values or expressions to $V(B)$ instead of just $V(B)$ itself. But in the translation to F_ω we find a snappy explanation: this definition simply erases dependency. By unifying the presentations and providing a common narrative through the three proofs, we found additional clarity in each.

The papers have differences, too. These typically result from the motivations of the authors. For example, the translation to F_ω is certainly the simplest of the developments and provides the most confidence in the result. It achieves this by relying on the existing strong normalization result for the simpler system. On the downside, the authors do not consider extensions.

The most complicated proof uses realizability semantics. The structure of the $\mathfrak{C}$-sets is somewhat intimidating, and in the end the model of CC uses relatively little of its expressiveness. The authors aim to provide a technique which extends to ECC, but perhaps in this they overreach: the reasoning about the structures involved is sometimes mistaken, and it is not clear how simple it would be to repair.

The traditional approach using saturated sets falls somewhere between these two. Though this technique must be extended to cope with CC, the proof is somewhat familiar and does not require any new structures. The authors succeed in extending the approach to various common datatypes with recursion, but do not consider more complicated additions like a predicative hierarchy of universes or large eliminations.

The popularity of dependently-typed programming languages continues to grow. So, too, do their lists of features. Our understanding of their metatheory and of strong normalization in particular has not quite kept pace. Thus, while the strong normalization of CC has been considered a settled issue for more than two decades, understanding its fundamentals is more important now than ever.

References

[1] T. Altenkirch. *Constructions, Inductive Types and Strong Normalization.* PhD thesis, University of Edinburgh, November 1993.

[2] T. Altenkirch. Proving Strong Normalization of CC by Modifying Realizability Semantics. In *TYPES '93*, volume 806 of *LNCS*, pages 3–18, 1994.

[3] H. Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press, 1992.

[4] T. Coquand. *Une Théorie des Constructions.* PhD thesis, Université Paris VII, 1985.

[5] T. Coquand and J. Gallier. A Proof of Strong Normalization For the Theory of Constructions Using a Kripke-Like Interpretation. In *Informal Proceedings of the First Annual Workshop on Logical Frameworks*, 1990.

[6] J. Gallier. *On Girard's "Candidats De Reductibilité"*, pages 123–230. Logic and Computer Science. Academic Press, 1990.

[7] H. Geuvers. A short and flexible proof of Strong Normalization for the Calculus of Constructions. In *TYPES '94*, volume 996 of *LNCS*, pages 14–38, 1995.

[8] H. Geuvers and M.-J. Nederhof. Modular proof of strong normalization for the calculus of constructions. *Journal of Functional Programming*, 1(2):155–189, 1991.

[9] H. Goguen. *A Typed Operational Semantics for Type Theory*. PhD thesis, University of Edinburgh, 1994.

[10] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40:194–204, 1993.

[11] Z. Luo. *Computation and reasoning: a type theory for computer science*. Oxford University Press, New York, NY, USA, 1994.

[12] P. Martin-Löf. An intutionistic theory of types. Unpublished manuscrupt, 1971.

[13] P.-A. Melliès and B. Werner. A generic normalisation proof for pure type systems. In *TYPES*, pages 254–276, 1996.

[14] B. Nordström, K. Petersson, and J. M. Smith. Clarendon Press, Oxford, 1990.

[15] U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September 2007.

[16] The Agda Team. *The Agda Wiki*, 2010. Available at `http://wiki.portal.chalmers.se/agda/`.

[17] The Coq Development Team. *The Coq Proof Assistant Reference Manual, Version 8.3*. LogiCal Project, 2010. Available at `http://coq.inria.fr/V8.3/refman/`.

[18] B. Werner. *Une théorie des constructions inuductives*. PhD thesis, Université Paris VII, 1994.

A Details for Section 6

Theorem (Soundness of the interpretation)**.** Suppose $\Gamma \vdash a : A$ and $\sigma \models \rho : \Gamma$. Then $[a]_\rho \in [\![A]\!]_\sigma$.

Proof. We go by induction on the typing derivation $\mathcal{D}$

Case: $\mathcal{D} = \dfrac{\vdash \Gamma \qquad (s_1, s_2) \in \mathcal{A}}{\Gamma \vdash * : \square}$ TSORT

The only axiom is $(*, \Box)$. This case is immediate, as $[*]_\rho = * \in \mathsf{SN} = [\![\Box]\!]_\sigma$

Case: $\mathcal{D} = \dfrac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A}\ \textsc{TVar}$

By the assumption $\sigma \models \rho : \Gamma$, the rule's second premise implies that $[x]_\rho \in [\![A]\!]_\sigma$.

Case: $\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash A : s_1} \qquad \overset{\mathcal{D}_2}{\Gamma, x : A \vdash B : s_2} \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash (x : A) \to B : s_3}\ \textsc{TPi}$

We must show that $(x : [A]_\rho) \to [B]_\rho \in \mathsf{SN}$. By IH for $\mathcal{D}_1$, $[A]_\rho \in \mathsf{SN}$, so it only remains to show that $[B]_\rho \in \mathsf{SN}$.

The IH for $\mathcal{D}_2$ says that given any σ' and ρ' such that $\sigma' \models \rho' : \Gamma, x : A$, we have $\rho' B \in SN$. For σ', pick σ if s_1 is $*$, or extend it with a canonical inhabitant of $V(A)$ if s_1 is $\Box$. For ρ', pick $\rho[x \mapsto x]$. Then $\rho' x = x$, which is in $[\![A]\!]_\sigma$ since the latter must be a saturated set by lemma 6.13. We conclude that $[B]_\rho = [B]_{\rho'} \in \mathsf{SN}$ as desired.

Case: $\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash A : s} \qquad \overset{\mathcal{D}_2}{\Gamma, x : A \vdash b : B}}{\Gamma \vdash \lambda x : A.b : (x : A) \to B}\ \textsc{TLam}$

There are two subcases: s is either $*$ or $\Box$. In either case, the IH for $\mathcal{D}_1$ gives us that $[A]_\rho \in \mathsf{SN}$. Since x is a bound variable, we may pick it to be fresh for the domain and range of ρ.

– Suppose s is $*$. Then we must show $\lambda x : [A]_\rho.[b]_\rho \in \Pi([\![A]\!]_\sigma, [\![B]\!]_\sigma)$. So let $a \in [\![A]\!]_\sigma$ be given, and observe it is enough to show $(\lambda x : [A]_\rho.[b]_\rho)\ a \in [\![B]\!]_\sigma$.

  We have $\sigma \models \rho[x \mapsto a] : G, x : A$. Thus, the IH for $\mathcal{D}_2$ gives us $[b]_{\rho[x \mapsto a]} \in [\![B]\!]_\sigma$. But we also know:

$$(\lambda x : [A]_\rho.[b]_\rho)\ a \rightsquigarrow [a/x][b]_\rho = [b]_{\rho[x \mapsto a]}$$

  And this step contracts a key redex. So, it suffices to show that $(\lambda x : [A]_\rho.[b]_\rho)\ a \in \mathsf{SN}$.

  Lemma 6.13 gives us that $[\![A]\!]_\sigma$ is a saturated set, so $a \in \mathsf{SN}$. We already observed that $[A]_\rho \in \mathsf{SN}$. By the second IH, $[b]_\rho = [b]_{\rho[x \mapsto x]} \in [\![B]\!]_\sigma$. But the classification lemma and lemma 6.13 imply that $[\![B]\!]_\sigma$ is a saturated set, so $[b]_\rho \in \mathsf{SN}$. Thus, by lemma 6.4, $(\lambda x : [A]_\rho.[b]_\rho)\ a \in \mathsf{SN}$ as desired.

– Suppose instead that s is $\Box$. We must show $\lambda x : [A]_\rho.[b]_\rho \in \Pi([\![A]\!]_\sigma, \bigcap_{S \in V(A)} [\![B]\!]_{\sigma[x \mapsto S]})$.

  Let an expression $a \in [\![A]\!]_\sigma$ and a saturated set $S \in V(A)$ be given. It is enough to show $(\lambda x : [A]_\rho.[b]_\rho)\ a \in [\![B]\!]_{\sigma[x \mapsto S]}$.

Because $\sigma[x \mapsto S] \models \rho[x \mapsto a] : \Gamma, x : A$, the IH for $\mathcal{D}_2$ gives us that $[b]_{\rho[x \mapsto a]} \in [\![B]\!]_{\sigma[x \mapsto S]}$. As in the previous subcase, we can observe that:

$$(\lambda x : [A]_\rho.[b]_\rho)\ a \leadsto [a/x][b]_\rho = [b]_{\rho[x \mapsto a]}$$

This step contracts a key redex, and by reasoning as before we find $(\lambda x : [A]_\rho.[b]_\rho)\ a \in [\![B]\!]_{\sigma[x \mapsto S]}$ as desired.

Case: $\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash a : (x : A) \to B} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash b : A}}{\Gamma \vdash a\ b : [b/x]B}$ App

We must show $[a]_\rho\ [b]_\rho \in [\![[b/x]B]\!]_\sigma$, and the IH for $\mathcal{D}_2$ is that $[b]_\rho \in [\![A]\!]_\sigma$. We consider two cases: by the classification lemma and inversion, A is either a kind or a Γ-type.

- Suppose first that A is a kind. Then the IH for $\mathcal{D}_1$ gives us that

  $$[a]_\rho \in \Pi([\![A]\!]_\sigma, \bigcap_{S \in V(A)} [\![B]\!]_{\sigma[x \mapsto S]})$$

  In particular, expanding the definition of $\Pi(\cdot, \cdot)$ and applying lemma 6.13, we have:

  $$[a]_\rho\ [b]_\rho \in [\![B]\!]_{\sigma[x \mapsto [\![b]\!]_\sigma]}$$

  By lemma 6.11, this is just what we wanted to show.

- The case where A is a Γ-type is similar to but slightly simpler than the last case.

Case: $\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash a : A} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash B : s} \qquad A =_\beta B}{\Gamma \vdash a : B}$ Conv

The IH for $\mathcal{D}_1$ gives us that $[a]_\rho \in [\![A]\!]_\sigma$. By the classification lemma and $\mathcal{D}_2$, both A and B are kinds or Γ-constructors. Thus, by lemma 6.12, $[a]_\rho \in [\![B]\!]_\sigma$ as desired. $\square$