Type Checking with Universes

Robert Harper* Robert Pollack†

Abstract

Various formulations of constructive type theories have been proposed to serve as the basis for machine-assisted proof and as a theoretical basis for studying programming languages. Many of these calculi include a cumulative hierarchy of "universes," each a type of types closed under a collection of type-forming operations. Universes are of interest for a variety of reasons, some philosophical (predicative *vs.* impredicative type theories), some theoretical (limitations on the closure properties of type theories), and some practical (to achieve some of the advantages of a type of all types without sacrificing consistency.) The *Generalized Calculus of Constructions* (CC^ω) is a formal theory of types that includes such a hierarchy of universes. Although essential to the formalization of constructive mathematics, universes are tedious to use in practice, for one is required to make specific choices of universe levels and to ensure that all choices are consistent. In this paper we study several problems associated with type checking in the presence of universes in the context of CC^ω. First, we consider the basic type checking and well-typedness problems for this calculus. Second, we consider a formulation of Russell and Whitehead's "typical ambiguity" convention whereby universe levels may be elided, provided that some consistent assignment of levels leads to a correct derivation. Third, we consider the introduction of definitions to both the basic calculus and the calculus with typical ambiguity. This extension leads to a notion of "universe polymorphism" analogous to the type polymorphism of ML. Although our study is conducted for CC^ω, we

*School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3890

†Laboratory for Foundations of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ

expect that our methods will apply to other variants of the Calculus of Constructions and to type theories such as Constable's V3.

1 Introduction

A number of formulations of intuitionistic type theory have been considered as a basis for studying machine-assisted formal proof development, and as a theoretical foundation for the study of programming languages (see, for example, [16, 46, 34, 36, 37, 9, 10, 11, 14, 2, 4, 19], to name but a few.) One such system, the Calculus of Constructions (CC), was introduced by Coquand and Huet as a comprehensive basis for the formalization of constructive mathematics. [11, 14]. CC may be viewed as the λ-calculus associated, via the propositions-as-types principle [24], with natural deduction proofs in an extension of Church's higher-order logic [6]. The system has been proved both proof-theoretically [11] and model-theoretically [29, 17, 27] consistent, and the type checking problem has been proved decidable [14, 11].

Although CC is an exceedingly rich formalism for expressing mathematical constructions, a variety of extensions to the calculus have been considered [12, 30, 31]. These extensions are motivated by a variety of concerns, ranging from the desire to delineate the space of consistent extensions to the calculus, to the practical needs of formal proof and program development. One such consideration is the representation of mathematical structures such as algebras, automata, and ordered sets. It is by now widely recognized [36, 10] that the appropriate type-theoretic representation of mathematical structures is as elements of "strong sum" types[1] introduced by Martin-Löf [35, 36, 37] and Howard [24]. Strong sums have also been used to model modularity constructs in programming languages [33, 41, 2, 4].

Unfortunately, strong sums are, in a sense, incompatible with impredicativity [12, 23, 41]. As a result, it is necessary to extend the calculus with a level of types, and to postulate the closure of this additional level under the formation of strong sums. Mathematical structures are then represented as elements of types of this higher level. Having made this extension, one immediately sees that this process may be iterated, and that yet higher lev-

[1] Also known in the literature as "dependent products" and "generalized sums." These are not to be confused with the "weak sums" (or "existential types") introduced in connection with data abstraction [43].

els are needed for the formalization of such notions as the "category of all small categories." In recognition of this fact, Coquand introduced the "generalized" Calculus of Constructions [12] (CC^{ω}) which includes a cumulative hierarchy of *universes*. A universe is a type that is closed under the type-forming operations of the calculus: the formation of products and strong sums indexed by a type of that universe level. Cumulative hierarchies of this kind arise in many formal systems for mathematics; they arise in various guises in *Principia Mathematica* [44, 47] and in many contemporary type theories [35, 36, 37, 8, 9, 10].

Universe hierarchies are tedious to use in practice. Many workers have attempted to avoid the complications of such a hierarchy by assuming that there is a type of all types [34, 2, 38, 4]. This assumption destroys the normalization property of the calculus [35, 38, 25]. As a result, every type is inhabited by some closed term, and the interpretation of propositions as types, central to many applications, is lost. In the context of type systems for programming languages, the merits and demerits of the "type:type" assumption are the subject of ongoing research [38, 3, 41, 22].

An alternative approach to dealing with stratification in formal systems was introduced by Russell and Whitehead in *Principia Mathematica.* They introduced an informal convention, called "typical ambiguity," in which universe levels are not explicitly mentioned, and in which it is tacitly asserted that there exists an assignment of levels such that the resulting proof is correct with respect to the predicativity requirements of the logic of *Principia Mathematica.* Moreover, they observed that in practice the exact choice of universe levels is unimportant; what matters is the relationship between choices of levels at different points within a proof. From the modern perspective, typical ambiguity can be described as a way to achieve the flexibility of having a type of all types without sacrificing the logical consistency of the theory. At the level of the concrete syntax, the user can work without explicit mention of universe levels, leaving it to the proof checker to ensure that there is always a choice of levels that yields a type-correct term in the underlying calculus with explicitly stratified universes.

In this paper we study the *type checking* and *well-typedness* problems for four variants of CC^{ω}. The type checking problem for a calculus is to decide, given a context, term, and candidate type whether or not that term has that type in the given context. The well-typedness problem is to decide, given a context and a term, whether or not there exists a type such that that

term has that type in the given context. In each case the solution to these problems is obtained by a reduction to a *type synthesis algorithm* that yields, given a context and term, a description of the set of all possible types for that term in that context. Of course, the exact definitions of "context" and "term" will vary for each of the calculi that we consider, but the general pattern remains the same.

This paper is organized as follows. In Section 2 we define the system CC^ω, and state some of its important properties. In Section 3 we introduce an "operational presentation" of CC^ω, following [46, 21, 20, 42, 18] (among others.) The significance of the operational presentation is that it provides a normal form for typing derivations that is exploited by the type synthesis algorithm. In Section 4 we present a type synthesis and conversion algorithm for CC^ω in the "natural semantics" style of [7]. This form of presentation facilitates the proofs of correctness of the algorithm and makes especially evident the relationship between it and the operational rules. In Section 5 we extend the calculus to include an "anonymous" universe as a means of implementing the "typical ambiguity" convention. Explicit universe levels may be omitted by using instead the anonymous universe, with the understanding that such an "ambiguous" term stands for some consistent replacement of the anonymous by specific universes. In Section 6 we extend both the basic calculus and the calculus with anonymous universes to admit definitions in the form of δ-reductions. The failure of type unicity induced by the cumulativity of the universe hierarchy leads to a form of "universe polymorphism" similar to the "type polymorphism" of ML. The combination of anonymous universes with definitions leads to a particularly flexible calculus for exploiting typical ambiguity. Finally, in Section 7 we discuss related research.

We are grateful to Rod Burstall, Thierry Coquand, Joëlle Despeyroux, Susumu Hayashi, Gérard Huet, Gilles Kahn, Zhaohui Luo, and an anonymous referee for their insightful comments. This research was supported by grants from the British Science and Engineering Research Council (GR/D 64612 and GR/F 78487) and the U.S. Defense Advanced Research Projects Agency (contract number 5404).

2 The Definition of CC^ω

2.1 Syntax

The Generalized Calculus of Constructions [12] (CC^ω) is obtained by extending the basic Calculus of Constructions with a full cumulative hierarchy of type universes. Let x, y, z range over some infinite set of variables, and i, j, k range over the natural numbers. We use syntax given by the following grammar:

$$\begin{array}{lcll} \kappa & ::= & \mathsf{Prop} \mid \mathsf{Type}_i & \textit{kinds} \\ M & ::= & x \mid \kappa \mid [x{:}M]M \mid \{x{:}M\}M \mid M\,M & \textit{terms} \\ \Gamma & ::= & \bullet \mid \Gamma[x{:}M] & \textit{contexts} \end{array}$$

The metavariables A, B, K, L, M, N, and P range over terms; κ ranges over kinds. The terms Type_i are called *universes*. The pair $x{:}M$ in a context Γ is a *declaration*, and *declares* x. We only consider contexts in which no variable is declared more than once. $\mathrm{Dom}(\Gamma)$ is the set of variables declared in Γ. We write $\Gamma = \Gamma_x[x{:}A]\Gamma^x$ to mean that the declaration $x{:}A$ occurs in Γ, in the indicated position. The notions of free and bound variable are defined as usual. $\mathrm{FV}(M)$ is the set of free variables of M. We sometimes use the notation $A{\to}B$ for $\{x{:}A\}B$ when x doesn't occurr free in B.

2.2 Reduction and Conversion

In this setting, a β-redex has the form $([x{:}A]M)N$, and its contractum is $[N/x]M$. The relations $\to$ (one step β-reduction), $\twoheadrightarrow$ (β-reduction), and $\simeq$ (β-conversion) are defined as usual. The Church-Rosser property holds for β conversion:

Theorem 2.1 (CR) *If $M_1 \simeq M_2$ then there exists M such that $M_1 \twoheadrightarrow M$ and $M_2 \twoheadrightarrow M$.*
Proof *See [34] or [46].* □

A term is *strongly normalizing* (SN) if every reduction sequence starting from that term is finite.

The relation $\xrightarrow{wh}$ (one step weak head reduction) is defined by the rules of Table 1. $\xrightarrow{wh}\!\!\!\!\!\twoheadrightarrow$ (weak-head-reduction) is the transitive, reflexive closure of $\xrightarrow{wh}$.

$$\frac{M\ \beta\text{-contracts to } N}{M \xrightarrow{wh} N}$$

$$\frac{M \xrightarrow{wh} M'}{M\ N \xrightarrow{wh} M'\ N}$$

Table 1: One step weak head reduction

M is in *weak head normal form* (whnf) if it does not weak head reduce to any term except itself. Clearly, every term *not* in whnf is either a β-redex, or an application $M\ N$ where M is not in whnf. Thus, a term in weak head normal form has one of the shapes x, κ, $\{x{:}A\}B$, $[x{:}A]B$, or $M\ N$ where M is itself a weak head normal form of shape other than $[x{:}A]B$.

2.3 The Type System

CC^ω is a formal system for deriving assertions of the form $\Gamma \vdash M : A$, read as "M has type A in the context of type assumptions Γ". The axioms and rules of derivation for CC^ω are given in Table 2. We often write $\Gamma \vdash M : A$ to mean that the indicated assertion is derivable in the formal system, omitting explicit mention of Γ when it is the empty context.

A brief summary of the rules of CC^ω may be helpful. Rules VALI1 and VALI2 define the valid contexts to be those consisting of a sequence of declarations assigning to a variable either a proposition or a type (of arbitrary universe level). The rules VALE1 and VALE2 introduce the constants Prop and Type_i, and rule VALE3 governs typing of variables. The rules PIF1, PIF2, and PIF3 encode the fundamental closure conditions of CC^ω. The class of propositions is closed under universal quantification over any type, including any proposition, the class of propositions, and the types at any universe level (rule PIF1). Each universe is closed under products indexed by either a proposition, or a type of that level (rules PIF2 and PIF3). Rules PII and PIE govern lambda abstraction and application. Rule CONV asserts the invariance of typing under conversion of type expression, and rule CUM asserts the cumulativity of the hierarchy of universes, a property that plays a central role in this paper.

VALI1	$\bullet$ valid
VALI2	$\dfrac{\Gamma \vdash A : \kappa \quad x \notin \mathrm{Dom}(\Gamma)}{\Gamma[x{:}A] \text{ valid}}$
VALE1	$\dfrac{\Gamma \text{ valid}}{\Gamma \vdash \mathsf{Prop} : \mathsf{Type}_0}$
VALE2	$\dfrac{\Gamma \text{ valid}}{\Gamma \vdash \mathsf{Type}_i : \mathsf{Type}_{i+1}}$
VALE3	$\dfrac{\Gamma_x[x{:}A]\Gamma^x \text{ valid}}{\Gamma_x[x{:}A]\Gamma^x \vdash x : A}$
PIF1	$\dfrac{\Gamma[x{:}A] \vdash B : \mathsf{Prop}}{\Gamma \vdash \{x{:}A\}B : \mathsf{Prop}}$
PIF2	$\dfrac{\Gamma \vdash A : \mathsf{Prop} \quad \Gamma[x{:}A] \vdash B : \mathsf{Type}_i}{\Gamma \vdash \{x{:}A\}B : \mathsf{Type}_i}$
PIF3	$\dfrac{\Gamma \vdash A : \mathsf{Type}_i \quad \Gamma[x{:}A] \vdash B : \mathsf{Type}_i}{\Gamma \vdash \{x{:}A\}B : \mathsf{Type}_i}$
PII	$\dfrac{\Gamma[x{:}A] \vdash M : B}{\Gamma \vdash [x{:}A]M : \{x{:}A\}B}$
PIE	$\dfrac{\Gamma \vdash M : \{x{:}A\}B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : [N/x]B}$
CONV	$\dfrac{\Gamma \vdash M : A \quad \Gamma \vdash B : \kappa \quad A \simeq B}{\Gamma \vdash M : B}$
CUM	$\dfrac{\Gamma \vdash M : \mathsf{Type}_i}{\Gamma \vdash M : \mathsf{Type}_{i+1}}$

Table 2: Definition of CC^ω

Theorem 2.2 (Luo) *(Some properties of* CC^ω *.)*

- *Any derivation of* $\Gamma[x{:}A]\Gamma' \vdash M : B$ *has a subderivation of* $\Gamma \vdash A : \kappa$ *for some kind* κ*.*
- *Any derivation of* $\Gamma \vdash M : A$ *has a subderivation of* Γ valid*.*
- *If* $\Gamma \vdash M : A$ *then* $\Gamma \vdash A : \kappa$ *for some kind* κ*.*
- *(Subject Reduction) If* $\Gamma \vdash M : A$ *and* $M \twoheadrightarrow N$*, then* $\Gamma \vdash N : A$*.*
- *(Strong Normalization) If* $\Gamma \vdash M : A$*, then* M *is SN.*

Proof *See [30, 31]* □

3 Operational Presentation

As a step towards the presentation of a type checking algorithm for CC^ω, it is helpful to give a syntax-directed, or *operational* presentation of the calculus with the property that at most one rule of inference applies to a term. We begin with such a presentation of the conversion relation.

3.1 Conversion

The relation $M \downarrow N$ is defined by the rules of Table 3. Informally, $M \downarrow N$ holds iff M and N reduce by a standard reduction sequence to a common term. Thus if $M \downarrow N$, then M and N are convertible in the usual sense. The converse fails, for if a standard reduction sequence of M fails to yield a normal form, then $M \not\downarrow M$. However, if we restrict attention to well-typed terms, these relations coincide, and, moreover, the relation $\downarrow$ is decidable.

Theorem 3.1 (Conversion algorithm)

(Soundness) *If* $M \downarrow N$*, then* $M \simeq N$*.*

(Completeness) *If* $M \simeq N$ *and* M *and* N *are SN, then* $M \downarrow N$*.*

(Decidability) *It is decidable for SN* M *and* N *whether or not* $M \downarrow N$*.*

OC-TYPE
$$\frac{K \overset{wh}{\twoheadrightarrow} \mathsf{Type}_i \quad L \overset{wh}{\twoheadrightarrow} \mathsf{Type}_i}{K \downarrow L}$$

$$\frac{M \overset{wh}{\twoheadrightarrow} x \quad N \overset{wh}{\twoheadrightarrow} x}{M \downarrow N} \qquad \frac{K \overset{wh}{\twoheadrightarrow} \mathsf{Prop} \quad L \overset{wh}{\twoheadrightarrow} \mathsf{Prop}}{K \downarrow L}$$

$$\frac{A \overset{wh}{\twoheadrightarrow} \{x{:}A_1\}A_2 \quad B \overset{wh}{\twoheadrightarrow} \{x{:}B_1\}B_2 \quad A_1 \downarrow B_1 \quad A_2 \downarrow B_2}{A \downarrow B}$$

$$\frac{M \overset{wh}{\twoheadrightarrow} [x{:}A_1]M_1 \quad N \overset{wh}{\twoheadrightarrow} [x{:}A_2]M_2 \quad A_1 \downarrow A_2 \quad M_1 \downarrow M_2}{M \downarrow N}$$

$$\frac{M \overset{wh}{\twoheadrightarrow} M_1\ M_2 \quad N \overset{wh}{\twoheadrightarrow} N_1\ N_2 \quad M_1 \downarrow N_1 \quad M_2 \downarrow N_2}{M \downarrow N}$$

Table 3: β-Conversion Algorithm

Proof *A derivation of $M \downarrow N$ essentially specifies reduction sequences from M and N to the same term. Thus soundness is proved by induction on the definition of $\downarrow$.*

For completeness, suppose that M and N are SN and that $M \simeq N$. Then M and N have unique weak head normal forms, M_0 and N_0 respectively, with $M_0 \simeq M \simeq N \simeq N_0$. Proceed by induction on the structure of M_0. For example, if $M_0 = M_1\ M_2$ (where M_1 is not of shape $[x{:}A]B$), then we must have $N_0 = N_1\ N_2$ with $M_i \simeq N_i$ $(i = 1, 2)$. By induction hypothesis $M_i \downarrow N_i$, hence $M \downarrow N$. The other cases are similar.

Decidability follows from the fact that the requisite weak head normal forms exist. □

3.2 Operational Presentation of CC^ω

The inference rules defining the relation $\Gamma \vdash M : A$ are not completely syntax-directed since the rules CONV and CUM are applicable to any term. The operational presentation of CC^ω is a syntax-directed formal system for CC^ω that admits only limited applications of these rules, without sacrificing completeness (in a sense to be made precise below). The operational presentation is given in Table 4; an assertion of the form $\Gamma \vdash M \Rightarrow A$ is intended to mean "A is a type for M in Γ".

The operational presentation differs from the basic definition of CC^ω in several respects. One important difference is in the handling of contexts: context validity is assumed, rather than enforced. As a result, the rules O-GEN and O-ABS explicitly check the validity of the type of the bound variable in order to maintain this assumption. This formulation of the rules is closer to a practical implementation since it avoids the overhead of repeatedly checking context validity for each atomic term.

Another important difference is in the use of type conversion. In particular, rules O-GEN and O-ABS use only weak head reduction, rather than conversion, since, in the presence of the Church-Rosser property, a term is convertible to a kind only if it may be weak head-reduced to it. The rule O-APP uses the operational definition of conversion discussed above to match the domain and argument types.

Having limited the uses of conversion, some care must be taken to ensure

O-PROP $$\Gamma \vdash \mathsf{Prop} \Rightarrow \mathsf{Type}_i \qquad (i \geq 0)$$

O-TYPE $$\Gamma \vdash \mathsf{Type}_j \Rightarrow \mathsf{Type}_i \qquad (i > j \geq 0)$$

O-VAR $$\Gamma_x[x{:}A]\Gamma^x \vdash x \Rightarrow \mathsf{cum}(A, i) \qquad (i \geq 0)$$

O-GEN $$\frac{\Gamma \vdash A \Rightarrow K \quad K \stackrel{wh}{\twoheadrightarrow} \kappa_1 \quad x \notin \mathrm{Dom}(\Gamma) \quad \Gamma[x{:}A] \vdash B \Rightarrow L \quad L \stackrel{wh}{\twoheadrightarrow} \kappa_2}{\Gamma \vdash \{x{:}A\}B \Rightarrow \kappa_1 \uparrow_i \kappa_2} \qquad (i \geq 0)$$

O-ABS $$\frac{\Gamma \vdash A \Rightarrow K \quad K \stackrel{wh}{\twoheadrightarrow} \kappa \quad x \notin \mathrm{Dom}(\Gamma) \quad \Gamma[x{:}A] \vdash M \Rightarrow B}{\Gamma \vdash [x{:}A]M \Rightarrow \{x{:}A\}B}$$

O-APP $$\frac{\Gamma \vdash M \Rightarrow A \quad A \stackrel{wh}{\twoheadrightarrow} \{x{:}A_1\}A_2 \quad \Gamma \vdash N \Rightarrow B \quad B \downarrow A_1}{\Gamma \vdash MN \Rightarrow \mathsf{cum}([N/x]A_2, i)} \qquad (i \geq 0)$$

where cum *and* $\uparrow$ *are defined by:*

$$\mathsf{cum}(A, i) := \begin{cases} \mathsf{Type}_{j+i} & \textit{if } A \stackrel{wh}{\twoheadrightarrow} \mathsf{Type}_j \\ A & \textit{otherwise} \end{cases}$$

$$\kappa_1 \uparrow_i \kappa_2 := \begin{cases} \mathsf{Prop} & \textit{if } \kappa_2 = \mathsf{Prop} \\ \mathsf{Type}_{j+i} & \textit{if } \kappa_1 = \mathsf{Prop}, \kappa_2 = \mathsf{Type}_j \\ \mathsf{Type}_{\max(j,k)+i} & \textit{if } \kappa_1 = \mathsf{Type}_j, \kappa_2 = \mathsf{Type}_k \end{cases}$$

Table 4: Operational Presentation of CC^ω

that all potential uses of cumulativity are accounted for. For example

$$[x{:}([y{:}\mathsf{Type}_0]\mathsf{Type}_0)\,\mathsf{Prop}] \vdash x : \mathsf{Type}_2$$

because $([y{:}\mathsf{Type}_0]\mathsf{Type}_0)\,\mathsf{Prop} \simeq \mathsf{Type}_0$, and cumulativity then applies. Similarly

$$[x{:}\{f{:}\mathsf{Type}_0 \to \mathsf{Type}_1\}(f\ \mathsf{Prop})] \vdash x\ ([y{:}\mathsf{Type}_0]\mathsf{Type}_0) : \mathsf{Type}_2$$

These two examples illustrate the need for the function cum in rules O-VAR and O-APP: the result type may be convertible to a universe, and hence cumulativity may apply at that point. Similarly, rule O-GEN is defined in terms of the auxiliary function $\uparrow$ to account for potential cumulativity. The following lemma shows that Table 4 indeed has "enough cumulativity".

Lemma 3.2 *If* $\Gamma \vdash M \Rightarrow A$ *and* $A \stackrel{wh}{\twoheadrightarrow} \mathsf{Type}_i$, *then for all* $j \geq i$, $\Gamma \vdash M \Rightarrow \mathsf{Type}_j$.

Proof *By inspection of the rules of Table 4.* □

The main theorem of this section establishes the relationship between CC^ω and its operational presentation:

Theorem 3.3 (The operational presentation of CC^ω**)**

(Soundness) *If* Γ *is a valid context and* $\Gamma \vdash M \Rightarrow A$, *then* $\Gamma \vdash M : A$.

(Completeness) *If* $\Gamma \vdash M : A$, *then there exists* B *such that* $\Gamma \vdash M \Rightarrow B$ *and* $B \simeq A$.

Proof *We use Theorem 2.2 without further mention.*

(Soundness) *Let* δ *be a derivation of* $\Gamma \vdash M \Rightarrow A$. *We build a derivation of* $\Gamma \vdash M : A$ *by induction on height* δ. *In the base case* δ *is* O-PROP *(respectively* O-TYPE, O-VAR*), and the result follows from* VALE1, *(respectively* VALE2, VALE3*), and* CUM. *In the induction case, the root node of* δ *is one of* O-GEN, O-ABS, *or* O-APP. *Suppose, for example, it is* O-ABS, *so for some* A, K, M, *and* B, δ *is*

$$\frac{\begin{array}{cccc}\vdots & & & \vdots \\ \Gamma \vdash A \Rightarrow K & K \stackrel{wh}{\twoheadrightarrow} \kappa & x \notin \mathrm{Dom}(\Gamma) & \Gamma[x{:}A] \vdash M \Rightarrow B\end{array}}{\Gamma \vdash [x{:}A]M \Rightarrow \{x{:}A\}B}$$

By induction hypothesis, we have $\Gamma \vdash A : K$. *Since* $K \stackrel{wh}{\twoheadrightarrow} \kappa$ *(and* κ *is necessarily well-typed),* CONV *and* VALI2 *give* $\Gamma[x{:}A]$ valid. *Thus by induction hypothesis* $\Gamma[x{:}A] \vdash M : B$, *and* PII *shows* $\Gamma \vdash [x{:}A]M : \{x{:}A\}B$.

It is also interesting to consider the case where the root of δ *is* O-APP. *For some* M, N, A, B, *and* $i \geq 0$, δ *is*

$$\frac{\begin{array}{cccc}\vdots & & \vdots & \\ \Gamma \vdash M \Rightarrow A & A \stackrel{wh}{\twoheadrightarrow} \{x{:}A_1\}A_2 & \Gamma \vdash N \Rightarrow B & B \downarrow A_1\end{array}}{\Gamma \vdash MN \Rightarrow \mathsf{cum}([N/x]A_2, i)}$$

By Theorem 3.1 $B \simeq A_1$. *By induction hypothesis, we have* $\Gamma \vdash M : A$ *and* $\Gamma \vdash N : B$. *Thus* A *is well-typed, and by subject reduction, so is* $\{x{:}A_1\}A_2$. *By* CONV $\Gamma \vdash M : \{x{:}A_1\}A_2$, *so by* PIE *and* CUM, $\Gamma \vdash MN : \mathsf{cum}([N/x]A_2, i)$ *as required. The other cases are similar.*

(Completeness) *Let* δ *be a derivation of* $\Gamma \vdash M : A$, *and build a derivation of* $\Gamma \vdash M \Rightarrow B$ *(for some* B*) by induction on height* δ. *Consider the possible cases for the root node of* δ. *If* δ *ends with* CONV *(respectively* CUM*), the result is immediate by induction (respectively Lemma 3.2). All other cases follow directly from the induction hypothesis.* □

The operational presentation is "syntax directed" in the sense that the structure of a derivation of $\Gamma \vdash M \Rightarrow A$ is determined by the structure of M. However, the relation $\Gamma \vdash M \Rightarrow A$ is not a partial function of Γ and M, due to the cumulativity of the universe hierarchy. In fact, this is the only source of variation: two derivations for a given term and context differ only in the choice of universe index parameters of the operational rules. The choice of these parameters is sometimes constrained by context. For example, when deriving a type for the term $([x{:}\mathsf{Type}_2]x)\,\mathsf{Prop}$, the universe level of the sole occurrence of Prop is constrained to be 0, 1, or 2 by the fact that it occurs as the argument to a function with domain Type_2. On the other hand, any universe level greater than 2 is admissible as the type of Type_2 itself. It is important to realize that the range of possible types for a term is determined by the structure of the term itself, and not by its type. For example $\vdash [x{:}\mathsf{Prop}]\mathsf{Prop} \Rightarrow \{x{:}\mathsf{Prop}\}\mathsf{Type}_i$ for all $i \geq 0$, but $[y{:}\{x{:}\mathsf{Prop}\}\mathsf{Type}_0] \vdash y \Rightarrow \{x{:}\mathsf{Prop}\}\mathsf{Type}_i$ is only derivable for $i = 0$. Thus, although cumulativity may be thought of as a form of type containment, it should be distinguished sharply from type systems that impose an upward closure condition on typing with respect to some pre-order on types.

In order to produce a deterministic algorithm based on the operational presentation, we remove the indeterminacy by *postponing decisions*: a choice of several possible outcomes is replaced by a single *schematic* outcome. To this end, we introduce notions of schematic term and constraint in the next section, and uniformly schematize the operational presentation. In fact, this approach allows us, in later sections, to formalize (operationally) and implement (algorithmically) notions of "typical ambiguity" and "universe polymorphism".

4 Decision Problems for CC^ω

In this section we present a *schematic type synthesis algorithm* that, given a valid context Γ and a term M, yields a schematic description of the set of possible types for M relative to Γ. This algorithm makes use of an algorithm for testing convertibility of schematic terms, which we also present. Solutions to the well-typedness and type checking problems for CC^ω are easily derived from these algorithms.

4.1 Schematic Terms

Let α, β, and γ range over some infinite set of *level variables*, and let λ and μ range over the *level expressions*, consisting of level variables and natural numbers. The *schematic terms*, ranged over by X, Y, and Z, are terms that may involve *universe schemes* of the form Type_α. Universe schemes are regarded as kinds; we still use κ to range over this extended notion of kinds. Thus:

λ	::=	$i \mid \alpha$	*level expressions*
κ	::=	$\mathsf{Prop} \mid \mathsf{Type}_\lambda$	*kinds*
X	::=	$x \mid \kappa \mid [x{:}X]X \mid \{x{:}X\}X \mid X\,X$	*terms*

$\mathrm{LV}(X)$ is the set of level variables occurring in X.

A constraint set is a finite set of inequalities of the form $\lambda \geq \mu$ or $\lambda > \mu$. We sometimes write $\lambda = \mu$ for the pair of constraints $\lambda \geq \mu, \mu \geq \lambda$. The metavariables $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$, $\mathcal{G}$ range over constraint sets. $\mathrm{LV}(\mathcal{C})$ is the set of level variables occurring in constraint set $\mathcal{C}$.

A *level assignment* is a partial function assigning natural numbers to a finite set of level variables. The metavariables σ and τ range over level assignments. $\mathrm{Dom}(\sigma)$ is the set of level variables σ assigns to; *i.e.* its domain as a function. A level assignment σ *satisfies* a constraint set $\mathcal{C}$, written $\sigma \models \mathcal{C}$, iff $\mathrm{Dom}(\sigma) \supseteq \mathrm{LV}(\mathcal{C})$ and each of the inequalities in $\mathcal{C}$ is true under the assignment σ. A constraint set is *satisfiable*, or *consistent* if there is some level assignment that satisfies it. The following result is due to Chan [5]:

Theorem 4.1 (Chan) *There is a polynomial time algorithm to determine whether or not there exists a level assignment satisfying a given constraint set.*

In fact the running time of this algorithm is bounded by $\mathcal{O}(m, n^3)$, where m is the number of constraints and n is the number of level variables.

Level assignments are extended to schematic terms in the obvious way: σX is the schematic term obtained from X by replacing all occurrences of Type_α, where $\alpha \in \mathrm{Dom}(\sigma)$, by $\mathsf{Type}_{\sigma(\alpha)}$. The term σX is called an *instance* of X. Notice that an instance of X may still contain level variables. Level assignments are written explicitly as $[\alpha_1 \mapsto i_1, \ldots, \alpha_k \mapsto i_k]$. We write $\sigma[\alpha \mapsto i]$ for the level assignment that assigns i to α, and otherwise behaves like σ.

Lemma 4.2 (Reduction and Schematic Terms)

1. *σX is SN iff X is SN.*
2. *$\sigma Z \overset{wh}{\twoheadrightarrow} \mathsf{Type}_\mu$ iff $Z \overset{wh}{\twoheadrightarrow} \mathsf{Type}_\lambda$ with $\sigma\lambda = \mu$*
3. *$\sigma Z \overset{wh}{\twoheadrightarrow} \{x{:}X_1\}X_2$ iff $Z \overset{wh}{\twoheadrightarrow} \{x{:}Z_1\}Z_2$ for some Z_1 and Z_2 such that $\sigma Z_1 = X_1$ and $\sigma Z_2 = X_2$.*
4. *Similar to 3. for cases: (a) $Z \overset{wh}{\twoheadrightarrow} [x{:}Z_1]Z_2$, (b) $Z \overset{wh}{\twoheadrightarrow} Z_1\ Z_2$.*

Proof *Let ∂ be a subterm occurrence of X (see [1]). Since level assignment completely respects the structure of terms, we may abuse notation to say that σ is a bijection between subterm occurrences in X and subterm occurrences in σX. Since reduction is defined without regard to universe levels, it is clear that ∂ is a redex in X iff $\sigma\partial$ is a redex in σX. Also, $\sigma X \overset{\sigma\partial}{\to} Y$ (contracting redex $\sigma\partial$ in σX produces Y) iff $X \overset{\partial}{\to} Z$ and $\sigma Z = Y$. By induction, we get a*

SC-TYPE
$$\frac{X \overset{wh}{\twoheadrightarrow} \mathsf{Type}_\lambda \quad Y \overset{wh}{\twoheadrightarrow} \mathsf{Type}_\mu}{X \downarrow Y \ (\{\, \lambda = \mu \,\})}$$

$$\frac{X \overset{wh}{\twoheadrightarrow} x \quad Y \overset{wh}{\twoheadrightarrow} x}{X \downarrow Y \ (\emptyset)} \qquad \frac{X \overset{wh}{\twoheadrightarrow} \mathsf{Prop} \quad Y \overset{wh}{\twoheadrightarrow} \mathsf{Prop}}{X \downarrow Y \ (\emptyset)}$$

$$\frac{X \overset{wh}{\twoheadrightarrow} \{x{:}X_1\}X_2 \quad Y \overset{wh}{\twoheadrightarrow} \{x{:}Y_1\}Y_2 \quad X_1 \downarrow Y_1 \ (\mathcal{C}_1) \quad X_2 \downarrow Y_2 \ (\mathcal{C}_2)}{X \downarrow Y \ (\mathcal{C}_1 \cup \mathcal{C}_2)}$$

$$\frac{X \overset{wh}{\twoheadrightarrow} [x{:}X_1]X_2 \quad Y \overset{wh}{\twoheadrightarrow} [x{:}Y_1]Y_2 \quad X_1 \downarrow Y_1 \ (\mathcal{C}_1) \quad X_2 \downarrow Y_2 \ (\mathcal{C}_2)}{X \downarrow Y \ (\mathcal{C}_1 \cup \mathcal{C}_2)}$$

$$\frac{X \overset{wh}{\twoheadrightarrow} X_1 \ X_2 \quad Y \overset{wh}{\twoheadrightarrow} Y_1 \ Y_2 \quad X_1 \downarrow Y_1 \ (\mathcal{C}_1) \quad X_2 \downarrow Y_2 \ (\mathcal{C}_2)}{X \downarrow Y \ (\mathcal{C}_1 \cup \mathcal{C}_2)}$$

Table 5: Schematic β-Conversion Algorithm

similar result for arbitrary reduction sequences. Thus σ can also be thought of as a bijection between the reduction sequences from X and those from σX, that preserves the bijection of subterm occurrences. This proves all parts of the lemma. □

Table 5 defines a conversion algorithm for schematic terms that, given schematic terms X and Y, yields the weakest constraint set $\mathcal{C}$ such that if $\sigma \models \mathcal{C}$, then $\sigma X \downarrow \sigma Y$. The precise characterization of this relation is given by the following theorem:

Theorem 4.3 (Conversion algorithm for schematic terms)

(Soundness) *If $X \downarrow Y \ (\mathcal{C})$ and $\sigma \models \mathcal{C}$, then $\sigma X \downarrow \sigma Y$.*

(Completeness) *If $\sigma X \downarrow \sigma Y$, then there exists $\mathcal{C}$ such that $X \downarrow Y\,(\mathcal{C})$ and $\sigma \models \mathcal{C}$.*

(Decidability) *It is decidable, given strongly normalizing schematic terms X and Y, whether or not there exists a constraint set $\mathcal{C}$ such that $X \downarrow Y\,(\mathcal{C})$ is derivable.*

Proof (Soundness) *Let δ be a derivation of $X \downarrow Y\,(\mathcal{C})$. The hypothesis $\sigma \models \mathcal{C}$ guarantees that $\sigma\delta$ is a derivation of $\sigma X \downarrow \sigma Y$ (proved by induction on the height of δ, using Lemma 4.2).*

(Completeness) *Let δ be a derivation of $\sigma X \downarrow \sigma Y$. By induction on height δ we construct a derivation of $X \downarrow Y\,(\mathcal{C})$ such that $\sigma \models \mathcal{C}$. For example, suppose δ is an instance of* OC-TYPE*:*

$$\frac{\sigma X \stackrel{wh}{\twoheadrightarrow} \mathsf{Type}_i \quad \sigma Y \stackrel{wh}{\twoheadrightarrow} \mathsf{Type}_i}{\sigma X \downarrow \sigma Y}$$

By Lemma 4.2, $X \stackrel{wh}{\twoheadrightarrow} \mathsf{Type}_\lambda$ with $\sigma\lambda = i$, and $Y \stackrel{wh}{\twoheadrightarrow} \mathsf{Type}_\mu$ with $\sigma\mu = i$, so by SC-TYPE*, $X \downarrow Y\,(\{\,\lambda = \mu\,\})$, and $\sigma \models \{\,\lambda = \mu\,\}$. The other cases are similar.*

(Decidability) *Since X and Y are SN, all the required weak head normal forms exist.* □

4.2 Schematic Type Synthesis

An algorithm for schematic type synthesis is given by the rules of Table 6. It is a system for deriving judgements of the form $\Gamma \vdash M \Rightarrow X, \mathcal{C}$. Intuitively, $X, \mathcal{C}$ schematically represent the set of types for M in Γ. The algorithm makes use of two auxiliary functions, CUM and $\Uparrow$, defined in Table 6. These functions are analogous to the functions cum and $\uparrow$ of Table 4, and are characterized by the following lemmas.

Lemma 4.4 *Suppose $\mathsf{CUM}(X, \mathcal{C}) = (Y, \mathcal{D})$ (respectively $\kappa_1 \Uparrow_{\mathcal{C}} \kappa_2 = (\kappa, \mathcal{D})$).*

1. *If $\sigma \models \mathcal{D}$, then there exists $i \geq 0$ such that $\sigma Y = \mathsf{cum}(\sigma X, i)$ (respectively $\sigma\kappa = \sigma\kappa_1 \uparrow_i \sigma\kappa_2$).*
2. *If $\sigma \models \mathcal{C}$, $\mathrm{Dom}(\sigma) \cap \mathrm{LV}(\mathcal{D}) = \mathrm{LV}(\mathcal{C})$, and $i \geq 0$, there exists τ extending σ such that $\tau \models \mathcal{D}$, $\mathrm{Dom}(\tau) = \mathrm{Dom}(\sigma) \cup \mathrm{LV}(\mathcal{D})$, and $\tau Y = \mathsf{cum}(\tau X, i)$ (respectively $\tau\kappa = \tau\kappa_1 \uparrow_i \tau\kappa_2$).*

A-PROP $$\Gamma \vdash \mathsf{Prop} \Rightarrow \mathsf{Type}_\alpha, \{\, \alpha \geq 0 \,\} \qquad (\alpha \ new)$$

A-TYPE $$\Gamma \vdash \mathsf{Type}_j \Rightarrow \mathsf{Type}_\alpha, \{\, \alpha > j \,\} \qquad (\alpha \ new)$$

A-VAR $$\Gamma_x[x{:}A]\Gamma^x \vdash x \Rightarrow \mathsf{CUM}(A, \emptyset)$$

A-GEN $$\frac{\Gamma \vdash A \Rightarrow X, \mathcal{C} \quad X \stackrel{wh}{\twoheadrightarrow} \kappa_1 \quad x \notin \mathrm{Dom}(\Gamma) \quad \mathcal{C} \text{ consistent} \qquad \Gamma[x{:}A] \vdash B \Rightarrow Y, \mathcal{D} \quad Y \stackrel{wh}{\twoheadrightarrow} \kappa_2}{\Gamma \vdash \{x{:}A\}B \Rightarrow \kappa_1 \Uparrow_{\mathcal{C}\cup\mathcal{D}} \kappa_2}$$

A-ABS $$\frac{\Gamma \vdash A \Rightarrow X, \mathcal{C} \quad X \stackrel{wh}{\twoheadrightarrow} \kappa \quad x \notin \mathrm{Dom}(\Gamma) \quad \mathcal{C} \text{ consistent} \qquad \Gamma[x{:}A] \vdash M \Rightarrow Y, \mathcal{D}}{\Gamma \vdash [x{:}A]M \Rightarrow \{x{:}A\}Y, \mathcal{C} \cup \mathcal{D}}$$

A-APP $$\frac{\Gamma \vdash M \Rightarrow X, \mathcal{C} \quad X \stackrel{wh}{\twoheadrightarrow} \{x : X_1\}X_2 \qquad \Gamma \vdash N \Rightarrow Y, \mathcal{D} \quad X_1 \downarrow Y(\mathcal{E})}{\Gamma \vdash MN \Rightarrow \mathsf{CUM}([N/x]X_2, \mathcal{C} \cup \mathcal{D} \cup \mathcal{E})}$$

where CUM *and* $\Uparrow$ *are defined by the following (where* α *is a new lvar):*

$$\mathsf{CUM}(X, \mathcal{C}) := \begin{cases} \mathsf{Type}_\alpha, \mathcal{C} \cup \{\, \alpha \geq \lambda \,\} & \text{if } X \stackrel{wh}{\twoheadrightarrow} \mathsf{Type}_\lambda \\ X, \mathcal{C} & \text{otherwise} \end{cases}$$

$$\kappa_1 \Uparrow_{\mathcal{C}} \kappa_2 := \begin{cases} \mathsf{Prop}, \mathcal{C} & \text{if } \kappa_2 = \mathsf{Prop} \\ \mathsf{Type}_\alpha, \mathcal{C} \cup \{\, \alpha \geq \lambda \,\} & \text{if } \kappa_1 = \mathsf{Prop}, \kappa_2 = \mathsf{Type}_\lambda \\ \mathsf{Type}_\alpha, \mathcal{C} \cup \{\, \alpha \geq \lambda, \alpha \geq \mu \,\} & \text{if } \kappa_1 = \mathsf{Type}_\lambda, \kappa_2 = \mathsf{Type}_\mu \end{cases}$$

Table 6: Type Synthesis Algorithm for CC^ω

Proof *We prove the CUM clauses; the other parts are similar.*

1. *If* $X \overset{wh}{\twoheadrightarrow} \mathsf{Type}_\lambda$ *then* $\mathcal{D} = \mathcal{C} \cup \{\, \alpha \geq \lambda \,\}$, *and* $Y = \mathsf{Type}_\alpha$. *So* $\sigma X \overset{wh}{\twoheadrightarrow} \mathsf{Type}_{\sigma\lambda}$, *and, since* $\sigma \models \mathcal{D}$, *we may take* $i = \sigma\alpha - \sigma\lambda \geq 0$ *to obtain*

$$\mathsf{cum}(\sigma X, i) = \mathsf{Type}_{\sigma\lambda+(\sigma\alpha-\sigma\lambda)} = \mathsf{Type}_{\sigma\alpha} = \sigma Y.$$

If $X \overset{wh}{\not\twoheadrightarrow} \mathsf{Type}_\lambda$ *for any* λ, *then* $\sigma X \overset{wh}{\not\twoheadrightarrow} \mathsf{Type}_j$ *for any* j, *so for any* i

$$\mathsf{cum}(\sigma X, i) = \sigma X = \sigma Y$$

2. *If* $\sigma X \overset{wh}{\twoheadrightarrow} \mathsf{Type}_j$, *then* $X \overset{wh}{\twoheadrightarrow} \mathsf{Type}_\lambda$ *for some* λ *with* $\sigma\lambda = j$. *Hence* $\mathcal{D} = \mathcal{C} \cup \{\, \alpha \geq \lambda \,\}$ *for some* $\alpha \notin \mathrm{LV}(\mathcal{C})$, $Y = \mathsf{Type}_\alpha$, *and* $\mathsf{cum}(\sigma X, i) = \mathsf{Type}_{j+i}$. *Let* τ *be* σ *extended with* $\alpha \mapsto j + i$. *If* $\sigma X \overset{wh}{\not\twoheadrightarrow} \mathsf{Type}_j$ *for any* j, *then* $X \overset{wh}{\not\twoheadrightarrow} \mathsf{Type}_\lambda$ *for any* λ, *so take* $\tau = \sigma$. □

The rules of Table 6 make use of an informal convention whereby level variables are required to be "new". This means that the level variable chosen at that rule occurrence is unique to that occurrence, and different from that associated with any occurrence of any other rule in the derivation under consideration. This convention can be made precise, at the expense of considerable technical complication, by introducing a set of "used" level variables, and requiring that α be chosen apart from this "used" set. (See [40] for a careful treatment of a similar problem.)

Theorem 4.5 (Type synthesis algorithm for CC^ω**)**

(Soundness) *If* $\Gamma \vdash M \Rightarrow X, \mathcal{C}$, *then*

1. $\mathrm{LV}(X) \subseteq \mathrm{LV}(\mathcal{C})$ *and* $\mathrm{LV}(\mathcal{C})$ *is a set of "new" level variables.*
2. *If* $\sigma \models \mathcal{C}$, *then* $\Gamma \vdash M \Rightarrow \sigma X$.

(Completeness) *If* $\Gamma \vdash M \Rightarrow A$, *then there exists* X, $\mathcal{C}$, *and* σ_A *such that*

1. $\Gamma \vdash M \Rightarrow X, \mathcal{C}$,
2. $\sigma_A \models \mathcal{C}$, *and* $\mathrm{Dom}(\sigma_A) = \mathrm{LV}(\mathcal{C})$
3. $\sigma_A X = A$.

(Decidability) *It is decidable, given a valid context* Γ *and a term* M*, whether or not there exists a schematic term* X *and consistent constraint set* $\mathcal{C}$ *such that* $\Gamma \vdash M \Rightarrow X,\mathcal{C}$ *is derivable.*

Proof (Soundness) *The first property is proved by inspection of the rules of Table 6. For the second property, consider a derivation* δ *of* $\Gamma \vdash M \Rightarrow X,\mathcal{C}$*. Roughly speaking, the constraint set* $\mathcal{C}$ *is sufficient to ensure that "*$\sigma\delta$*" is a valid derivation of* $\Gamma \vdash M \Rightarrow \sigma X$*. More precisely, we build a derivation of* $\Gamma \vdash M \Rightarrow \sigma X$ *by induction on the height of* δ*. The induction proceeds by case analysis of the root node of* δ *based on the rules of Table 6. The most interesting case is when the root of* δ *is an instance of rule* (A-APP)*. Then* δ *has the form:*

$$\frac{\begin{array}{cccc}\vdots & & \vdots & \\ \Gamma \vdash M \Rightarrow Z,\mathcal{F} & Z \stackrel{wh}{\twoheadrightarrow} \{x : Z_1\}Z_2 & \Gamma \vdash N \Rightarrow Y,\mathcal{D} & Z_1 \downarrow Y(\mathcal{E})\end{array}}{\Gamma \vdash MN \Rightarrow X,\mathcal{C}}$$

where $(X,\mathcal{C}) = \mathsf{CUM}([N/x]Z_2, \mathcal{F} \cup \mathcal{D} \cup \mathcal{E})$*. By definition of* CUM*,* $\mathcal{F} \cup \mathcal{D} \cup \mathcal{E} \subseteq \mathcal{C}$*, so* $\sigma \models \mathcal{F}$*,* $\sigma \models \mathcal{D}$*, and* $\sigma \models \mathcal{E}$*. Thus* $\Gamma \vdash M \Rightarrow \sigma Z$ *and* $\Gamma \vdash N \Rightarrow \sigma Y$ *by induction hypothesis. Also* $\sigma Z \stackrel{wh}{\twoheadrightarrow} \{x{:}\sigma Z_1\}\sigma Z_2$ *(by Lemma 4.2), and* $\sigma Z_1 \downarrow \sigma Y$ *(by Theorem 4.3). We have shown the hypotheses of rule* (O-APP) *are satisfied, and conclude:*

$$\Gamma \vdash MN \Rightarrow \mathsf{cum}([N/x]\sigma Z_2, i) \qquad \textit{(for any } i \geq 0\textit{)}.$$

Finally, by Lemma 4.4, there is some $i \geq 0$ *such that* $\sigma X = \mathsf{cum}([N/x]\sigma Z_2, i)$ *as required. The other cases are handled similarly.*

(Completeness) *Let* δ *be a derivation of* $\Gamma \vdash M \Rightarrow A$*. By induction on height* δ *we build a derivation of* $\Gamma \vdash M \Rightarrow X,\mathcal{C}$ *(for* some $X,\mathcal{C}$*), and an assignment* σ_A*. As in the Soundness proof, proceed by case analysis of the root node of* δ *based on the rules of Table 4. We consider two cases; the rest are handled similarly.*

If δ *is an instance of the axiom* (O-TYPE) *(this is a base case):*

$$\Gamma \vdash \mathsf{Type}_j \Rightarrow \mathsf{Type}_i \qquad (\textit{some } i,j\textit{, with } i > j)$$

we have rule (A-TYPE) $\Gamma \vdash \mathsf{Type}_j \Rightarrow \mathsf{Type}_\alpha, \{\,\alpha > j\,\}$*, and* $\sigma_{\mathsf{Type}_i} = \{\,\alpha \mapsto i\,\}$*.*

If the root of δ is an instance of rule (O-APP)*:*

$$\frac{\begin{array}{cccc}\vdots & & \vdots & \\ \Gamma \vdash M \Rightarrow A & A \overset{wh}{\twoheadrightarrow} \{x{:}A_1\}A_2 & \Gamma \vdash N \Rightarrow B & B \downarrow A_1\end{array}}{\Gamma \vdash MN \Rightarrow \mathsf{cum}([N/x]A_2, i)} \qquad (\textit{some } i \geq 0)$$

By induction hypothesis there exist $Z_A, \mathcal{E}_A$ and σ_A such that:

$$\Gamma \vdash M \Rightarrow Z_A, \mathcal{E}_A, \quad \sigma_A \models \mathcal{E}_A, \quad \mathrm{Dom}(\sigma_A) = \mathrm{LV}(\mathcal{E}_A), \quad \sigma_A Z_A = A$$

and by Lemma 4.2:

$$Z_A \overset{wh}{\twoheadrightarrow} \{x{:}Z_1\}Z_2 \quad \textit{where } \sigma_A Z_1 = A_1 \textit{ and } \sigma_A Z_2 = A_2.$$

Similarly there exist $Z_B, \mathcal{E}_B$ and σ_B such that:

$$\Gamma \vdash N \Rightarrow Z_B, \mathcal{E}_B, \quad \sigma_B \models \mathcal{E}_B, \quad \mathrm{Dom}(\sigma_B) = \mathrm{LV}(\mathcal{E}_B), \quad \sigma_B Z_B = B$$

By the "new" convention $LV(\mathcal{E}_A)$ and $\mathrm{LV}(\mathcal{E}_B)$ are disjoint, so let $\sigma_{A,B} = \sigma_A \cup \sigma_B$. Now

$$\sigma_{A,B} Z_1 = \sigma_A Z_1 = A_1 \downarrow B = \sigma_B Z_B = \sigma_{A,B} Z_B$$

and by Theorem 4.3:

$$Z_1 \downarrow Z_B\,(\mathcal{F}) \qquad \textit{with} \qquad \sigma_{A,B} \models \mathcal{F}$$

We have shown the hypotheses of rule (A-APP) *are satisfied:*

$$\Gamma \vdash M \Rightarrow Z_A, \mathcal{E}_A, \quad Z_A \overset{wh}{\twoheadrightarrow} \{x{:}Z_1\}Z_2, \quad \Gamma \vdash N \Rightarrow Z_B, \mathcal{E}_B, \quad Z_1 \downarrow Z_B\,(\mathcal{F})$$

so we have

$$\Gamma \vdash MN \Rightarrow W, \mathcal{G} \quad \textit{where } W, \mathcal{G} = \mathsf{CUM}([N/x]Z_2, \mathcal{E}_A \cup \mathcal{E}_B \cup \mathcal{F})$$

Now Lemma 4.4 applies to extend $\sigma_{A,B}$ to the required $\sigma_{\mathsf{cum}([N/x]A_2,i)}$.

(Decidability) *The proof is by induction on the structure of M, keeping in mind that the rules of Table 6 are syntax-directed. The base cases (*Prop, Type_i*, and variables) are all trivial: for the case of a variable x, we need*

only check that x is declared in the context. (The constraint set is clearly satisfiable.)

For the induction, consider the case of an application, MN. We are to show that we can decide whether or not there exists a schematic term Z and a consistent constraint set $\mathcal{F}$ such that $\Gamma \vdash MN \Rightarrow Z, \mathcal{F}$. If any such Z and $\mathcal{F}$ exist, then the required derivation must end with an application of rule A-APP. *By the induction hypothesis it is decidable whether or not there exists $X, \mathcal{C}$ and $Y, \mathcal{D}$ such that $\Gamma \vdash M \Rightarrow X, \mathcal{C}$ and $\Gamma \vdash N \Rightarrow Y, \mathcal{D}$ are both derivable, and such that both $\mathcal{C}$ and $\mathcal{D}$ are consistent. If not, then fail, for otherwise no derivation of the required form can exist. To see this, note that even if both subderivations exist, but with either $\mathcal{C}$ or $\mathcal{D}$ inconsistent, then the only possible choice of $\mathcal{F}$ is also inconsistent. Otherwise, by soundness, Theorem 3.3, Lemma 4.2, and Theorem 2.2, both X and Y are strongly normalizing. Hence, we may effectively test whether or not $X \stackrel{wh}{\twoheadrightarrow} \{x{:}X_1\}X_2$ and, by Theorem 4.3, whether or not there exists $\mathcal{E}$ such that $X_1 \downarrow Y\,(\mathcal{E})$. If either of these conditions fail, then there can be no derivation of the required form. Otherwise, we may apply Chan's algorithm to test whether or not the constraint set $\mathcal{C} \cup \mathcal{D} \cup \mathcal{E}$ is consistent. If so, succeed with $Z = [N/x]X_2$ and $\mathcal{F} = \mathcal{C} \cup \mathcal{D} \cup \mathcal{E}$, and fail otherwise (there is no other choice of Z and $\mathcal{F}$.)*

For the remaining cases we have only to note that the check for consistency of the constraint set $\mathcal{C}$ is rules A-GEN *and* A-ABS *ensures (by soundness) that the validity of the context is preserved.* □

Corollary 4.6 *The well-typedness and type checking problems for* CC^ω *are effectively solvable.*

Proof *Let Γ be a valid context, and let M be a term. By the theorem we can effectively decide whether or not there exists a schematic term X and consistent constraint set $\mathcal{C}$ such that $\Gamma \vdash M \Rightarrow X, \mathcal{C}$. By soundness and completeness, such X and $\mathcal{C}$ exist iff M is well-typed in Γ. To check whether A is a valid type for M in Γ, we may effectively check (by Theorem 4.3) whether or not there exists $\mathcal{D}$ such that $X \downarrow A\,(\mathcal{D})$, and whether or not (by Theorem 4.1) $\mathcal{C} \cup \mathcal{D}$ is consistent. If so, then by the soundness of the conversion and type synthesis algorithms, A is a valid type for Γ. If not, then by the completeness of the conversion and type synthesis algorithms, A cannot be a valid type for M in Γ.* □

5 Anonymous Universes

In this section we consider the well-typedness and type checking problems for the extension of CC^ω with an *anonymous universe*, Type. This extension is intended to model Russell and Whitehead's "typical ambiguity" convention. The idea is that in a proof explicit universe levels may be soundly omitted, provided that some consistent assignment of levels exists. Moreover, every consistent assignment results in a valid proof: it is not the absolute values of the universe levels that matters, only their relation to one another.

5.1 Extending the Operational Presentation

Let Q, R, S, and T range over *ambiguous* terms which may contain occurrences of the anonymous universe, Type. An ambiguous term, Q, is to be understood as a convenient shorthand for some *reading* obtained by replacing each occurrence of Type in Q by a specific universe Type_i. From an algorithmic point of view, the ambiguity in a term is resolved during type checking, with the choice of reading constrained by the context of the occurrence. For example, in the term $([x{:}\mathsf{Type}_2]x)\,\mathsf{Type}$ the anonymous universe may be read as standing for either Type_0 or Type_1, but not as Type_2 or any higher universe. Similarly, in the term $([x{:}\mathsf{Type}]x)\,\mathsf{Type}_1$, the type of the bound variable x can be read as standing for Type_i only for $i \geq 2$ due to the application to Type_1.

Table 7 is an operational presentation of the typing rules for CC^ω with anonymous universes. These rules specify the derivability conditions for judgements of the form $\Gamma \vdash Q \Rightarrow M, A$, where Γ is a context (as defined in Section 2), Q is an ambiguous term, and M and A are ordinary terms. This judgement is to be understood as expressing that M is a reading of Q, and A is a type for M. It is important to stress that the context Γ *cannot* contain ambiguous terms: the type of a variable is fixed when it is put into the context (see, for example, rule O-A-ABS in Table 7.)

The fundamental properties of the system of Table 7 are summarized by the following theorem.

Theorem 5.1

(Soundness) *If* $\Gamma \vdash Q \Rightarrow M, A$, *then* M *is a reading of* Q *and* $\Gamma \vdash M \Rightarrow A$.

(Completeness) *If* M *is a reading of* Q *and* $\Gamma \vdash M \Rightarrow A$, *then* $\Gamma \vdash Q \Rightarrow M, A$.

O-A-PROP $$\Gamma \vdash \mathsf{Prop} \Rightarrow \mathsf{Prop}, \mathsf{Type}_i \qquad (i \geq 0)$$

O-A-TYPE $$\Gamma \vdash \mathsf{Type}_j \Rightarrow \mathsf{Type}_j, \mathsf{Type}_i \qquad (i > j \geq 0)$$

O-A-ANON $$\Gamma \vdash \mathsf{Type} \Rightarrow \mathsf{Type}_j, \mathsf{Type}_i \qquad (i > j \geq 0)$$

O-A-VAR $$\Gamma_x[x{:}A]\Gamma^x \vdash x \Rightarrow x, \mathsf{cum}(A, i) \qquad (i \geq 0)$$

O-A-GEN $$\frac{\Gamma \vdash Q \Rightarrow A, K \quad K \stackrel{wh}{\twoheadrightarrow} \kappa_1 \quad x \notin \mathrm{Dom}(\Gamma) \quad \Gamma[x{:}A] \vdash R \Rightarrow B, L \quad L \stackrel{wh}{\twoheadrightarrow} \kappa_2}{\Gamma \vdash \{x{:}Q\}R \Rightarrow \{x{:}A\}B, \kappa_1 \uparrow_i \kappa_2} \qquad (i \geq 0)$$

O-A-ABS $$\frac{\Gamma \vdash Q \Rightarrow A, K \quad K \stackrel{wh}{\twoheadrightarrow} \kappa \quad x \notin \mathrm{Dom}(\Gamma) \quad \Gamma[x{:}A] \vdash R \Rightarrow M, B}{\Gamma \vdash [x{:}Q]R \Rightarrow [x{:}A]M, \{x{:}A\}B}$$

O-A-APP $$\frac{\Gamma \vdash Q \Rightarrow M, A \quad A \stackrel{wh}{\twoheadrightarrow} \{x{:}A_1\}A_2 \quad \Gamma \vdash R \Rightarrow N, B \quad B \downarrow A_1}{\Gamma \vdash QR \Rightarrow MN, \mathsf{cum}([N/x]A_2, i)} \qquad (i \geq 0)$$

where cum *and* $\uparrow$ *are as in Table 4.*

Table 7: Operational Presentation of CC^ω with Anonymous Universes

Proof

(Soundness) *The proof is by induction on the height of a derivation δ of $\Gamma \vdash Q \Rightarrow M, A$, with a case analysis on the last rule used in δ. For example, suppose that δ has the form*

$$\frac{\begin{array}{cccc}\vdots & & \vdots & \\ \Gamma \vdash Q \Rightarrow M, A & A \overset{wh}{\twoheadrightarrow} \{x{:}A_1\}A_2 & \Gamma \vdash R \Rightarrow N, B & B \downarrow A_1\end{array}}{\Gamma \vdash QR \Rightarrow MN, \mathsf{cum}([N/x]A_2, i)}$$

Then, by induction, M is a reading of Q, N is a reading of R, $\Gamma \vdash M \Rightarrow A$, and $\Gamma \vdash N \Rightarrow B$. The result follows by rule O-APP *and by noting that MN is a reading of QR. The remaining cases are handled similarly.*

(Completeness) *The proof is by induction on the height of a derivation δ of $\Gamma \vdash M \Rightarrow A$ where M is a reading of an ambiguous term Q. For example, if $Q = \mathsf{Type}$, $M = \mathsf{Type}_i$, and δ is an instance of* O-TYPE, *deriving $\Gamma \vdash \mathsf{Type}_i \Rightarrow \mathsf{Type}_j$ for some $j > i$, then by rule* O-ANON, *we obtain $\Gamma \vdash \mathsf{Type} \Rightarrow \mathsf{Type}_i, \mathsf{Type}_j$, as desired. The remaining cases follow easily by induction.*

5.2 Type Checking with Anonymous Universes

Decision procedures for the type checking and well-typedness problems are once again based on a reduction to schematic type synthesis. Level variables are used in two distinct ways: to encode the flexibility due to cumulativity in the type of a term (as before), and to govern the set of possible readings of an ambiguous term. This second use of level variables must also be regulated by constraint sets since the set of correct readings for an ambiguous term is constrained by the context in which that term occurs.

The type synthesis algorithm is presented in Table 8 as a set of rules for deriving assertions of the form $\Phi, \mathcal{C} \vdash Q \Rightarrow X, Y, \mathcal{D}$. The pair $(\Phi, \mathcal{C})$ is a *schematic context*, where Φ is a context built from declarations of the form $x{:}X$ where X is a schematic term such that $\mathrm{LV}(X) \subseteq \mathrm{LV}(\mathcal{C})$. Anonymous universes may not occur in any declaration in Φ. If $(\Phi, \mathcal{C})$ is a schematic context and $\sigma \models \mathcal{C}$, then $\sigma\Phi$ is the ordinary context obtained by replacing

A-A-PROP $$\Phi, \mathcal{C} \vdash \mathsf{Prop} \Rightarrow \mathsf{Prop}, \mathsf{Type}_\alpha, \{\, \alpha \geq 0 \,\} \qquad (\alpha \ new)$$

A-A-TYPE $$\Phi, \mathcal{C} \vdash \mathsf{Type}_j \Rightarrow \mathsf{Type}_j, \mathsf{Type}_\alpha, \{\, \alpha > j \,\} \qquad (\alpha \ new)$$

A-A-ANON $$\Phi, \mathcal{C} \vdash \mathsf{Type} \Rightarrow \mathsf{Type}_\beta, \mathsf{Type}_\alpha, \{\, \alpha > \beta \geq 0 \,\} \qquad (\alpha, \beta \ new)$$

A-A-VAR $$\Phi_x[x{:}X]\Phi^x, \mathcal{C} \vdash x \Rightarrow x, \mathsf{CUM}(X, \emptyset)$$

A-A-GEN $$\frac{\begin{array}{c}\Phi, \mathcal{C} \vdash Q \Rightarrow U, X, \mathcal{D} \quad X \stackrel{wh}{\twoheadrightarrow} \kappa_1 \quad \mathcal{C} \cup \mathcal{D} \text{ consistent} \\ \Phi[x{:}U], \mathcal{C} \cup \mathcal{D} \vdash R \Rightarrow V, Y, \mathcal{E} \quad Y \stackrel{wh}{\twoheadrightarrow} \kappa_2 \quad x \notin \mathrm{Dom}(\Phi)\end{array}}{\Phi, \mathcal{C} \vdash \{x{:}Q\}R \Rightarrow \{x{:}U\}V, \kappa_1 \Uparrow_{\mathcal{D} \cup \mathcal{E}} \kappa_2}$$

A-A-ABS $$\frac{\begin{array}{c}\Phi, \mathcal{C} \vdash Q \Rightarrow U, X, \mathcal{D} \quad X \stackrel{wh}{\twoheadrightarrow} \kappa \quad \mathcal{C} \cup \mathcal{D} \text{ consistent} \\ \Phi[x{:}U], \mathcal{C} \cup \mathcal{D} \vdash R \Rightarrow V, Y, \mathcal{E} \quad x \notin \mathrm{Dom}(\Phi)\end{array}}{\Phi, \mathcal{C} \vdash [x{:}Q]R \Rightarrow [x{:}U]V, \{x{:}U\}Y, \mathcal{D} \cup \mathcal{E}}$$

A-A-APP $$\frac{\begin{array}{c}\Phi, \mathcal{C} \vdash Q \Rightarrow U, X, \mathcal{D} \quad X \stackrel{wh}{\twoheadrightarrow} \{x : X_1\}X_2 \\ \Phi, \mathcal{C} \vdash R \Rightarrow V, Y, \mathcal{E} \quad X_1 \downarrow Y(\mathcal{F})\end{array}}{\Phi, \mathcal{C} \vdash QR \Rightarrow UV, \mathsf{CUM}([V/x]X_2, \mathcal{D} \cup \mathcal{E} \cup \mathcal{F})}$$

where CUM *and* $\Uparrow$ *are as in Table 6.*

Table 8: Type Synthesis Algorithm for CC^ω with Anonymous Universes

each declaration $x{:}X$ in Φ by the declaration $x{:}\sigma X$. A schematic context $(\Phi, \mathcal{C})$ is said to be *valid* iff $\mathcal{C}$ is consistent and for every $\sigma \models \mathcal{C}$, $\sigma\Phi$ is valid.

Theorem 5.2 (Soundness) *If* $\Phi, \mathcal{C} \vdash Q \Rightarrow X, Y, \mathcal{D}$, *then*

1. $\mathrm{LV}(X) \subseteq \mathrm{LV}(\mathcal{D})$, $\mathrm{LV}(Y) \subseteq \mathrm{LV}(\mathcal{C} \cup \mathcal{D})$, *and* $\mathrm{LV}(\mathcal{D}) \setminus \mathrm{LV}(\mathcal{C})$ *is a set of "new" level variables.*
2. *If* $\sigma \models \mathcal{C} \cup \mathcal{D}$, *then* $\sigma\Phi \vdash Q \Rightarrow \sigma X, \sigma Y$.

(Completeness) *If* $\sigma \models \mathcal{C}$ *with* $\mathrm{Dom}(\sigma) = \mathrm{LV}(\mathcal{C})$ *and* $\sigma\Phi \vdash Q \Rightarrow M, A$, *then there exists* $X, Y, \mathcal{D}$, *and* τ *such that*

1. $\Phi, \mathcal{C} \vdash Q \Rightarrow X, Y, \mathcal{D}$,
2. τ *extends* σ, $\tau \models \mathcal{D}$, $\mathrm{Dom}(\tau) = \mathrm{LV}(\mathcal{C} \cup \mathcal{D})$, *and*
3. $\tau X = M$, *and* $\tau Y = A$.

(Decidability) *It is decidable, given valid schematic context* $(\Phi, \mathcal{C})$ *and ambiguous term* Q, *whether or not there exists schematic terms* X *and* Y, *and constraint set* $\mathcal{D}$ *such that* $\Phi, \mathcal{C} \vdash Q \Rightarrow X, Y, \mathcal{D}$ *and* $\mathcal{C} \cup \mathcal{D}$ *is consistent.*

Proof

(Soundness) *The proof is by induction on the height of a derivation* δ *of* $\Phi, \mathcal{C} \vdash Q \Rightarrow X, Y, \mathcal{D}$. *The conditions on the level variables are all proved by inspection of the rules of Table 8, keeping in mind the conventions about "new" level variables. For the second claim consider, for example, the case in which* δ *has the form*

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Phi, \mathcal{C} \vdash Q \Rightarrow U, X, \mathcal{D} & \Phi, \mathcal{C} \vdash R \Rightarrow V, Y, \mathcal{E}\end{array}}{\Phi, \mathcal{C} \vdash QR \Rightarrow UV, Z, \mathcal{G}}$$

where $X \overset{wh}{\twoheadrightarrow} \{x : X_1\}X_2$, $X_1 \downarrow Y(\mathcal{F})$, *and* $Z, \mathcal{G} = \mathsf{CUM}([V/x]X_2, \mathcal{D} \cup \mathcal{E} \cup \mathcal{F})$. *Since* $\mathcal{D} \cup \mathcal{E} \cup \mathcal{F} \subseteq \mathcal{G}$ *and* $\sigma \models \mathcal{C} \cup \mathcal{G}$, *we have by induction*

$$\sigma\Phi \vdash Q \Rightarrow \sigma U, \sigma X \qquad \textit{and} \qquad \sigma\Phi \vdash R \Rightarrow \sigma V, \sigma Y.$$

By Lemma 4.2, $\sigma X \stackrel{wh}{\twoheadrightarrow} \{x{:}\sigma X_1\}\sigma X_2$ *and by Theorem 4.3* $\sigma X_1 \downarrow \sigma Y$*, and hence by* O-A-APP*,*

$$\sigma\Phi \vdash QR \Rightarrow \sigma U\, \sigma V, \mathsf{cum}([\sigma V/x]\sigma X_2, i)$$

for any $i \geq 0$*. Now* $\sigma U\, \sigma V = \sigma(UV)$*, and* $[\sigma V/x]\sigma X_2 = \sigma[V/x]X_2$*, and so by Lemma 4.4, there exists* i *such that* $\sigma Z = \mathsf{cum}([\sigma V/x]\sigma X_2, i)$*, as desired.*

(Completeness) *The proof is by induction on the height of a derivation* δ *of* $\sigma\Phi \vdash Q \Rightarrow M, A$ *where* $\sigma \models \mathcal{C}$*. Suppose that* δ *is an instance of* O-A-ANON*; i.e.,* $\sigma\Phi \vdash \mathsf{Type} \Rightarrow \mathsf{Type}_j, \mathsf{Type}_i$ *where* $i > j$*. Choose* $X = \mathsf{Type}_\beta$*,* $Y = \mathsf{Type}_\alpha$*, and* $\mathcal{D} = \{\, \alpha > \beta \geq 0 \,\}$ *(where* α *and* β *are "new") to obtain the required derivation, and choose* $\tau = \sigma[\alpha \mapsto i, \beta \mapsto j]$ *as the required level assignment. It is easy to see that* $\tau \models \mathcal{C} \cup \mathcal{D}$*,* $\tau X = \mathsf{Type}_j$*, and* $\tau Y = \mathsf{Type}_i$*, as required.*

Now suppose that δ *is a derivation of the form*

$$\frac{\begin{array}{cc}\vdots & \vdots \\ \sigma\Phi \vdash Q \Rightarrow M, A & \sigma\Phi \vdash R \Rightarrow N, B\end{array}}{\sigma\Phi \vdash QR \Rightarrow MN, \mathsf{cum}([N/x]A_2, i)}$$

where $i \geq 0$*,* $A \stackrel{wh}{\twoheadrightarrow} \{x{:}A_1\}A_2$*, and* $B \downarrow A_1$*. By induction there exists* U*,* X*,* $\mathcal{D}$*, and an extension* τ_A *of* σ *such that*

$$\Phi, \mathcal{C} \vdash Q \Rightarrow U, X, \mathcal{D} \quad \tau_A \models \mathcal{D} \quad \tau_A U = M \quad \tau_A X = A.$$

Similarly, by induction there exists V*,* Y*,* $\mathcal{E}$*, and an extension* τ_B *of* σ *such that*

$$\Phi, \mathcal{C} \vdash R \Rightarrow V, Y, \mathcal{E} \quad \tau_B \models \mathcal{E} \quad \tau_B V = N \quad \tau_B Y = B.$$

Now by the "new" convention, $\mathrm{LV}(\mathcal{D}) \cap \mathrm{LV}(\mathcal{E}) \subseteq \mathrm{LV}(\mathcal{C})$*, and by the conditions on* τ_A *and* τ_B*, we may form the level assignment* $\tau_{A,B} = \tau_A \cup \tau_B$ *with the properties that* $\tau_{A,B}$ *is an extension of* σ*,* $\tau_{A,B} \models \mathcal{D} \cup \mathcal{E}$*,* $\tau_{A,B} U = M$*,* $\tau_{A,B} V = N$*,* $\tau_{A,B} X = A$*, and* $\tau_{A,B} Y = B$*. Hence, by Lemma 4.2, there exists* X_1 *and* X_2 *such that* $X \stackrel{wh}{\twoheadrightarrow} \{x{:}X_1\}X_2$ *with*

$\tau_{A,B}X_i = A_i$ $(i = 1, 2)$, and by Theorem 4.3, there exists $\mathcal{F}$ such that $X_1 \downarrow Y(\mathcal{F})$ and $\tau_{A,B} \models \mathcal{F}$. Therefore by A-A-APP *we have*

$$\Phi, \mathcal{C} \vdash QR \Rightarrow UV, Z, \mathcal{G}.$$

where $Z, \mathcal{G} = \mathsf{CUM}([V/x]X_2, \mathcal{D} \cup \mathcal{E} \cup \mathcal{F})$. Now $[N/x]A_2 = [\tau_{A,B}V/x]\tau_{A,B}X_2 = \tau_{A,B}[V/x]X_2$, and so by Lemma 4.4, there exists $\tau \models \mathcal{G}$ extending $\tau_{A,B}$ such that $\tau Z = \mathsf{cum}(\tau[V/x]X_2, i) = \mathsf{cum}([N/x]A_2, i)$, which completes the proof.

(Decidability) *Similar to the proof of Theorem 4.5.* □

6 Definitions

In this section we treat the extension of the two calculi (CC^ω and CC^ω with anonymous universes) to admit defined identifiers. We take as a fundamental principle the eliminability of definitions: a defined identifier is indistinguishable from its definition. This principle leads to the notion of *universe polymorphism*, whereby a defined identifier may take on any of some constrained set of types determined by its definition. Since this form of polymorphism is associated with definitions, it is very similar to the form of polymorphism found in ML [39, 15, 40]. For the sake of simplicity we omit consideration of local definitions (which would be introduced using a form of *let* expression). However, we expect that the methods described below can be extended to handle this case.

6.1 Definitions in CC^ω

A *definitional context* Δ is a finite sequence of *declarations* of the form $x{:}A$ and *definitions* of the form $x{=}M$ subject to the following conditions. First, no variable may be declared or defined more than once, nor may any variable be both declared and defined. Thus a definitional context Δ has a well-defined domain given by $\mathrm{Dom}(\Delta) = \mathrm{Def}(\Delta) \cup \mathrm{Dec}(\Delta)$ where $\mathrm{Def}(\Delta)$ is the set of defined variables in Δ, and $\mathrm{Dec}(\Delta)$ is its set of declared variables. Second, if $\Delta = \Delta_x[x{:}M]\Delta^x$, or $\Delta = \Delta_x[x{=}M]\Delta^x$, then $\mathrm{FV}(M) \subseteq \mathrm{Dom}(\Delta_x)$. (In the only interesting case, valid contexts, this is no restriction at all.) Third, as a matter of technical convenience, we require that no defined variables

occur in the right-hand side of a definition. This convention avoids certain complications in the proofs arising from the possibility of "definition chains" whereby an identifier x is defined to be y, itself a defined identifier.

In order to give expression to the eliminability of definitions, we shall need the *expansion function* defined by induction on the structure of terms as follows:

$$\begin{array}{rcl}
\Delta(x) & := & \begin{cases} M & \textit{if } \Delta = \Delta_x[x{=}M]\Delta^x \\ x & \textit{otherwise} \end{cases} \\
\Delta(\kappa) & := & \kappa \\
\Delta(\{x{:}A\}B) & := & \{x{:}\Delta(A)\}\Delta(B) \quad \textit{if } x \notin \mathrm{Def}(\Delta) \\
\Delta([x{:}A]M) & := & [x{:}\Delta(A)]\Delta(M) \quad \textit{if } x \notin \mathrm{Def}(\Delta) \\
\Delta(M\ N) & := & \Delta(M)\ \Delta(N)
\end{array}$$

(We assume that bound variables are chosen so as to avoid conflicts, as necessary.) Expansion is extended to definitional contexts as follows:

$$\begin{array}{rcl}
|\bullet| & := & \bullet \\
|\Delta[x{:}A]| & := & |\Delta|[x{:}\Delta(A)] \\
|\Delta[x{=}M]| & := & |\Delta|
\end{array}$$

Define $\Delta \vdash M \downarrow N$ (Δ-conversion) to hold iff $\Delta(M) \downarrow \Delta(N)$. Since definition expansion is always terminating, this relation is decidable if $\Delta(M)$ and $\Delta(N)$ are strongly normalizing (as will be the case for well-typed terms). A direct definition of this relation could be given by taking a defined identifier to be convertible to its definition, along with the usual β-conversion axiom.

The relation $\Delta \vdash M \overset{wh}{\twoheadrightarrow} N$ (Δ-weak head reduction) is defined similarly to $M \overset{wh}{\twoheadrightarrow} N$ (see Section 2.2), taking

$$\Delta_x[x{=}M]\Delta^x \vdash x \overset{wh}{\rightarrow} M$$

as an added axiom in Table 1. Thus a term in Δ-weak head normal form has one of the shapes x (where $x \notin \mathrm{Def}(\Delta)$), κ, $\{x{:}A\}B$, $[x{:}A]B$, or $M\ N$ where M is itself a Δ-weak head normal form of shape other than $[x{:}A]B$.

Lemma 6.1

1. *If* $\Delta \vdash M \overset{wh}{\twoheadrightarrow} N$, *then* $\Delta(M) \overset{wh}{\twoheadrightarrow} \Delta(N)$.

O-D-PROP $$\Delta \vdash \mathsf{Prop} \Rightarrow \mathsf{Type}_i \qquad (i \geq 0)$$

O-D-TYPE $$\Delta \vdash \mathsf{Type}_j \Rightarrow \mathsf{Type}_i \qquad (i > j \geq 0)$$

O-D-VAR $$\Delta_x[x{:}A]\Delta^x \vdash x \Rightarrow \mathsf{cum}_{\Delta_x}(A, i) \qquad (i \geq 0)$$

O-D-DEF $$\frac{\Delta_x \vdash M \Rightarrow A}{\Delta_x[x{=}M]\Delta^x \vdash x \Rightarrow A}$$

O-D-GEN $$\frac{\begin{array}{c}\Delta \vdash A \Rightarrow K \quad \Delta \vdash K \stackrel{wh}{\twoheadrightarrow} \kappa_1 \quad x \notin \mathrm{Dom}(\Delta) \\ \Delta[x{:}A] \vdash B \Rightarrow L \quad \Delta \vdash L \stackrel{wh}{\twoheadrightarrow} \kappa_2\end{array}}{\Delta \vdash \{x{:}A\}B \Rightarrow \kappa_1 \uparrow_i \kappa_2} \qquad (i \geq 0)$$

O-D-ABS $$\frac{\begin{array}{c}\Delta \vdash A \Rightarrow K \quad \Delta \vdash K \stackrel{wh}{\twoheadrightarrow} \kappa \quad x \notin \mathrm{Dom}(\Delta) \\ \Delta[x{:}A] \vdash M \Rightarrow B\end{array}}{\Delta \vdash [x{:}A]M \Rightarrow \{x{:}A\}B}$$

O-D-APP $$\frac{\begin{array}{c}\Delta \vdash M \Rightarrow A \quad \Delta \vdash A \stackrel{wh}{\twoheadrightarrow} \{x{:}A_1\}A_2 \\ \Delta \vdash N \Rightarrow B \quad \Delta \vdash B \downarrow A_1\end{array}}{\Delta \vdash MN \Rightarrow \mathsf{cum}_\Delta([N/x]A_2, i)} \qquad (i \geq 0)$$

where $\uparrow$ *is as in Table 4 and* cum_Δ *is defined by* $\mathsf{cum}_\Delta(A, i) = \mathsf{cum}(\Delta(A), i)$.

Table 9: Operational Presentation of CC^ω with Definitions

2. *If* $\Delta(M) \stackrel{wh}{\twoheadrightarrow} N$, *then there exists* P *such that* $\Delta \vdash M \stackrel{wh}{\twoheadrightarrow} P$ *and* $\Delta(P) = N$.

An extension of CC^ω to admit definitions is presented in operational style in Table 9. This system is essentially Table 4 extended with the rule O-D-DEF. This rule expresses the eliminability of definitions principle, and introduces the notion of universe polymorphism. For example, the following assertion is derivable in the system of Table 9:

$$[x{=}[y{:}\mathsf{Prop}]\mathsf{Prop}] \vdash ([z{:}\mathsf{Prop}{\to}\mathsf{Type}_0][w{:}\mathsf{Prop}{\to}\mathsf{Type}_1]\mathsf{Prop})\ x\ x \Rightarrow \mathsf{Type}_0$$

In this derivation the two instances of x are given the distinct types $\mathsf{Prop}{\to}\mathsf{Type}_0$ and $\mathsf{Prop}{\to}\mathsf{Type}_1$, both of which are correct types for $[y{:}\mathsf{Prop}]\mathsf{Prop}$. Notice that no single type for x will do.

The soundness of the operational system with definitions (Table 9) with respect to the basic operational system (Table 4) is expressed by the following theorem.

Theorem 6.2 *If* $\Delta \vdash M \Rightarrow A$, *then* $|\Delta| \vdash \Delta(M) \Rightarrow \Delta(A)$.

Proof *The proof is by induction on the height of a derivation* δ *of* $\Delta \vdash M \Rightarrow A$. *Suppose that* δ *is a derivation of the form*

$$\frac{\begin{array}{c}\vdots\\ \Delta_x \vdash M \Rightarrow A\end{array}}{\Delta_x[x{=}M]\Delta^x \vdash x \Rightarrow A}$$

By induction hypothesis we have

$$|\Delta_x| \vdash \Delta_x(M) \Rightarrow \Delta_x(A).$$

Now $\mathrm{FV}(A) \subseteq \mathrm{Dom}(\Delta)$, *and so* $\Delta_x(A) = \Delta(A)$. *By the conditions on definitional contexts,* $\Delta(M) = \Delta_x(M)$, *and by the definition of* $|-|$, $|\Delta|$ *is an extension of* $|\Delta_x|$. *It is easy to see that derivability is closed under context extension, and therefore we may derive* $|\Delta| \vdash \Delta(x) \Rightarrow \Delta(A)$, *as desired.*

Suppose, now, that δ *is a derivation of the form*

$$\frac{\begin{array}{cc}\vdots & \vdots\\ \Delta \vdash M \Rightarrow A & \Delta \vdash N \Rightarrow B\end{array}}{\Delta \vdash MN \Rightarrow \mathsf{cum}_\Delta([N/x]A_2, i)}$$

where $i \geq 0$, $\Delta \vdash A \stackrel{wh}{\twoheadrightarrow} \{x{:}A_1\}A_2$, *and* $\Delta \vdash B \downarrow A_1$. *By the conventions on bound variables, we may assume that* $x \notin \mathrm{Def}(\Delta)$. *By induction we obtain*

$$|\Delta| \vdash \Delta(M) \Rightarrow \Delta(A) \quad \textit{and} \quad |\Delta| \vdash \Delta(N) \Rightarrow \Delta(B).$$

By Lemma 6.1, $\Delta(A) \stackrel{wh}{\twoheadrightarrow} \Delta(\{x{:}A_1\}A_2) = \{x{:}\Delta(A_1)\}\Delta(A_2)$. *Moreover* $\Delta(B) \downarrow \Delta(A_1)$ *by the definition of* $\downarrow$, *so by rule* O-APP *we obtain*

$$|\Delta| \vdash \Delta(M)\Delta(N) \Rightarrow \mathsf{cum}([\Delta(N)/x]\Delta(A_2), i),$$

and the result follows easily from the definition of cum_Δ. *The remaining cases are handled similarly.* □

A converse to this theorem may also be proved, yielding as a corollary the decidability of the system with definitions. We prefer, however, to give a direct presentation of a type synthesis algorithm that avoids unnecessary expansion of definitions. The idea is to adapt the methods of [15], and associate with a definition a type scheme summarizing the set of all possible types for the definiens. Under suitable assumptions about the associated type scheme, we obtain a sound and complete type synthesis algorithm for CC^ω with definitions.

A *generic definitional context* Θ is defined similarly to a definitional context, except that definitions have the form $x{=}M{:}X, \mathcal{G}$, where X is a schematic term, and $\mathcal{G}$ is a constraint set such that $\mathrm{LV}(X) \subseteq \mathrm{LV}(\mathcal{G})$. By abuse of notation, we use Θ in situations where a definitional context is expected, under the convention that the stored schematic type information is to be ignored. When it is important to stress the distinction, we write $\hat{\Theta}$ for the underlying definitional context of Θ.

Δ-conversion and Δ-weak head reduction are extended to schematic terms in the obvious way, following the pattern of Section 4.1.

A definitional context Δ is *valid* iff for each x such that $\Delta = \Delta_x[x{:}A]\Delta^x$, there exists some K such that $\Delta_x \vdash A \Rightarrow K$ with $\Delta \vdash K \stackrel{wh}{\twoheadrightarrow} \kappa$, and for each x such that $\Delta = \Delta_x[x = M]\Delta^x$ there exists some A such that $\Delta_x \vdash M \Rightarrow A$. Thus types in declarations must indeed be types, and the definiens of a definition must be well-formed. A generic definitional context Θ is *valid* iff $\hat{\Theta}$ is valid and for each x such that $\Theta = \Theta_x[x{=}M{:}X, \mathcal{G}]\Theta^x$, the constraint set $\mathcal{G}$ is satisfiable, and $\hat{\Theta}_x \vdash M \Rightarrow \sigma X$ whenever $\sigma \models \mathcal{G}$ (that is, $(X, \mathcal{G})$ must

have an instance, and all instances must be valid types for M). Conversely, Θ is *principal* iff it is valid, and whenever $\Theta = \Theta_x[x{=}M{:}X, \mathcal{G}]\Theta^x$ and $\hat{\Theta}_x \vdash M \Rightarrow A$, then there exists $\sigma \models \mathcal{G}$ such that $\sigma X = A$ (that is, the type scheme $(X, \mathcal{G})$ must capture all valid types for M). It is worth remarking that these conditions are naturally preserved under the extension of the system with local definitions.

The type synthesis algorithm for CC^ω with definitions is given in Table 10. The rule A-D-DEF makes use of an operation $\nu_{\mathcal{V}}$ mapping level variables in the set $\mathcal{V}$ to "new" level variables (*i.e.*, that do not otherwise appear in the derivation.) This operation is extended to schematic terms and constraint sets in the obvious way. The properties of the type synthesis algorithm are give by the following theorem:

Theorem 6.3

(Soundness) *If* $\Theta \vdash M \Rightarrow X, \mathcal{C}$ *for some valid definitional context* Θ*, then*

1. $\mathrm{LV}(X) \subseteq \mathrm{LV}(\mathcal{C})$ *and* $\mathrm{LV}(\mathcal{C})$ *is a set of "new" level variables.*
2. *If* $\sigma \models \mathcal{C}$*, then* $\hat{\Theta} \vdash M \Rightarrow \sigma X$*.*

(Completeness) *If* $\hat{\Theta} \vdash M \Rightarrow A$ *for some principal definitional context* Θ*, then there exists* X*,* $\mathcal{C}$*, and* σ *such that*

1. $\Theta \vdash M \Rightarrow X, \mathcal{C}$*,*
2. $\sigma \models \mathcal{C}$*, and* $\mathrm{Dom}(\sigma) = \mathrm{LV}(\mathcal{C})$
3. $\sigma X = A$*.*

(Decidability) *It is decidable, given valid definitional context* Θ *and term* M*, whether or not there exists schematic term* X *and consistent constraint set* $\mathcal{C}$ *such that* $\Theta \vdash M \Rightarrow X, \mathcal{C}$*.*

Proof *The proof is essentially the same as that of Theorem 4.5; the assumptions on* Θ *suffice for defined identifiers.* □

A-D-PROP $$\Theta \vdash \mathsf{Prop} \Rightarrow \mathsf{Type}_\alpha, \{\, \alpha \geq 0 \,\} \qquad (\alpha \ new)$$

A-D-TYPE $$\Theta \vdash \mathsf{Type}_i \Rightarrow \mathsf{Type}_\alpha, \{\, \alpha > i \,\} \qquad (\alpha \ new)$$

A-D-VAR $$\Theta_x[x{:}A]\Theta^x \vdash x \Rightarrow \mathsf{CUM}_{\Theta_x}(A, \emptyset)$$

A-D-DEF $$\Theta_x[x{=}M{:}X, \mathcal{G}]\Theta^x \vdash x \Rightarrow \nu_{\mathrm{LV}(\mathcal{G})}(X, \mathcal{G})$$

A-D-GEN $$\frac{\Theta \vdash A \Rightarrow X, \mathcal{C} \quad \Theta \vdash X \stackrel{wh}{\twoheadrightarrow} \kappa_1 \quad x \notin \mathrm{Dom}(\Theta) \quad \mathcal{C} \text{ consistent} \qquad \Theta[x{:}A] \vdash B \Rightarrow Y, \mathcal{D} \quad \Theta \vdash Y \stackrel{wh}{\twoheadrightarrow} \kappa_2}{\Theta \vdash \{x{:}A\}B \Rightarrow \kappa_1 \Uparrow_{\mathcal{C}\cup\mathcal{D}} \kappa_2}$$

A-D-ABS $$\frac{\Theta \vdash A \Rightarrow X, \mathcal{C} \quad \Theta \vdash X \stackrel{wh}{\twoheadrightarrow} \kappa \quad x \notin \mathrm{Dom}(\Theta) \quad \mathcal{C} \text{ consistent} \qquad \Theta[x{:}A] \vdash M \Rightarrow Y, \mathcal{D}}{\Theta \vdash [x{:}A]M \Rightarrow \{x{:}A\}Y, \mathcal{C} \cup \mathcal{D}}$$

A-D-APP $$\frac{\Theta \vdash M \Rightarrow X, \mathcal{C} \quad \Theta \vdash X \stackrel{wh}{\twoheadrightarrow} \{x : X_1\}X_2 \qquad \Theta \vdash N \Rightarrow Y, \mathcal{D} \quad \Theta \vdash X_1 \downarrow Y(\mathcal{E})}{\Theta \vdash MN \Rightarrow \mathsf{CUM}_\Theta([N/x]X_2, \mathcal{C} \cup \mathcal{D} \cup \mathcal{E})}$$

where $\nu_\mathcal{V}$ assigns "new" level variables to each of the level variables in $\mathcal{V}$, $\Uparrow$ is as in Table 6, and CUM_Θ is defined by $\mathsf{CUM}_\Theta(X, \mathcal{C}) = \mathsf{CUM}(\Theta(X), \mathcal{C})$.

Table 10: Type Synthesis Algorithm for CC$^\omega$ with Definitions

6.2 Definitions with Anonymous Universes

The final extension of CC^ω that we shall consider is the combination of universe polymorphism and typical ambiguity. As we have seen, definitions introduce a form of polymorphism induced by the cumulativity of the universe hierarchy. "Ambiguous" definitions allow for further flexibility since the definiens is "re-read" on each use of the definition. For example, if f is defined to be the term $[x{:}\mathsf{Type}]x$, then both $f\ \mathsf{Prop}$ and $f\ \mathsf{Type}_0$ are well-formed, as is $f\ \mathsf{Type}$, since in each case the ambiguous definition of f receives a reading appropriate to the context. Moreover, each of these terms could occur as subterms of a single term: the principle of eliminability implies that defined identifiers are "polymorphic" in that each occurrence corresponds to a distinct "reading" of the definition. An interesting example is self-application of the polymorphic identity function. With the definition

$$I{=}[t{:}\mathsf{Type}][x{:}t]x$$

the term

$$I\ (\{t{:}\mathsf{Type}\}t{\to}t)\ I$$

is well typed; the two instances of I receive two distinct readings, and are assigned two distinct types.

It is also important to realize that this notion of polymorphism does not extend to declarations A declaration $x{:}A$ or $x{:}X$ assigns a *single*, perhaps underdetermined, type for x. Thus (referring back to the polymorphic identity example above) if J is declared by

$$J{:}\{t{:}\mathsf{Type}\}t{\to}t$$

then

$$J\ (\{t{:}\mathsf{Type}\}t{\to}t)\ J$$

should *not* be well typed[2]. This is as it should be, for there is no reading of the type of J such that the above term is well-typed in CC^ω.

These considerations are formalized in an operational presentation in Table 11. The rules of Table 11 are essentially a combination of those of Tables 7 and 9. Note that the type of variables receive a fixed "reading" before being

[2] We thank Gérard Huet for this example

O-AD-PROP $$\Delta \vdash \mathsf{Prop} \Rightarrow \mathsf{Prop}, \mathsf{Type}_i \qquad (i \geq 0)$$

O-AD-TYPE $$\Delta \vdash \mathsf{Type}_j \Rightarrow \mathsf{Type}_j, \mathsf{Type}_i \qquad (i > j \geq 0)$$

O-AD-ANON $$\Delta \vdash \mathsf{Type} \Rightarrow \mathsf{Type}_j, \mathsf{Type}_i \qquad (i > j \geq 0)$$

O-AD-VAR $$\Delta_x[x{:}A]\Delta^x \vdash x \Rightarrow x, \mathsf{cum}(A, i) \qquad (i \geq 0)$$

O-AD-DEF $$\frac{\Delta_x \vdash Q \Rightarrow M, A}{\Delta_x[x{=}Q]\Delta^x \vdash x \Rightarrow M, A}$$

O-AD-GEN $$\frac{\Delta \vdash Q \Rightarrow A, K \quad K \stackrel{wh}{\twoheadrightarrow} \kappa_1 \quad x \notin \mathrm{Dom}(\Delta) \quad \Delta[x{:}A] \vdash R \Rightarrow B, L \quad L \stackrel{wh}{\twoheadrightarrow} \kappa_2}{\Delta \vdash \{x{:}Q\}R \Rightarrow \{x{:}A\}B, \kappa_1 \uparrow_i \kappa_2} \qquad (i \geq 0)$$

O-AD-ABS $$\frac{\Delta \vdash Q \Rightarrow A, K \quad K \stackrel{wh}{\twoheadrightarrow} \kappa \quad x \notin \mathrm{Dom}(\Delta) \quad \Delta[x{:}A] \vdash R \Rightarrow M, B}{\Delta \vdash [x{:}Q]R \Rightarrow [x{:}A]M, \{x{:}A\}B}$$

O-AD-APP $$\frac{\Delta \vdash Q \Rightarrow M, A \quad A \stackrel{wh}{\twoheadrightarrow} \{x{:}A_1\}A_2 \quad \Delta \vdash R \Rightarrow N, B \quad B \downarrow A_1}{\Delta \vdash QR \Rightarrow MN, \mathsf{cum}([N/x]A_2, i)} \qquad (i \geq 0)$$

where cum *and* $\uparrow$ *are as in Table 4*

Table 11: Operational Presentation of Anonymous Universes and Definitions

added to the context, so that declarations are always unambiguous. Definitions, on the other hand, may be ambiguous, and receive a fresh "reading" on each use. In order to ensure that declared identifiers remain unambiguous even in the presence of definitions, we require that no defined identifiers occur in the type of any declared variable. This restriction is preserved by rules O-AD-GEN and O-AD-ABS, since the reading of a term has all defined identifiers eliminated.

The soundness of this system with respect to the operational presentation of anonymous universes (Table 7) is expressed by the following theorem.

Theorem 6.4 *If* $\Delta \vdash Q \Rightarrow M, A$, *then* $|\Delta| \vdash \Delta(Q) \Rightarrow M, A$.
Proof *The proof is by induction on the height of a derivation* δ *of* $\Delta \vdash Q \Rightarrow M, A$. *The most interesting case is when* δ *has the form:*

$$\dfrac{\begin{array}{c}\vdots \\ \Delta_x \vdash Q \Rightarrow M, A\end{array}}{\Delta_x[x{=}Q]\Delta^x \vdash x \Rightarrow M, A}$$

By induction, $|\Delta_x| \vdash \Delta_x(Q) \Rightarrow M, A$. *Clearly* $|\Delta_x[x{=}Q]\Delta^x|$ *is an extension of* $|\Delta_x|$ *and* $\Delta_x(Q) = \Delta(x)$, *so* $|\Delta| \vdash \Delta(x) \Rightarrow M, A$, *as required. The remaining cases are similar.* □

Once again, a converse to this theorem could be proved, with decidability following as a corollary. However, we prefer to give a direct presentation of a type synthesis algorithm for the system with ambiguous terms and definitions.

A *schematic, generic definitional context (sgd-context)* is a pair $(\Psi, \mathcal{C})$ where Ψ is a context built from declarations of the form $x{:}X$ with $\mathrm{LV}(X) \subseteq \mathrm{LV}(\mathcal{C})$, and definitions of the form $x{=}X{:}Y, \mathcal{G}$ where $\mathrm{LV}(X) \subseteq \mathrm{LV}(\mathcal{G}) \setminus \mathrm{LV}(\mathcal{C})$ and $\mathrm{LV}(Y) \subseteq \mathrm{LV}(\mathcal{C} \cup \mathcal{G})$. Note that the anonymous universe cannot occur in schematic terms: X and Y in the foregoing are unambiguous. The conditions regarding well-formedness of definitional contexts apply to sgd-contexts as well.

The type synthesis algorithm for anonymous universes and definitions is given in Table 12.

To state the soundness and completeness of the type synthesis algorithm, it is necessary to introduce some additional terminology. If X is a schematic

A-AD-PROP $\quad \Psi, \mathcal{C} \vdash \mathsf{Prop} \Rightarrow \mathsf{Prop}, \mathsf{Type}_\alpha, \{\, \alpha \geq 0 \,\} \qquad (\alpha \ new)$

A-AD-TYPE $\quad \Psi, \mathcal{C} \vdash \mathsf{Type}_j \Rightarrow \mathsf{Type}_j, \mathsf{Type}_\alpha, \{\, \alpha > j \,\} \qquad (\alpha \ new)$

A-AD-ANON $\quad \Psi, \mathcal{C} \vdash \mathsf{Type} \Rightarrow \mathsf{Type}_\beta, \mathsf{Type}_\alpha, \{\, \alpha > \beta \geq 0 \,\} \qquad (\alpha, \beta \ new)$

A-AD-VAR $\quad \Psi_x[x{:}X]\Psi^x, \mathcal{C} \vdash x \Rightarrow x, \mathsf{CUM}(X, \emptyset)$

A-AD-DEF $\quad \Psi_x[x{=}X{:}Y, \mathcal{G}]\Psi^x, \mathcal{C} \vdash x \Rightarrow \nu_{\mathrm{LV}(\mathcal{G})\setminus\mathrm{LV}(\mathcal{C})}(X, Y, \mathcal{G})$

A-AD-GEN $\quad \dfrac{\begin{array}{c}\Psi, \mathcal{C} \vdash Q \Rightarrow U, X, \mathcal{D} \quad X \stackrel{wh}{\twoheadrightarrow} \kappa_1 \quad \mathcal{C} \cup \mathcal{D} \text{ consistent} \\ \Psi[x{:}U], \mathcal{C} \cup \mathcal{D} \vdash R \Rightarrow V, Y, \mathcal{E} \quad Y \stackrel{wh}{\twoheadrightarrow} \kappa_2 \quad x \notin \mathrm{Dom}(\Psi)\end{array}}{\Psi, \mathcal{C} \vdash \{x{:}Q\}R \Rightarrow \{x{:}U\}V, \kappa_1 \Uparrow_{\mathcal{D}\cup\mathcal{E}} \kappa_2}$

A-AD-ABS $\quad \dfrac{\begin{array}{c}\Psi, \mathcal{C} \vdash Q \Rightarrow U, X, \mathcal{D} \quad X \stackrel{wh}{\twoheadrightarrow} \kappa \quad \mathcal{C} \cup \mathcal{D} \text{ consistent} \\ \Psi[x{:}U], \mathcal{C} \cup \mathcal{D} \vdash R \Rightarrow V, Y, \mathcal{E} \quad x \notin \mathrm{Dom}(\Psi)\end{array}}{\Psi, \mathcal{C} \vdash [x{:}Q]R \Rightarrow [x{:}U]V, \{x{:}U\}Y, \mathcal{D} \cup \mathcal{E}}$

A-AD-APP $\quad \dfrac{\begin{array}{c}\Psi, \mathcal{C} \vdash Q \Rightarrow U, X, \mathcal{D} \quad X \stackrel{wh}{\twoheadrightarrow} \{x : X_1\}X_2 \\ \Psi, \mathcal{C} \vdash R \Rightarrow V, Y, \mathcal{E} \quad X_1 \downarrow Y(\mathcal{F})\end{array}}{\Psi, \mathcal{C} \vdash QR \Rightarrow UV, \mathsf{CUM}([V/x]X_2, \mathcal{D} \cup \mathcal{E} \cup \mathcal{F})}$

where $\Uparrow$ *and* CUM *are as defined in Table 6, and* $\nu_{\mathcal{V}}$ *is as defined in Table 10.*

Table 12: Algorithm for Anonymous Universes and Definitions

term, let $\hat{X}$ denote the ambiguous term resulting from replacing all occurrences of Type_α by the anonymous universe Type. If $(\Psi,\mathcal{C})$ is an sgd-context and $\sigma \models \mathcal{C}$, then $\widehat{\sigma\Psi}$ denotes the definitional context obtained by

1. replacing each declaration $x{:}X$ by the declaration $x{:}\sigma X$, and
2. replacing each definition $x{=}X{:}Y,\mathcal{G}$ by the definition $x{=}\hat{X}$.

The idea is that the constraint set $\mathcal{C}$ governs the possible readings of of each of the declarations in Ψ, and that in each definition $x{=}X{:}Y,\mathcal{G}$, the level variables in X result from a schematic reading of some ambiguous term, and hence are erased in passing to the underlying definitional context. The type information associated with a definition is, as before, simply erased.

An sgd-context $(\Psi,\mathcal{C})$ is *valid* iff $\mathcal{C}$ is satisfiable, and if $\sigma \models \mathcal{C}$, then

1. if $\Psi = \Psi_x[x{:}X]\Psi^x$, then $\widehat{\sigma\Psi}_x \vdash \sigma X \Rightarrow \sigma X, K$ for some $K \overset{wh}{\twoheadrightarrow} \kappa$,
2. if $\Psi = \Psi_x[x{=}X{:}Y,\mathcal{G}]\Psi^x$, then for all variants $(Y^*,\mathcal{G}^*) = \nu_{\mathrm{LV}(\mathcal{G})\setminus\mathrm{LV}(\mathcal{C})}(Y,\mathcal{G})$,
   (a) $\sigma\mathcal{G}^*$ is satisfiable,
   (b) if τ is an extension of σ such that $\tau \models \mathcal{G}^*$, then $\widehat{\sigma\Psi}_x \vdash \hat{X} \Rightarrow \tau X, \tau Y$.

Intuitively, in a definition $x{=}X{:}Y,\mathcal{G}$ the constraint set $\mathcal{G}$ is to govern both the set of possible readings for X and the set of possible types Y for each reading. Since X may mention variables declared earlier in the context, the type Y might involve level variables in the constraint set $\mathcal{C}$.

Conversely, $(\Psi,\mathcal{C})$ is *principal* iff it is valid and if $\sigma \models \mathcal{C}$ with $\mathrm{Dom}(\sigma) = \mathrm{LV}(\mathcal{C})$, then for each x such that $\Psi = \Psi_x[x{=}X{:}Y,\mathcal{G}]\Psi^x$, if $\widehat{\sigma\Psi}_x \vdash \hat{X} \Rightarrow M, A$, then for each variant $(Y^*,\mathcal{G}^*) = \nu_{\mathrm{LV}(\mathcal{G})\setminus\mathrm{LV}(\mathcal{C})}(Y,\mathcal{G})$, there exists an extension τ of σ such that $\tau \models \mathcal{G}^*$, $\tau X = M$, and $\tau Y = A$. The idea is that every possible reading, and every possible type for each reading, of $\hat{X}$ is obtainable as an instance of the schematic definition and stored type information.

Theorem 6.5

(Soundness) *Let $(\Psi,\mathcal{C})$ be a valid sgd-context. If $\Psi,\mathcal{C} \vdash Q \Rightarrow X, Y, \mathcal{D}$, then*

1. $\mathrm{LV}(X) \subseteq \mathrm{LV}(\mathcal{D}) \setminus \mathrm{LV}(\mathcal{C})$, $\mathrm{LV}(Y) \subseteq \mathrm{LV}(\mathcal{C} \cup \mathcal{D})$, *and* $\mathrm{LV}(\mathcal{D}) \setminus \mathrm{LV}(\mathcal{C})$ *is a set of "new" level variables.*
2. *If* $\sigma \models \mathcal{C} \cup \mathcal{D}$, *then* $\widehat{\sigma\Psi} \vdash Q \Rightarrow \sigma X, \sigma Y$.

(Completeness) *Let* $(\Psi, \mathcal{C})$ *be a principal sgd-context. If* $\sigma \models \mathcal{C}$ *with* $\mathrm{Dom}(\sigma) = \mathrm{LV}(\mathcal{C})$, *and* $\widehat{\sigma\Psi} \vdash Q \Rightarrow M, A$, *then there exists* X, Y, $\mathcal{D}$, *and* τ *such that*

1. $\Psi, \mathcal{C} \vdash Q \Rightarrow X, Y, \mathcal{D}$,
2. τ *extends* σ, $\tau \models \mathcal{D}$, $\mathrm{Dom}(\tau) = \mathrm{LV}(\mathcal{C} \cup \mathcal{D})$, *and*
3. $\tau X = N$, *and* $\tau Y = A$.

(Decidability) *It is decidable, given valid sgd-context* $\Psi, \mathcal{C}$ *and ambiguous term* Q, *whether or not there exist schematic terms* X *and* Y, *and constraint set* $\mathcal{D}$ *such that* $\Psi, \mathcal{C} \vdash Q \Rightarrow X, Y, \mathcal{D}$ *and* $\mathcal{C} \cup \mathcal{D}$ *is consistent.*

Proof

(Soundness) *The proof is by induction on the height of a derivation* δ *of* $\Psi, \mathcal{C} \vdash Q \Rightarrow X, Y, \mathcal{D}$. *We consider here only the rule* A-AD-DEF. *Suppose that* δ *is a derivation of*

$$\Psi_x[x{=}X{:}Y, \mathcal{G}]\Psi^x, \mathcal{C} \vdash x \Rightarrow \nu_{\mathrm{LV}(\mathcal{G}) \setminus \mathrm{LV}(\mathcal{C})}(X, Y, \mathcal{G})$$

Writing $\Psi = \Psi_x[x{=}X{:}Y, \mathcal{G}]\Psi^x$ *and* $(X^*, Y^*, \mathcal{G}^*) = \nu_{\mathrm{LV}(\mathcal{G}) \setminus \mathrm{LV}(\mathcal{C})}(X, Y, \mathcal{G})$, *we have* $\sigma \models \mathcal{C} \cup \mathcal{G}^*$, *thus by the validity of* $(\Psi, \mathcal{C})$, $\widehat{\sigma\Psi_x} \vdash \hat{X} \Rightarrow \sigma X, \sigma Y$. *But* $\widehat{\sigma\Psi} = \widehat{\sigma\Psi_x}[x{=}\hat{X}]\widehat{\sigma\Psi^x}$, *and so, by* O-AD-DEF, $\widehat{\sigma\Psi} \vdash x \Rightarrow \sigma X, \sigma Y$, *as desired.*

(Completeness) *The proof is by induction on the height of a derivation* δ *of* $\widehat{\sigma\Psi} \vdash Q \Rightarrow M, A$, *under the conditions stated in the theorem. The most interesting case is when* δ *is of the form*

$$\dfrac{\begin{array}{c}\vdots \\ \widehat{\sigma\Psi_x} \vdash \hat{X} \Rightarrow M, A\end{array}}{\widehat{\sigma\Psi} \vdash x \Rightarrow M, A}$$

where $\Psi = \Psi_x[x{=}X{:}Y, \mathcal{G}]\Psi^x$. *Therefore by* A-AD-DEF, $\Psi \vdash x \Rightarrow X^*, Y^*, \mathcal{G}^*$ *where* $(X^*, Y^*, \mathcal{G}^*) = \nu_{\mathrm{LV}(\mathcal{G}) \setminus \mathrm{LV}(\mathcal{C})}(X, Y, \mathcal{G})$. *By principality*

of $(\Psi, \mathcal{C})$*, there exists* τ *extending* σ *such that* $\tau \models \mathcal{G}^*$, $\tau X^* = M$, *and* $\tau Y^* = A$, *as desired.*

(Decidability) *Similar to the proof of Theorem 4.5.* □

7 Related Work

Huet, in an unpublished manuscript [26], has independently developed an algorithm for handling universes in the Calculus of Constructions. His approach is to drop the assumption that the universes form a linearly-ordered cumulative hierarchy indexed by the natural numbers, and to consider instead a family of calculi in which there is some well-founded partial ordering of universes. The input language is correspondingly restricted so that *specific* universes are disallowed; only the anonymous universe Type may be used. The principal advantage of this approach over the one considered here is that the consistency checking algorithm is significantly more efficient than Chan's algorithm, reducing to an acyclicity check in a dependency graph of universe levels. Efficiency considerations aside, this approach is equivalent to ours for the restricted language that Huet considers because any countable well-founded partial ordering can be embedded in a countable linear ordering. However, our method has the advantage that we can, for example, easily restrict the type checker to terms that check within, say, one universe, or any fixed bound. This can be of use for calibrating the strength of the proof-theory needed to formalize an argument. Moreover, "local" constraints can be imposed in a proof simply by using a specific, rather than anonymous, universe.

In response to our observation that cumulativity entails flexibility in the type of a term that is not determined by the shape of type alone (see the discussion following Theorem 3.3), Luo [31] developed an alternative formulation of a cumulative hierarchy of universes (called "fully cumulative") that eliminates the need for schematic type expressions in the basic type checking algorithm for CC^ω. The idea is to introduce a partial ordering on type expressions with the property that a type of the form $\{x_1{:}A_1\}\cdots\{x_n{:}A_n\}\mathsf{Type}_i$ is less than, in this ordering, to a type of the form $\{x_1{:}A_1\}\cdots\{x_n{:}A_n\}\mathsf{Type}_j$ whenever $i \leq j$. (In addition, Prop is taken to be less than Type_0, but this does not effect the type checking algorithm.) The types of a term are

then required to form an upward-closed set in this ordering. In this way the need for schematic terms and constraint sets in the basic type synthesis algorithm is replaced by a more complex application rule. Of course, every derivation in our system is a valid derivation in this extended sense, but the converse fails. Nevertheless, the resulting system is consistent and decidable, as demonstrated in [31]. It should be stressed that our methods for handling definitions and anonymous universes extend directly to Luo's calculus. However, the implementation of definitions and anonymous universes in Luo's calculus can be significantly more efficient than for CC^ω since constraints are generated only in connection with typical ambiguity and universe polymorphism, and not as part of the basic type checking algorithm.

We know of two machine implementations of CC^ω. Gérard Huet and coworkers are developing an implementation of CC^ω [13] that supports Huet's variant of typical ambiguity discussed above. The second author has implemented the algorithms of this paper in the Lego proof checker [32]. Lego supports several type theories, including CC^ω and Luo's variation on it, extended with typical ambiguity and universe polymorphism.

References

[1] Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, revised edition, 1984.

[2] Rod Burstall and Butler Lampson. A kernel language for abstract data types and modules. In Kahn et al. [28], pages 1–50.

[3] Luca Cardelli. Phase distinctions in type theory. unpublished manuscript.

[4] Luca Cardelli. A polymorphic λ-calculus with Type:Type. Technical report, DEC SRC, 1986.

[5] Tat-Hung Chan. An algorithm for checking PL/CV arithmetical inferences. Technical Report 77–236, Computer Science Department, Cornell University, Ithaca, New York, 1977.

[6] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[7] Dominique Clément, Joëlle Despeyroux, Thierry Despeyroux, Laurent Hascoet, and Gilles Kahn. Natural semantics on the computer. Technical Report RR 416, INRIA, Sophia–Antipolis, France, June 1985.

[8] Robert L. Constable and Daniel R. Zlatin. Report on the type theory (V3) of the programming logic PL/CV3. In *Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*. Springer-Verlag, 1982.

[9] Robert L. Constable and Daniel R. Zlatin. The type theory of PL/CV3. *ACM Transactions on Programming Languages and Systems*, 7(1):72–93, January 1984.

[10] Robert L. Constable, *et. al. Implementing Mathematics with the NuPRL Proof Development System*. Prentice-Hall, 1986.

[11] Thierry Coquand. *Une Théorie des Constructions*. PhD thesis, Université Paris VII, January 1985.

[12] Thierry Coquand. An analysis of Girard's paradox. In *Symposium on Logic in Computer Science*, pages 227–236, Boston, June 1986.

[13] Thierry Coquand, Gilles Dowek, Gérard Huet, and Christine Paulin-Mohring. The Calculus of Constructions: Documentation and user's guide. Technical report, Projet Formel, INRIA-ENS, July 1989.

[14] Thierry Coquand and Gérard Huet. Constructions: A higher-order proof system for mechanizing mathematics. In B. Buchberger, editor, *EUROCAL '85: European Conference on Computer Algebra*, volume 203 of *Lecture Notes in Computer Science*, pages 151–184. Springer-Verlag, 1985.

[15] Luis Damas and Robin Milner. Principal type schemes for functional programs. In *Ninth ACM Symposium on Principles of Programming Languages*, pages 207–212, 1982.

[16] Nicolas G. de Bruijn. A survey of the project AUTOMATH. In Seldin and Hindley [45], pages 589–606.

[17] Thomas Erhard. A categorical semantics of Constructions. In *Third Symposium on Logic in Computer Science*, pages 264–273, Edinburgh, July 1988.

[18] Paola Giannini and Simona Ronchi della Rocca. Characterization of typings in polymorphic type discipline. In *Third Symposium on Logic in Computer Science*, pages 61–71, July 1988.

[19] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. In *Second Symposium on Logic in Computer Science*, pages 194–204, Ithaca, New York, June 1987.

[20] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. Technical Report CMU–CS–89–173, School of Computer Science, Carnegie Mellon University, January 1989. Revised and expanded version of [19], submitted for publication.

[21] Robert Harper, Robin Milner, and Mads Tofte. A type discipline for program modules. In *TAPSOFT '87*, volume 250 of *Lecture Notes in Computer Science*. Springer-Verlag, March 1987.

[22] Robert Harper, John C. Mitchell, and Eugenio Moggi. Higher-order modules and the phase distinction. (To appear POPL '90), 1989.

[23] James G. Hook and Douglas Howe. Impredicative strong existential equivalent to Type:Type. Technical Report TR 86–760, Cornell University, Ithaca, New York, 1986.

[24] William A. Howard. The formulas-as-types notion of construction. In Seldin and Hindley [45], pages 479–490.

[25] Douglas Howe. The computational behavior of Girard's paradox. In *Second Symposium on Logic in Computer Science*, pages 205–214, Ithaca, New York, June 1987.

[26] Gérard Huet. Extending the Calculus of Constructions with Type:Type. unpublished manuscript, April 1987.

[27] J. Martin E. Hyland and Andrew M. Pitts. The Theory of Constructions: Categorical semantics and topos-theoretic models. In *Proceedings of*

the Boulder Conference on Categories in Computer Science, 1988. To appear.

[28] Gilles Kahn, David MacQueen, and Gordon Plotkin, editors. *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*. Springer-Verlag, June 1984.

[29] Zhaolui Luo. A higher-order calculus and theory abstraction. Technical Report ECS–LFCS–88–57, Laboratory for the Foundations of Computer Science, Edinburgh University, July 1988.

[30] Zhaolui Luo. $\mathrm{CC}^{\infty}_{\subset}$ and its metatheory. Technical Report ECS–LFCS–88-58, Laboratory for the Foundations of Computer Science, Edinburgh University, July 1988.

[31] Zhaolui Luo. Ecc, an extended calculus of constructions. In *Fourth Symposium on Logic in Computer Science*, Asilomar, California, June 1989.

[32] Zhaolui Luo, Robert Pollack, and Paul Taylor. How to use lego: A preliminary user's manual. Technical Report LFCS-TN-27, Laboratory for the Foundations of Computer Science, Edinburgh University, October 1989.

[33] David MacQueen. Using dependent types to express modular structure. In *Thirteenth ACM Symposium on Principles of Programming Languages*, 1986.

[34] Per Martin-Löf. A theory of types. Unpublished manuscript.

[35] Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975.

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

[37] Per Martin-Löf. *Intuitionistic Type Theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, Naples, 1984.

[38] Albert Meyer and Mark Reinhold. 'Type' is not a type: Preliminary report. In *Thirteenth ACM Symposium on Principles of Programming Languages*, 1986.

[39] Robin Milner. A theory of type polymorphism in programming languages. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[40] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML*. MIT Press, 1990.

[41] John Mitchell and Robert Harper. The essence of ML. In *Fifteenth ACM Symposium on Principles of Programming Languages*, San Diego, California, January 1988.

[42] John C. Mitchell. Type inference and type containment. In Kahn et al. [28], pages 257–278.

[43] John C. Mitchell and Gordon Plotkin. Abstract types have existential type. In *Twelfth ACM Symposium on Principles of Programming Languages*, 1985.

[44] Bertrand Russell. Mathematical logic as based on a theory of types. *American Journal of Mathematics*, 30:222–262, 1908.

[45] J. P. Seldin and J. R. Hindley, editors. *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*. Academic Press, 1980.

[46] Diedrik T. van Daalen. *The Language Theory of AUTOMATH*. PhD thesis, Technical University of Eindhoven, Eindhoven, Netherlands, 1980.

[47] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica, Volume 1*. Cambridge University Press, 1925.