Type theory in type theory with single substitutions

Anonymous Author(s)

Abstract

Type theory can be described as a generalised algebraic theory. This automatically gives a notion of model and the existence of the syntax as the initial model, which is a quotient inductive-inductive type. Algebraic definitions of type theory include Ehrhard's definition of model, categories with families (CwFs), contextual categories, Awodey's natural models, C-systems, B-systems. With the exception of B-systems, these notions are based on a parallel substitution calculus where substitutions form a category. In this paper we define a single substitution calculus (SSC) for type theory and show that the SSC syntax and the CwF syntax are isomorphic for a theory with dependent function space and a hierarchy of universes. SSC only includes single substitutions and single weakenings, and eight equations relating these: four equations describe how to substitute variables, and there are four equations on types which are needed to typecheck other equations. All the other equations are substitution (naturality) rules or computation rules for different type formers. SSC provides a simple, minimalistic alternative to parallel substitution calculi or B-systems for defining type theory. SSC relates to CwF as extensional combinatory calculus relates to lambda calculus. If we have some additional type formers, we show that SSC models are actually equivalent to CwF-based models in a weak sense. Most results in this paper were formalised in Agda.

1 Introduction

What is type theory? Here we refer to type theory as a particular formal system based on Martin-Löf's original definition [44], and not to the study of type systems (e.g. [47]).

Type theory is a language which can be described as a second-order generalised algebraic theory (SOGAT [37]). Some properties of this description:

(i) It is an intrinsic presentation [9, 13] as opposed to extrinsic [1, 56]. That is, we only consider well-formed, well-scoped and well-typed abstract syntax trees, there are no meaningless terms.

(ii) Terms are quotiented by the conversion relation.

(iii) Every operation is stable under substitution.

For example, type theory with Π types and universes à la Coquand [25] is specified by the following SOGAT.

$$\begin{aligned}
&\mathsf{Ty} &&: \mathbb{N} \to \mathsf{Set}\\
&\mathsf{Tm} &&: \mathsf{Ty}\, i \to \mathsf{Set}\\
&\Pi &&: (A : \mathsf{Ty}\, i) \to (\mathsf{Tm}\, A \to \mathsf{Ty}\, i) \to \mathsf{Ty}\, i\\
&\mathsf{app} &&: \mathsf{Tm}\,(\Pi\, A\, B) \cong ((a : \mathsf{Tm}\, A) \to \mathsf{Tm}\,(B\, a)) : \mathsf{lam}\\
&\mathsf{U} &&: (i : \mathbb{N}) \to \mathsf{Ty}\,(1 + i)\\
&\mathsf{El} &&: \mathsf{Tm}\,(\mathsf{U}\, i) \cong \mathsf{Ty}\, i : \mathsf{c}\\
&\mathsf{Lift} &&: \mathsf{Ty}\, i \to \mathsf{Ty}\,(1 + i)\\
&\mathsf{un} &&: \mathsf{Tm}\,(\mathsf{Lift}\, A) \cong \mathsf{Tm}\, A : \mathsf{mk}
\end{aligned} \tag{1}$$

Because types are isomorphic to certain terms (witnessed by El and c), we have to index types by their (universe) level, and types at lower levels are included at upper levels by Lift. We use implicit arguments, e.g. Tm takes the $i : \mathbb{N}$ implicitly, and app takes i, A and B implicitly. $f : X \cong Y : g$ denotes an isomorphism, that is, $f : X \to Y$, $g : Y \to X$ with $g\,(f\,x) = x$ and $f\,(g\,y) = y$ for all x, y. Note that Π and lam are second-order functions. We say that Π binds a Tm-variable in its second Ty-argument. Every binder is represented by a second-order function, and as a result, we don't need to talk about contexts and substitutions when describing the theory: we re-use the binding features of the metatheory to describe those of our object theory (this technique is also known as higher-order abstract syntax [33], logical framework [31, 46] or two-level type theory [8, 14]). There is no good notion of homomorphism between second-order models, thus the semantics of a SOGAT is given by a first order generalised algebraic theory (GAT [21]). GATs are generalisations of single-sorted algebraic theories (such as monoids, groups) to multiple sorts where later sorts can be indexed over previous sorts. The usual results from universal algebra (e.g. models form a category with finite limits and initial object, existence of free and co-free models) carry over to GATs [41, 45]. An example of a GAT is the theory of categories where there is a sort of objects and a sort of morphisms double-indexed over objects.

Kaposi and Xie [37] present two different first order semantics for any SOGAT: one based on parallel substitutions, and one based on single substitutions.

In the parallel semantics, the GAT always starts with a category with a terminal object, objects of the category are called contexts, morphisms are substitutions. Every sort/operation of the SOGAT gives rise to a sort/operation in the GAT which is now indexed by contexts. The sorts moreover come with instantiation (sometimes called substitution) operations which

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

are functorial, and all the operations are natural with respect to instantiation. For those sorts which are bound by an operation in the SOGAT description, we have context extension, meaning that contexts can carry variables of that sort (in other words, the sort is given by a locally representable presheaf). The parallel semantics of the SOGAT

$$\begin{aligned} \mathsf{Ty} &: \mathsf{Set} \\ \mathsf{Tm} &: \mathsf{Ty} \to \mathsf{Set} \end{aligned} \tag{2}$$

is the GAT of categories with families (CwFs [22, 28], Definition 3.1). We define the notion "model of the SOGAT" by its parallel substitution semantics, which is a GAT, and that has an obvious notion of model. The syntax of the language is the initial model, which exists for every GAT [35]. This is an intrinsic syntax (only well-typed terms), quotiented by the equations in the SOGAT, which describe conversion for the language.

The single substitution semantics is a GAT which does not start with a category: there are contexts and maps between contexts, but these maps are single substitutions or single weakenings, they don't compose. There are fewer operations and fewer equations than in the parallel semantics. In this paper we investigate the single substitution semantics of the type theory specified by the SOGAT (1). This presentation of type theory is simpler than CwFs, the operations are all easy to motivate, we illustrate this in Section 2. Single substitution calculi are closer to the usual presentation of type theory using derivation rules, this was one of the motivations of Voevodsky for developing B-systems [4]. Our single substitution semantics of (2) is simpler and has more models than B-systems.

The single substitution semantics is in some sense too minimalistic: for example, the equation $b[\mathsf{p}^+][\langle \mathsf{q} \rangle] = b$ is not *derivable* in any model, but it is *admissible* in the syntax. (Here $b : \mathsf{Tm}\,(\Gamma \triangleright A)\,B$ meaning that it is a term which depends on a context Γ and an extra variable of type A; $b[\mathsf{p}^+] : \mathsf{Tm}\,(\Gamma \triangleright A \triangleright A[\mathsf{p}])\,(B[\mathsf{p}^+])$ is a version where we weakened *before* the last variable; in $b[\mathsf{p}^+][\langle \mathsf{q} \rangle] : \mathsf{Tm}\,(\Gamma \triangleright A)\,B$ we substituted the last variable for the previous one.) This is analogous to e.g. parametricity results [16], which do not hold in an arbitrary model, but they hold in the syntax. In this paper, we show that for the particular SOGAT (1), the *syntaxes* of the single and parallel semantics are isomorphic. The techniques that we use for this proof are not specific to the SOGAT (1), we expect that they work for any SOGAT. Isomorphism of the syntaxes means that the corresponding sorts (Con, Ty, Tm) are isomorphic. For a general SOGAT, there are more models of the correponding single substitution GAT than the parallel substitution GAT. Every parallel model is also a single model, but not necessarily the other way. While B-systems are equivalent to CwFs (and C-systems [4]), there are more single substitution models of (2) than parallel substitution models (CwFs). The situation is similar to that of extensional combinatory calculus and lambda calculus, where the syntaxes are equivalent, and every lambda model is a combinatory model, but not the other way round [12]. A simpler example is the relationship of monoids over a set X and nil/cons algebras over X, where every monoid is a nil/cons-algebra, but not the other way; it is well-known that free monoids over X (syntax of monoid over X) are the same as lists of X (the syntax of nil/cons algebras over X).

When showing that the parallel and single substitution syntaxes are equivalent, we define the parallel operations by induction on the single substitution syntax. One of the operations in the single substitution syntax is the instantiation operation: $-[-] : \mathsf{Ty}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta$, and similarly for terms. When doing induction on types/terms, we have to provide a method for these operations, but this is difficult as substitutions don't compose. This is why we derive a new induction principle which works on α-normal types and terms. Just as β-normal forms don't distinguish between β-equal terms, α-normal forms don't include explicit substitutions. α-normal forms are not usual normal forms in the sense that they are still quotiented by the computation/uniqueness rules like β/η for function space. To derive the new induction principle for the syntax, we first prove α-normalisation.

In the particular theory (1), every type is represented by a term. In its single substitution calculus, we use this to adapt some equations and remove some others, obtaining a more minimalistic, but isomorphic GAT.

Theory (1) can be extended with unit and Σ types, more precisely with the following rules.

$$\begin{aligned} \top &: \mathsf{Ty}\,0 \\ \mathsf{tt} &: \mathsf{Tm}\,\top \\ \top\eta &: t = \mathsf{tt} \\ \Sigma &: (A : \mathsf{Ty}\,i) \to (\mathsf{Tm}\,A \to \mathsf{Ty}\,i) \to \mathsf{Ty}\,i \\ \mathsf{fst}, \mathsf{snd} &: \mathsf{Tm}\,(\Sigma\,A\,B) \cong (a : \mathsf{Tm}\,A) \times \mathsf{Tm}\,(B\,a) : -, - \end{aligned} \tag{3}$$

Using the same insight about types and the universe, the fact that dependences can be represented by function spaces, and that collections of dependencies can be collected using Σ types, we can actually derive a parallel model from any single substitution model. This way we show that for the particular theory (1)+(3), single and parallel substitution models are actually equivalent in a weak sense.

To summarise, our contributions are the following:

- A new generalised algebraic presentation of type theory in the form of a minimalistic single substitution calculus. We show that there is no need for parallel substitutions, empty substitution, parallel weakenings, telescopes, or combinations of these to describe the syntax of type theory in an intrinsic way. We present our calculus in an easy-to-understand way.
- The α-normalisation technique and deriving parallel substitutions from the single substitution syntax.

- For type theory with Π, U, a minimised presentation of the equations which results in an isomorphic theory.
- For type theory with Π, U, ⊤, Σ, we show that single and parallel models are equivalent in a weak sense.

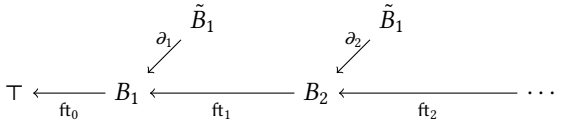
1.1 Structure of the paper

In Section 2, we introduce the single substitution calculus (SSC) only relying on a working knowledge of an implementation of type theory such as Agda, no prior experience with the metatheory of type theory is required. Section 3 shows that in the syntax of SSC, several new equations are admissible, and that the syntax of SSC is actually isomorphic to the CwF-based syntax. In Section 4 we minimise our SSC obtaining a theory with fewer equations, relying on the presence of Π and U in our theory. In Section 5, we show that in the presence of Π, U and Σ, SSC-models are actually equivalent to CwF-models in a weak sense. We conclude in Section 6.

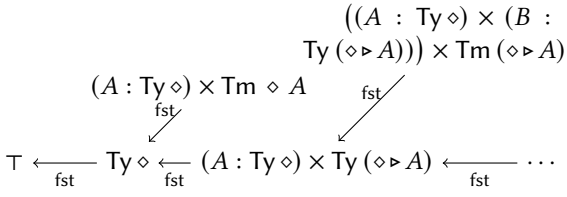
1.2 Related work

In the 1990s single substitution calculi for type theory were popular, but usually in the extrinsic setting, see e.g. the thesis of Altenkirch [6].

B-systems were introduced by Voevodsky [55] as an algebraic way of describing type theories close to the notations using typing judgements. B-systems are an intrinsic, essentially algebraic presentation of type theory using single substitutions. B-systems correspond to our single substitution calculus as essentially algebraic theories correpond to generalised algebraic theories, or sets with a map into I correspond to indexed families over I [21, page 221]. Another difference is that we have fewer and less general equations, resulting in the fact that we have more models than B-systems. However in the syntax of our theory, all the rules of B-systems are admissible. We describe the relationship in more detail. B-systems are B-frames together with substitution, weakening and generic element operations. A B-frame is given by sets B_i, $\tilde{B}_i$ and functions between them as shown below [4].



Using our notation, a B-frame contains the following data.



The substitution operation $\mathbb{S}$ for an element of $x : \tilde{B}_{n+1}$ is a homomorphism of the slice B-frames $\mathbb{B}/\partial(x) \to \mathbb{B}/\mathrm{ft}(\partial(x))$. In our notation, $x = (\Gamma, A, a)$ where $a : \mathsf{Tm}\,\Gamma\,A$, and the homomorphism $\mathbb{S}$ corresponds to the following maps for Ty (mapping between the bottom rows of the diagrams):

$$
\begin{aligned}
&-[\langle a\rangle] &&: \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma\\
&-[\langle a\rangle^+] &&: \mathsf{Ty}\,(\Gamma \triangleright A \triangleright B) \to \mathsf{Ty}\,(\Gamma \triangleright B[\langle a\rangle])\\
&-[\langle a\rangle^{++}] &&: \mathsf{Ty}\,(\Gamma \triangleright A \triangleright B \triangleright C) \to \mathsf{Ty}\,(\Gamma \triangleright B[\langle a\rangle] \triangleright C[\langle a\rangle^+])\\
&\ldots,
\end{aligned}
$$

and similarly $\mathbb{S}$ also includes all the (lifted) substitution operations for terms (top rows of the diagrams). Analogously, the weakening operation corresponds to $-[\mathsf{p}^{+\cdots+}]$ operations, and the generic element is q in our notation. The six groups of equations correspond to our $[\langle\rangle][]$, $[\mathsf{p}][^+]$, $\mathsf{q}[^+]$, $[\mathsf{p}][\langle\rangle]$, $\mathsf{q}[\langle\rangle]$, $[\mathsf{p}^+][\langle\mathsf{q}\rangle]$ equations, in this order. However we don't have the lifted versions of our equations, equations $[\langle\rangle][]$, $[\mathsf{p}^+][\langle\mathsf{q}\rangle]$ are only stated for types, and $\mathsf{q}[^+]$, $\mathsf{q}[\langle\rangle]$ are only stated for terms. The missing equations are admissible, see Section 3. In the presence of universes and Π types, we reduce the needed equations even more, see Section 4.

Kaposi and Luksa [39] defined a telescopic single substitution calculus for simple type theory, it can be seen as a simply typed version of B-systems. They show that the category of contextual models of their calculus is equivalent to the category of contextual simply typed CwFs with function space. Their equivalence result holds even without the presence of function space. In the presence of function space, the same equivalence holds for the simply typed version of our substitution calculus, see the formalisation accompanying this paper.

The first generalised algebraic presentation of type theory was Thomas Ehrhard's calculus [26, 29] which featured a parallel substitution calculus with p, $-^+$ and $\langle - \rangle$ operations, just like in our calculus. Our single substitution calculus is essentially Ehrhard's calculus with the categorical composition and identity operations removed. Categories with families (CwFs [22, 28]) feature p, q, $-$, $-$ operations which make it more apparent that substitutions are lists of terms. CwFs are equivalent to Ehrhard's calculus, and so are contextual categories/C-systems [5, 21], the natural models of Awodey [15] and B-systems [4].

Most computer formalisations of type theory are extrinsic [2, 3, 51]: they define the syntax as abstract syntax trees, and equip it with typing and conversion relations. A higher level representation is working in setoid hell [23]: terms are intrinsically well-typed, but conversion is still an explicit relation. For formalising the GAT-level syntax, one needs a stronger metatheory than ordinary Agda or Coq: the metatheory has to support quotient inductive-inductive types (QIITs [35]), in other words, initial models of GATs. Altenkirch and Kaposi [9] formalised the syntax of type theory using postulated QIITs in Agda, together with parametricity and normalisation [10] proofs. Brunerie and de Boer [20] constructed the initial contextual category in Agda using postulated quotients. QIITs are supported by the cubical set model [27] and the setoid model [38], thus Cubical Agda [54] and setoid/observational

type theory [7, 49] support QIITs. For example, in Cubical Agda, set-truncated and groupoid-truncated syntaxes of type theory have been formalised [11]. Without QIITs, tricks such as shallow embedding the syntax can be used to formalise metatheoretic results about type theory such as gluing [34], and its special cases canonicity and parametricity [36].

At the level of abstraction of SOGATs [37, 53], the difference between single and parallel substitution calculi are not visible. However, the metatheory of type theory can be still investigated in this abstract setting, via the methods of Synthethic Tait Computability [52] or internal sconing [18].

1.3 Metatheory and formalisation

Our metatheory is observational type theory [49] with quotient inductive-inductive types (QIITs). On paper we usually omit writing coercions, so our notation is close to extensional type theory. Sections 2 and 3 of this paper are formalised in the proof assistant Agda, we use a strict Prop-valued [30] equality type with postulated coercion rule (transport rule) and postulated QIITs with computation rules added using rewrite rules [24]. Thus, we work in a setting of uniqueness of identity proofs (UIP), and our metatheory is incompatible with homotopy type theory [48]. Our notation on paper is close to Agda's, we write Π types as $(x : A) \to B$, Σ types as $(x : A) \times B$, and use implicit arguments and overloading extensively, for example $-[-]$ for types and terms overloaded.

2 Single substitution syntax

In this section, we introduce the syntax of type theory with function space and universes in a minimalistic way, only introducing operations that are unavoidable. We eschew boilerplate by only defining well-formed, well-scoped, well-typed terms that are quotiented by conversion. This means that the equalities that hold between terms are the ways we can convert terms into each other when running them as programs. We give a tutorial-style introduction to the syntax, we do not assume prior knowledge of the metatheory of type theory. The definition of type theory that we obtain is the same as the single substitution GAT semantics [37] of the SOGAT (1).

First of all, we need a sort of terms which have to be indexed by types because we only want well-typed terms. Both types and terms can include variables, and to keep track of the currently available variables, we also index them by contexts: a context is a list of types, the length is the number of available variables, and we add new variables at the end (that is, they are snoc-lists rather than cons-lists). Types are also indexed by their universe level, this is a technical requirement for avoiding Russell's paradox. The index i is an implicit argument of Tm.

$$\begin{aligned}\mathsf{Con} &: \mathsf{Set}\\ \mathsf{Ty} &: \mathsf{Con} \to \mathbb{N} \to \mathsf{Set}\\ \mathsf{Tm} &: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Set}\end{aligned}$$

Just as lists have two constructors, there are two ways to form contexts: the empty context $\diamond$ and context extension $\triangleright$ which is like the snoc operation for lists. Context extension takes a type which can have variables in the context preceeding the type.

$$\begin{aligned}\diamond &: \mathsf{Con}\\ - \triangleright - &: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Con}\end{aligned}$$

We will refer to variables by their distance from the end of the context. We would like to describe the operation providing the last variable in a context, the zero De Bruijn index [19], we denote it by $\mathsf{q} : \mathsf{Tm}\,(\Gamma \triangleright A)\,A'$. It is a term in an extended context, and it should have type A, but the issue is that $A : \mathsf{Ty}\,\Gamma$, and $A' : \mathsf{Ty}\,(\Gamma \triangleright A)$. We need a way to weaken the type A to obtain such an A'. For this, we will introduce an operation $-[\mathsf{p}] : \mathsf{Ty}\,\Gamma\,i \to \mathsf{Ty}\,(\Gamma \triangleright A)\,i$ and define $A' := A[\mathsf{p}]$. Instead of introducing just $-[\mathsf{p}]$, we generalise a bit and add a new sort Sub which for now only has the single element p and the operation $-[-]$ is called *instantiation*. We instantiate the type with the $\mathsf{Sub}\,\Delta\,\Gamma$ weakening. Elements of Sub will be later called substitutions, hence the name, but for now, we only have single end-of context weakenings in Sub. Δ is called the domain, Γ the codomain of a $\mathsf{Sub}\,\Delta\,\Gamma$.

$$\begin{aligned}\mathsf{Sub} &: \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}\\ \mathsf{p} &: \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma\\ -[-] &: \mathsf{Ty}\,\Gamma\,i \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta\,i\\ \mathsf{q} &: \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}])\end{aligned}$$

The operations p and q take three implicit parameters, Γ, i and A, while $-[-]$ takes Γ, i and Δ implicitly. Still, we only have the last variable q in the context, we don't have e.g. the last but one variable in $\mathsf{Tm}\,(\Gamma \triangleright A \triangleright B)\,(A[\mathsf{p}][\mathsf{p}])$ where we had to weaken A twice to make it fit with its context. To obtain more variables, we also allow weakening of terms, more precisely, we introduce an instantiation operation for terms with an overloaded notation.

$$-[-] : \mathsf{Tm}\,\Gamma\,A \to (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma])$$

Note that this is a dependent function as the type of the resulting term has to be weakened the same way as the term itself. Now we can define all variables counting from the end of the context (De Bruijn indices): $0 := \mathsf{q}$, $1 := \mathsf{q}[\mathsf{p}]$, $2 := \mathsf{q}[\mathsf{p}][\mathsf{p}]$, $3 := \mathsf{q}[\mathsf{p}][\mathsf{p}][\mathsf{p}]$, and so on.

Next we add dependent function space Π: the domain of a dependent function is a type in some context Γ, and the codomain can also refer to a variable in the domain, so we extend the context of the codomain type with the type of

the domain. For simplicity, both types are at the same level (this can be remedied using Lift, see later).

$$\Pi : (A : \mathsf{Ty}\,\Gamma\,i) \to \mathsf{Ty}\,(\Gamma \triangleright A)\,i \to \mathsf{Ty}\,\Gamma\,i$$

The nondependent function space $- \Rightarrow - : \mathsf{Ty}\,\Gamma\,i \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Ty}\,\Gamma\,i$ is a special case of Π and we define it as the abbreviation $A \Rightarrow B := \Pi\,A\,(B[\mathsf{p}])$.

Because we introduced weakening, we now have to explain what happens to a Π type once we weaken it. Assume $A : \mathsf{Ty}\,\Gamma\,i$, $B : \mathsf{Ty}\,(\Gamma \triangleright A)\,i$ and we have $\mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright C)\,\Gamma$. Then we say $(\Pi\,A\,B)[\mathsf{p}] = \Pi\,(A[\mathsf{p}])\,(B[\mathsf{p}'])$, but B cannot be weakened by p because there we need $\mathsf{p}' : \mathsf{Sub}\,(\Gamma \triangleright C \triangleright A[\mathsf{p}])\,(\Gamma \triangleright A)$. So p' has to be a new kind of weakening which adds a new variable in the last but one position. But after introducing p', we would need another equation computing $(\Pi\,A\,B)[\mathsf{p}']$, introducing a new weakening which adds a new variable in the last but two position, and so on. We solve all of these issues by allowing the *lifting* of a weakening: γ^+ will be the same as the weakening $\gamma : \mathsf{Sub}\,\Delta\,\Gamma$, except that it will add an extra variable both at the end of the domain and codomain context.

$$-^+ : (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Sub}\,(\Delta \triangleright A[\gamma])\,(\Gamma \triangleright A)$$

Note that A is an implicit argument of $-^+$, and that it has to be instantiated by the weakening γ in the domain context in order to fit in. Now we can explain how instantiation acts on Π by the following equation.

$$\Pi[] : (\Pi\,A\,B)[\gamma] = \Pi\,(A[\gamma])\,(B[\gamma^+])$$

The above equation has 6 implicit arguments, namely Γ, i, A, B, Δ and γ. This rule works for any weakening, no matter how many $-^+$s have been applied to it. As of now, all elements of Sub have the form

$$\mathsf{p}^{+^n} : \mathsf{Sub}\,(\Gamma \triangleright A \triangleright B_1[\mathsf{p}] \triangleright B_2[\mathsf{p}^+] \triangleright \ldots \triangleright B_n[\mathsf{p}^{+^{n-1}}])\,(\Gamma \triangleright B_1 \triangleright B_2 \triangleright \ldots \triangleright B_n).$$

where $+^n$ means the n-times iteration of $-^+$.

Now that we have weakenings of the form γ^+, we have to say how they act on variables, that is, terms of the form $\mathsf{q}[\mathsf{p}]\ldots[\mathsf{p}]$. We express this using two rules: we say what $\mathsf{q}[\gamma^+]$ computes to, and what $b[\mathsf{p}][\gamma^+]$ computes to where b is an arbitrary term. $\mathsf{q}[\gamma^+]$ should be the same as q (with different implicit arguments as the q in $\mathsf{q}[\gamma^+]$), as the weakening happens somewhere in the middle of the context, so the index of the variable remains unchanged. However, assuming $\mathsf{q} : \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}])$, we have $\mathsf{q}[\gamma^+] : \mathsf{Tm}\,(\Delta \triangleright A[\gamma])\,(A[\mathsf{p}][\gamma^+])$, but in the same context, we have $\mathsf{q} : \mathsf{Tm}\,(\Delta \triangleright A[\gamma])\,(A[\gamma][\mathsf{p}])$, hence the terms in the two sides of the equation $\mathsf{q}[\gamma^+] = \mathsf{q}$ have different types. But these two types should be the same: weakening a type at the end of the context, and then applying another lifted weakening should be the same as first weakening somewhere and then weakening at the end. We first assume this equation for types, then the equation $\mathsf{q}[\gamma^+] = \mathsf{q}$ becomes well-typed (in the metatheory). The equation for $b[\mathsf{p}][\gamma^+]$ has the same shape as the newly assumed rule for types. Thus we add the following three equations.

$$[\mathsf{p}][^+] : B[\mathsf{p}][\gamma^+] = B[\gamma][\mathsf{p}]$$
$$[\mathsf{p}][^+] : b[\mathsf{p}][\gamma^+] = b[\gamma][\mathsf{p}]$$
$$\mathsf{q}[^+] \quad : \mathsf{q}[\gamma^+] = \mathsf{q}$$

The second and third equations only make sense because of the first equation. The phenomenon that later equations (or operations) depend on previous equations is usual in the world of GATs. In fact, the later equations cannot even be stated without having some previous equations.

Now that we have $[\mathsf{p}][^+]$ for types, we can derive the instantiation rule for nondependent function space as follows.

$$\begin{aligned}
&(A \Rightarrow B)[\gamma] && = \\
&(\Pi\,A\,(B[\mathsf{p}]))[\gamma] && =(\Pi[]) \\
&\Pi\,(A[\gamma])\,(B[\mathsf{p}][\gamma^+]) && =([\mathsf{p}][^+]) \\
&\Pi\,(A[\gamma])\,(B[\gamma][\mathsf{p}]) && = \\
&(A[\gamma]) \Rightarrow (B[\gamma])
\end{aligned}$$

So far our only terms are variables, but we would like to define functions via lambda abstraction. Abstraction takes a term in a context extended by the domain of the function. It comes with a rule for instantiation analogous to $\Pi[]$.

$$\mathsf{lam} \quad : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)$$
$$\mathsf{lam}[] : (\mathsf{lam}\,b)[\gamma] = \mathsf{lam}\,(b[\gamma^+])$$

Note that the equation $\mathsf{lam}[]$ only makes sense because of the previous equation $\Pi[]$: the left hand side is in $\mathsf{Tm}\,\Delta\,(\Pi\,A\,B[\gamma])$, in the right hand side, $b[\gamma^+] : \mathsf{Tm}\,(\Delta \triangleright A[\gamma])\,(B[\gamma^+])$, hence the right hand side is in $\mathsf{Tm}\,\Delta\,(\Pi\,(A[\gamma])\,(B[\gamma^+]))$.

The functions $\lambda x.x$ and $\lambda x\,y.x$ can be defined in our syntax as $\mathsf{lam}\,\mathsf{q} : \mathsf{Tm}\,\Gamma\,(A \Rightarrow A)$ and $\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}])) : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,(B \Rightarrow (A[\mathsf{p}])))$ which make sense for all Γ, A, B.

The application operation for dependent function space is a bit tricky, because the return type depends on the input: the argument of the function will appear in the return type. We write application by an infix $- \cdot - : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to (a : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,B'$, where $B' : \mathsf{Ty}\,\Gamma$ should be $B : \mathsf{Ty}\,(\Gamma \triangleright A)$ where the last variable is *substituted* (instantiated) by a. For this, we introduce a new element of Sub called single substitution which goes the opposite way of p. Now we can use instantiation $-[-]$ to substitute the last variable in B.

$$\langle - \rangle \quad : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright A)$$
$$- \cdot - : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to (a : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,(B[\langle a \rangle])$$

We could have introduced new, separate sorts and $-[-]$ operations for weakenings and substitutions, but we merge them for simplicity. There is no need for separation: just as weakenings can be lifted, single substitutions can also be lifted, and the weakening-rules $\Pi[]$, $\mathsf{lam}[]$ also work for single substitutions. We won't have more ways to introduce

elements of Sub, and there are no equations on Sub. An element of Sub is either a single weakening p lifted a couple of times, or a single substitution lifted a couple of times. We call elements of Sub substitutions for simplicity.

With the introduction of $-\cdot-$, we need a new substitution rule, but again it only makes sense with an extra equation on types saying that first substituting the last variable and then an arbitrary substitution is the same as first the lifted version of the arbitrary substitution that does not touch the last variable, and then substituting the last variable.

$$\begin{aligned}[\langle\rangle][] &: B[\langle a\rangle][\gamma] = B[\gamma^+][\langle a[\gamma]\rangle]\\ \cdot[] &: (t\cdot a)[\gamma] = (t[\gamma])\cdot(a[\gamma])\end{aligned}$$

Following the introduction of the operation $\langle-\rangle$, we need to explain how it acts on variables. Given a variable in the middle of the context (a term $b[\mathsf{p}]$), substituting its last variable simply returns b. Substituting the last variable q reads out the term from $\langle-\rangle$. As usual, the rules only typecheck if we have an equation for types.

$$\begin{aligned}[\mathsf{p}][\langle\rangle] &: B[\mathsf{p}][\langle a\rangle] = B\\ [\mathsf{p}][\langle\rangle] &: b[\mathsf{p}][\langle a\rangle] = b\\ \mathsf{q}[\langle\rangle] &: \mathsf{q}[\langle a\rangle] = a\end{aligned}$$

We don't have a separate sort of variables, so the rule $[\mathsf{p}][\langle\rangle]$ holds not only for variables, but for arbitrary terms. This is not an issue, as weakening a term at the end of its context, and then substituting the newly introduced variable is the same as not doing anything.

We add the computation and uniqueness rule for function space, where the uniqueness rule again needs an extra equation on the codomain type of the function to make sense. This is the typal version of the equation mentioned in the introduction (we don't have its term version).

$$\begin{aligned}\Pi\beta &: \mathsf{lam}\, b\cdot a = b[\langle a\rangle]\\ [\mathsf{p}^+][\langle\mathsf{q}\rangle] &: B[\mathsf{p}^+][\langle\mathsf{q}\rangle] = B\\ \Pi\eta &: t = \mathsf{lam}\,(t[\mathsf{p}]\cdot\mathsf{q})\end{aligned}$$

We add the rules for universes. A universe is a type containing codes for types, this is witnessed by El (elements) and c (code), which make an isomorphism between terms of type $\mathsf{U}\,i$ and types of level i. All three operations come with substitution rules (El[] and c[] are interderivable, so it would be enough to assume one of the two, but we add both to completely align with the algorithm [37] generating the rules from the SOGAT (1)).

$$\begin{aligned}\mathsf{U} &: (i:\mathbb{N}) \to \mathsf{Ty}\,\Gamma\,(1+i) & \mathsf{U}[] &: (\mathsf{U}\,i)[\gamma] = \mathsf{U}\,i\\ \mathsf{El} &: \mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i) \to \mathsf{Ty}\,\Gamma\,i & \mathsf{El}[] &: (\mathsf{El}\,\hat{A})[\gamma] = \mathsf{El}\,(\hat{A}[\gamma])\\ \mathsf{c} &: \mathsf{Ty}\,\Gamma\,i \to \mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i) & \mathsf{c}[] &: (\mathsf{c}\,A)[\gamma] = \mathsf{c}\,(A[\gamma])\\ \mathsf{U}\beta &: \mathsf{El}\,(\mathsf{c}\,A) = A\\ \mathsf{U}\eta &: \mathsf{c}\,(\mathsf{El}\,\hat{A}) = \hat{A}\end{aligned}$$

Finally, we add the rules for moving types one level up. Because we defined function space only between types at the same level, without lifting, we cannot even define the polymorphic identity function.

$$\begin{aligned}\mathsf{Lift} &: \mathsf{Ty}\,\Gamma\,i \to \mathsf{Ty}\,\Gamma\,(1+i) & (\mathsf{Lift}\,A)[\gamma] &= \mathsf{Lift}\,(A[\gamma])\\ \mathsf{mk} &: \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,(\mathsf{Lift}\,A) & (\mathsf{mk}\,a)[\gamma] &= \mathsf{mk}\,(a[\gamma])\\ \mathsf{un} &: \mathsf{Tm}\,\Gamma\,(\mathsf{Lift}\,A) \to \mathsf{Tm}\,\Gamma\,A & (\mathsf{un}\,a)[\gamma] &= \mathsf{un}\,(a[\gamma])\\ \mathsf{Lift}\beta &: \mathsf{un}\,(\mathsf{mk}\,a) = a\\ \mathsf{Lift}\eta &: \mathsf{mk}\,(\mathsf{un}\,a) = a\end{aligned}$$

Definition 2.1 (Single substitution calculus with Π, U, Lift)**.** This concludes the first-order single substitution semantics of the SOGAT (1). For reference, we list the same rules again in Appendix A.

The theory can be extended with new type and term formers in the same fashion: each operation has to be indexed over arbitrary contexts and come equipped with a substitution rule. There is no need to add more structural (substitution calculus) rules. For example, we show how to extend Definition 2.1 with the SOGAT-rules (3).

Definition 2.2 (Single substitution calculus with Π, U, Lift, ⊤, and Σ types)**.** We extend Definition 2.1 with the following rules.

$$\begin{aligned}\top &: \mathsf{Ty}\,\Gamma\,0 & \top[] &: \top[\gamma] = \top\\ \mathsf{tt} &: \mathsf{Tm}\,\Gamma\,\top & \mathsf{tt}[] &: \mathsf{tt}[\gamma] = \mathsf{tt}\\ \top\eta &: t = \mathsf{tt}\\ \Sigma &: (A:\mathsf{Ty}\,\Gamma\,i) \to \mathsf{Ty}\,(\Gamma\triangleright A)\,i \to \mathsf{Ty}\,\Gamma\,i\\ \Sigma[] &: (\Sigma\,A\,B)[\gamma] = \Sigma\,(A[\gamma])\,(B[\gamma^+])\\ \mathsf{fst},\mathsf{snd} &: \mathsf{Tm}\,\Gamma\,(\Sigma\,A\,B) \cong\\ &\quad (a:\mathsf{Tm}\,\Gamma\,A)\times\mathsf{Tm}\,\Gamma\,(B[\langle a\rangle]) : (-,-)\\ ,[] &: (a,b)[\gamma] = (a[\gamma], b[\gamma])\end{aligned}$$

To save space, we compressed the introduction, elimination, β and η rules for Σ into one isomorphism, and did not list substitution laws for fst and snd as they are derivable.

2.1 Examples

The polymorphic identity function for types at the level 0 is defined as

$$\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q}) : \mathsf{Tm}\,\diamond\,\Big(\Pi\,(\mathsf{U}\,0)\,\big(\mathsf{Lift}\,(\mathsf{El}\,\mathsf{q}) \Rightarrow \mathsf{Lift}\,(\mathsf{El}\,\mathsf{q})\big)\Big).$$

Note that readability of this term essentially relies on implicit arguments. For example, q takes $(\diamond\triangleright\mathsf{U}\,0)$ as its first, 1 as its second and $\mathsf{Lift}\,(\mathsf{El}\,\mathsf{q})$ as its third implicit argument, and these arguments themselves are terms written using implicit arguments. Inrinsically typed terms are the same as derivation trees, we illustrate this by deriving this term as

follows.

$$\dfrac{\dfrac{\overline{\mathsf{q} : \mathsf{Tm}\,(\diamond \triangleright \mathsf{U}\,0 \triangleright \mathsf{Lift}\,(\mathsf{El}\,\mathsf{q}))\,(\mathsf{Lift}\,(\mathsf{El}\,\mathsf{q})[\mathsf{p}])}}{\mathsf{lam}\,\mathsf{q} : \mathsf{Tm}\,(\diamond \triangleright \mathsf{U}\,0)\,(\mathsf{Lift}\,(\mathsf{El}\,\mathsf{q}) \Rightarrow \mathsf{Lift}\,(\mathsf{El}\,\mathsf{q}))}}{\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q}) : \mathsf{Tm}\,\diamond\,\big(\Pi\,(\mathsf{U}\,0)\,(\mathsf{Lift}\,(\mathsf{El}\,\mathsf{q}) \Rightarrow \mathsf{Lift}\,(\mathsf{El}\,\mathsf{q}))\big)}\ (*)$$

Note that in step (*), lam made it sure that $\mathsf{U}\,0$ and $\mathsf{Lift}\,(\mathsf{El}\,\mathsf{q}) \Rightarrow \mathsf{Lift}\,(\mathsf{El}\,\mathsf{q})$ are types at the same level, this is why we had to lift $\mathsf{El}\,\mathsf{q}$. Another subtle thing is happening when deriving the third implicit argument of q. We have to coerce q along the equality $\mathsf{U}[]$.

$$\dfrac{\dfrac{\dfrac{\overline{\mathsf{q} : \mathsf{Tm}\,(\diamond \triangleright \mathsf{U}\,0)\,((\mathsf{U}\,0)[\mathsf{p}])} \qquad \overline{\mathsf{U}[] : (\mathsf{U}\,0)[\mathsf{p}] = \mathsf{U}\,0}}{\mathsf{q} : \mathsf{Tm}\,(\diamond \triangleright \mathsf{U}\,0)\,(\mathsf{U}\,0)}}{\mathsf{El}\,\mathsf{q} : \mathsf{Ty}\,(\diamond \triangleright \mathsf{U}\,0)\,0}}{\mathsf{Lift}\,(\mathsf{El}\,\mathsf{q}) : \mathsf{Ty}\,(\diamond \triangleright \mathsf{U}\,0)\,1}$$

When working informally (or in extensional type theory), we don't write the coercion, but Agda requires it.

Given a type $A : \mathsf{Ty}\ \diamond\ 0$, and $a : \mathsf{Tm}\ \diamond\ A$, we have

$$\begin{aligned}
&\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q}) \cdot \mathsf{c}\,A \cdot a &&= (\Pi\beta)\\
&(\mathsf{lam}\,\mathsf{q}[\langle \mathsf{c}\,A\rangle]) \cdot a &&= (\mathsf{lam}[])\\
&(\mathsf{lam}\,(\mathsf{q}[\langle \mathsf{c}\,A\rangle^+])) \cdot a &&= ([\mathsf{q}][^+])\\
&(\mathsf{lam}\,\mathsf{q}) \cdot a &&= (\Pi\beta)\\
&\mathsf{q}[\langle a\rangle] &&= (\mathsf{q}[\langle\rangle])\\
&a
\end{aligned}$$

If we want to specify open inputs $A' : \mathsf{Ty}\,(\diamond \triangleright C)\,0$, and $a' : \mathsf{Tm}\,(\diamond \triangleright C)\,A'$, we have to weaken our function to accept them:

$$\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q})[\mathsf{p}] : \mathsf{Tm}\,(\diamond \triangleright C)\ \Big(\Pi\,(\mathsf{U}\,0)\,\big(\mathsf{Lift}\,(\mathsf{El}\,\mathsf{q}) \Rightarrow \mathsf{Lift}\,(\mathsf{El}\,\mathsf{q})\big)[\mathsf{p}]\Big)$$

But we have

$$\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q})[\mathsf{p}] \overset{\mathsf{lam}[]\ 2\mathsf{x}}{=} \mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}^{++}])) \overset{\mathsf{q}[^+]}{=} \mathsf{lam}\,(\mathsf{lam}\,\mathsf{q})$$

(with different implicit arguments), so we are able apply $\mathsf{c}\,A'$ and a' just as before. What we cannot do is to directly weaken a term in context $\diamond$ to an arbitrary metavariable context Γ, we can only do this step by step by applying weakening $-[\mathsf{p}]$ for each type in the context.

2.2 Standard model

As a sanity check for our notion of SSC-model, we defined the standard (metacircular [9]) model of the SSC-based calculus calculus in Agda. We used the techniques of Kovács [40] to define inductive-recursive universes to model the infinite hierarchy of types, and all the SSC-equations hold by reflexivity in this model. We defined the analogous CwF-based standard model.

3 Admissible equations

In this section we show that our SSC-based syntax is isomorphic to the CwF-based syntax for the same theory (namely the SOGAT (1)). We conjecture that the method presented here works generically for any SOGAT.

Definition 3.1 (CwF)**.** A category with $\mathbb{N}$-many families is defined as a category (objects denoted Con, morphisms Sub) with a terminal object $\diamond$ (ϵ denotes the unique morphism into it); for each i a presheaf of types (action on objects denoted $\mathsf{Ty} - i$, action on morphisms $-[-]$); for each i a locally representable dependent presheaf Tm over $\mathsf{Ty} - i$ (actions denoted Tm, $-[-]$, local representability is denoted $- \triangleright - : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Con}$ with an isomorphism $(\mathsf{p} \circ -, \mathsf{q}[-]) : \mathsf{Sub}\,\Delta\,(\Gamma \triangleright A) \cong (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \times \mathsf{Tm}\,\Delta\,(A[\gamma])$: $(-,-)$ natural in Δ).

The following four equations correspond to equations in B-systems which cannot be derived from the equations of SSC, these are versions of equations in Section 2 lifted over arbitrary amount of $-^+$s.

$$\begin{aligned}
B[\mathsf{p}^{+^n}][(\gamma^+)^{+^n}] &= B[\gamma^{+^n}][\mathsf{p}^{+^n}]\\
B[\mathsf{p}^{+^n}][\langle a\rangle^{+^n}] &= B\\
B[\langle a\rangle^{+^n}][\gamma^{+^n}] &= B[(\gamma^+)^{+^n}][\langle a[\gamma]\rangle^{+^n}]\\
B[(\mathsf{p}^+)^{+^n}][\langle \mathsf{q}\rangle^{+^n}] &= B
\end{aligned} \tag{4}$$

The corresponding equations on terms are also needed. Note that we do not have the non-lifted term versions of the last two equations in SSC.

To formally state the equations in (4), we define telescopes as an inductive type together with a recursive operation to append telescopes to contexts.

$$\begin{aligned}
&\mathsf{Tel} &&: \mathsf{Con} \to \mathsf{Set}\\
&- + - &&: (\Gamma : \mathsf{Con}) \to \mathsf{Tel}\,\Gamma \to \mathsf{Con}\\
&\diamond &&: \mathsf{Tel}\,\Gamma\\
&- \triangleright - &&: (\Omega : \mathsf{Tel}\,\Gamma) \to \mathsf{Ty}\,(\Gamma + \Omega) \to \mathsf{Tel}\,\Omega
\end{aligned}$$

The appending operation is defined recursively on telescopes.

$$\begin{aligned}
\Gamma + \diamond &:= \Gamma\\
\Gamma + (\Omega \triangleright A) &:= (\Gamma + \Omega) \triangleright A
\end{aligned}$$

We then define a lifting operation over any telescope mutually with an instantiation operation on telescopes by recursion on telescopes.

$$\begin{aligned}
&-[-] &&: \mathsf{Tel}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tel}\,\Delta\\
&-^{+^\Omega} &&: (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Sub}\,(\Delta + \Omega[\gamma])\,(\Gamma + \Omega)
\end{aligned}$$

If we try to prove the equations in (4) by induction on the types and terms, we run into difficulties in the instantiation case, as we would have to commute the instantiations without the equations to do so. Therefore, we first α-normalize the syntax to compute away all the instantiations, then prove the equations by induction on the α-normal forms.

A type or term is α-normal if it does not contain instantiation operations, except at the leaves of the syntax tree as variables. However, α-normal terms can still contain β/η redexes, so that we do not need to do full normalization. We define variables and α-normal forms as inductive predicates in Prop as follows.

- $\mathsf{q} : \mathsf{Tm}\,(\Gamma \triangleright A)\ (A[\mathsf{p}])$ is a variable.
- $x[\mathsf{p}] : \mathsf{Tm}\,(\Gamma \triangleright A)\ (B[\mathsf{p}])$ is a variable if $x : \mathsf{Tm}\,\Gamma\,B$ is a variable.
- $x : \mathsf{Tm}\,\Gamma\,A$ is α-normal if x is a variable.
- $\Pi\,A\,B : \mathsf{Ty}\,\Gamma\,i$ is α-normal if A and B are α-normal types.
- $\mathsf{app}\,f\,a : \mathsf{Tm}\,\Gamma\,(B[\langle a\rangle])$ is α-normal if A and B are α-normal types, $f : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)$ and $a : \mathsf{Tm}\,\Gamma\,A$ are α-normal terms.

The predicate is defined similarly for the other type and term formers such as lam. Notably, we do not state that instantiated types/terms are α-normal (except variables). It is important to truncate the α-normal predicate to be propositions to ensure that it preserves the β/η conversion rules.

We prove that the α-normal predicate holds for α-normal types and terms instantiated with α-normal substitutions. However, this needs to be proved separately for weakenings and α-normal single substitutions for the induction to be well-founded. We define predicates for weakening and α-normal single substitutions as follows.

- p is a weakening.
- γ^+ is a weakening if γ is a weakening.
- $\langle a\rangle$ is an α-normal single substitution if a is an α-normal term.
- γ^+ is an α-normal single substitution if γ is an α-normal single substitution.

We define α-normal substitutions to be the disjoint union of weakenings and α-normal single substitutions.

Lemma 3.2. *The α-normal predicate holds for any type and term.*

Proof. By induction on the syntax, α-normalizing the substitutions at the same time. Note that induction on the syntax refers to the elimination principle of the corresponding QIIT. □

Instead of doing induction on α-normal forms for each of the equations in (4), we define a general lemma which can lift any equation between instantiations over a telescope. For this we define Sub^* to be Sub with freely added identity and composition operations, we do not require it to satisfy the laws of a category as we will not compare Sub^*s directly for equality, only between types and terms instantiated with Sub^*s. All instantiation operations, lifting operations, and substitution rules are redefined for Sub^*.

Lemma 3.3. *Given $\gamma_0, \gamma_1 : \mathsf{Sub}^*\,\Delta\,\Gamma$, if $A[\gamma_0] = A[\gamma_1]$ and $x[\gamma_0] = x[\gamma_1]$ for any $A : \mathsf{Ty}\,\Gamma$ and variable $x : \mathsf{Tm}\,\Gamma\,A$, then*

- $\Omega[\gamma_0] = \Omega[\gamma_1]$ *for* $\Omega : \mathsf{Tel}\,\Gamma$
- $A[\gamma_0^{+^\Omega}] = A[\gamma_1^{+^\Omega}]$ *for* $A : \mathsf{Ty}\,(\Gamma + \Omega)$
- $a[\gamma_0^{+^\Omega}] = a[\gamma_1^{+^\Omega}]$ *for* $a : \mathsf{Tm}\,(\Gamma + \Omega)\,A$

Note that the later equations depend the the earlier ones.

Proof. Assuming the first equation, we prove that $x[\gamma_0^{+^\Omega}] = x[\gamma_1^{+^\Omega}]$ for any variable x by induction on the telescope and the variable. Then the two latter equations can be proven by mutual induction on α-normalized types and terms, still assuming the first equation. Finally we prove the first equation by induction on the telescope, using the previously proven equation for types, discharging its assumption. □

We can simulate parallel substitutions using Sub^* as iterated single substitutions, however it does not satisfy the equations of parallel substitutions. Thus we define parallel substitutions as a list of terms with a map into Sub^*, and reuse the instantiation operation of Sub^* to convert it into a sequence of instantiations of single substitutions.

$$
\begin{aligned}
\mathsf{Tms} &: \mathsf{Con} \to \mathsf{Con} \to \mathsf{Con}\\
\llcorner - \lrcorner &: \mathsf{Tms}\,\Delta\,\Gamma \to \mathsf{Sub}^*\,\Delta\,\Gamma\\
\epsilon &: \mathsf{Tms}\,\Gamma\,\diamond\\
-, - &: (\gamma : \mathsf{Tms}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\llcorner\gamma\lrcorner]) \to \mathsf{Tms}\,\Delta\,(\Gamma \triangleright A)\\
\llcorner\epsilon\lrcorner &:= \mathsf{id} \circ \mathsf{p} \circ \cdots \circ \mathsf{p}\\
\llcorner\gamma, a\lrcorner &:= \llcorner\gamma\lrcorner^+ \circ \langle a\rangle
\end{aligned}
$$

The computation of instantiating with Tms is illustrated below.

$$B[\llcorner\epsilon, a_1, a_2, \ldots, a_{n-1}, a_n\lrcorner] =$$
$$B[\mathsf{p}^{+^n}] \ldots [\mathsf{p}^{+^n}][\langle a_1\rangle^{+^{n-1}}][\langle a_2\rangle^{+^{n-2}}] \ldots [\langle a_{n-1}\rangle^{+}][\langle a_n\rangle]$$

All CwF operations and equations can be defined with Tms by induction, using Lemma 3.3 to avoid further induction on the syntax.

Theorem 3.4. *Contexts, types, and terms in the SSC syntax are isomorphic to the corresponding sorts in the CwF syntax. In addition* Tms *is isomorphic to CwF substitutions.*

Proof.

- ⇒ By recursion on the syntax, SSC operations can be trivially interpreted by CwF operations.
- ⇐ By recursion on the syntax, using Tms to interpret parallel substitutions.

The roundtrips are proven by induction on the syntax. □

4 Minimisation

In this section we show that if have we universes and Π types (which is the case in the theory (1)), we can decrease the number of equations.

In our single substitution calculus, we have equation $[\mathsf{p}][^+]$ stated both for types and terms (we know that the left one is

about types because of our convention that type metavariables are uppercase Latin letters and term metavariables are lowercase):

$$B[\mathsf{p}][\gamma^+] = B[\gamma][\mathsf{p}] \qquad b[\mathsf{p}][\gamma^+] =_{[\mathsf{p}][^+]} b[\gamma][\mathsf{p}]$$

We even need the first equation to typecheck the second one. We made the dependency explicit by adding a subscript of the equality in the equation for terms (in Agda, this dependency has to be made explicit). An alternative presentation of the second equation without requiring the first one is the following:

$$[\mathsf{p}][^+]' : (e : B[\mathsf{p}][\gamma^+] = B[\gamma][\mathsf{p}]) \to b[\mathsf{p}][\gamma^+] =_e b[\gamma][\mathsf{p}]$$

That is, for any type B for which $B[\mathsf{p}][\gamma^+] = B[\gamma][\mathsf{p}]$, we have that for any b we have the equation (suitably over the input equation). Thus this is a conditional equation.

It turns out that if we have Coquand-universes, the conditional $[\mathsf{p}][^+]'$ rule is enough: the input equation holds for $B := \mathsf{U}\,i$ via $\mathsf{U}[]$, thus we get that for any $\hat{A} : \mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i)$, $\hat{A}[\mathsf{p}][\gamma^+] = \hat{A}[\gamma][\mathsf{p}]$. But every type has a code in the universe, so for a $B : \mathsf{Ty}\,\Gamma\,i$,

$$\begin{aligned}
&B[\mathsf{p}][\gamma^+] && =(\mathsf{U}\beta)\\
&\mathsf{El}\,(\mathsf{c}\,B)[\mathsf{p}][\gamma^+] && =(\mathsf{El}[])\\
&\mathsf{El}\,((\mathsf{c}\,B)[\mathsf{p}][\gamma^+]) && =([\mathsf{p}][^+]'\,\mathsf{U}[])\\
&\mathsf{El}\,((\mathsf{c}\,B)[\gamma][\mathsf{p}]) && =(\mathsf{El}[])\\
&\mathsf{El}\,(\mathsf{c}\,B)[\gamma][\mathsf{p}] && =(\mathsf{U}\beta)\\
&B[\gamma][\mathsf{p}].
\end{aligned}$$

Equation $[\mathsf{q}][^+] : \mathsf{q}[\gamma^+] = \mathsf{q}$ also only makes sense if we have $[\mathsf{p}][^+]$ for types, so either we have to make $[\mathsf{q}][^+]$ conditional, or we coerce it along the derived equation.

We play the exact same game with $[\mathsf{p}][\langle\rangle]$: we replace it with the conditional equation

$$[\mathsf{p}][\langle\rangle]' : (e : B[\mathsf{p}][\langle a\rangle] = B) \to b[\mathsf{p}][\langle a\rangle] =_e b,$$

notice that we have the input for $B = \mathsf{U}\,i$, and derive the input equation for all types. We make $\mathsf{q}[\langle\rangle]$ conditional as well.

Now we turn our attention to the equation

$$[\langle\rangle][] : B[\langle a\rangle][\gamma] = B[\gamma^+][\langle a[\gamma]\rangle].$$

We were forced to introduce it so that we can state the substitution rule for function application $\cdot[]$. What if we made $\cdot[]$ itself conditional?

$$\begin{aligned}
\cdot\,[]' : (e : B[\langle a\rangle][\gamma] = B[\gamma^+][\langle a[\gamma]\rangle]) \to\\
(t \cdot a)[\gamma] =_e (t[\gamma]) \cdot (a[\gamma])
\end{aligned}$$

Again, for $B = \mathsf{U}$ we have the assumption e. But then t is in $\mathsf{Tm}\,\Gamma\,(\Pi\,A\,\mathsf{U})$, which is isomorphic to $\mathsf{Ty}\,(\Gamma \triangleright A)$. So we argue as follows for any $B : \mathsf{Ty}\,(\Gamma \triangleright A)$.

$$\begin{aligned}
&B[\langle a\rangle][\gamma] && =(\mathsf{U}\beta)\\
&\mathsf{El}\,(\mathsf{c}\,(B[\langle a\rangle][\gamma])) && =(\mathsf{c}[])\\
&\mathsf{El}\,((\mathsf{c}\,B)[\langle a\rangle][\gamma]) && =(\Pi\beta)\\
&\mathsf{El}\,((\mathsf{lam}\,(\mathsf{c}\,B)\cdot a)[\gamma]) && =(\cdot[]'\,\mathsf{U}[])\\
&\mathsf{El}\,((\mathsf{lam}\,(\mathsf{c}\,B)[\gamma])\cdot(a[\gamma])) && =(\mathsf{lam}[])\\
&\mathsf{El}\,(\mathsf{lam}\,(\mathsf{c}\,B[\gamma^+])\cdot(a[\gamma])) && =(\Pi\beta)\\
&\mathsf{El}\,(\mathsf{c}\,B[\gamma^+][\langle a[\gamma]\rangle]) && =(\mathsf{c}[])\\
&\mathsf{El}\,(\mathsf{c}\,B)[\gamma^+][\langle a[\gamma]\rangle] && =(\mathsf{U}\beta)\\
&B[\gamma^+][\langle a[\gamma]\rangle]
\end{aligned}$$

Hence, the conditional $\cdot[]'$ equation implies the condition for all types.

Finally, we will remove equation $[\mathsf{p}^+][\langle\mathsf{q}\rangle]$ by making $\Pi\eta$ conditional. This needs another change: we substitute $\Pi\beta$ for a more general variant which is also conditional on the same equation.

$$\Pi\eta' : (e : B[\mathsf{p}^+][\langle\mathsf{q}\rangle] = B) \to t =_e \mathsf{lam}\,(t[\mathsf{p}]\cdot\mathsf{q})$$
$$\Pi\beta' : (e : B[\mathsf{p}^+][\langle\mathsf{q}\rangle] = B) \to (\mathsf{lam}\,b)[\mathsf{p}]\cdot\mathsf{q} =_e b$$

But first we need to check that the usage of $\Pi\beta$ above when proving $B[\langle a\rangle][\gamma] = B[\gamma^+][\langle a[\gamma]\rangle]$ is derivable from $\Pi\beta'$. Assuming a $\hat{B} : \mathsf{Tm}\,(\Gamma \triangleright A)\,\mathsf{U}$,

$$\begin{aligned}
\Pi\beta^{\mathsf{U}} : \mathsf{lam}\,\hat{B}\cdot a && =(\mathsf{q}[\langle\rangle])\\
\mathsf{lam}\,\hat{B}\cdot(\mathsf{q}[\langle a\rangle]) && =([\mathsf{p}][\langle\rangle])\\
((\mathsf{lam}\,\hat{B})[\mathsf{p}][\langle a\rangle])\cdot(\mathsf{q}[\langle a\rangle]) && =(\cdot[]'\,\mathsf{U}[])\\
((\mathsf{lam}\,\hat{B})[\mathsf{p}]\cdot\mathsf{q})[\langle a\rangle] && =(\Pi\beta'\,\mathsf{U}[])\\
\hat{B}[\langle a\rangle].
\end{aligned}$$

Now we derive the assumption of $\Pi\beta/\eta'$ for any type $B : \mathsf{Ty}\,(\Gamma \triangleright A)$.

$$\begin{aligned}
&B[\mathsf{p}^+][\langle\mathsf{q}\rangle] && =(\mathsf{U}\beta)\\
&\mathsf{El}\,(\mathsf{c}\,B)[\mathsf{p}^+][\langle\mathsf{q}\rangle] && =(\mathsf{El}[])\\
&\mathsf{El}\,(\mathsf{c}\,B[\mathsf{p}^+][\langle\mathsf{q}\rangle]) && =(\Pi\beta^{\mathsf{U}})\\
&\mathsf{El}\,(\mathsf{lam}\,(\mathsf{c}\,B[\mathsf{p}^+])\cdot\mathsf{q}) && =(\mathsf{lam}[])\\
&\mathsf{El}\,(\mathsf{lam}\,(\mathsf{c}\,B)[\mathsf{p}]\cdot\mathsf{q}) && =(\Pi\beta'\,\mathsf{U}[])\\
&\mathsf{El}\,(\mathsf{c}\,B) && =(\mathsf{U}\beta)\\
&B
\end{aligned}$$

We summarise this section by formally stating what we defined.

Definition 4.1 (Minimised single substitution calculus with Π and U)**.** This section defined the minimised first-order version of the SOGAT (1), relying essentially on the presence of universes and Π types. For reference, we list all the rules of the minimised calculus in Appendix B.

Theorem 4.2. *The GATs of Definition 4.1 and Definition 2.1 are isomorphic, in particular, all equations are interderivable.*

Proof. Clear from the construction in this section. □

5 CwF from SSC with Σ, Π and U

In this section we show that if the single substitution calculus has certain type formers, then it is actually weakly equivalent to CwFs with the same type formers. The idea is that

- using Σ types we emulate contexts,
- using functions between these Σ types we emulate parallel substitutions,
- using the functions into the universe we emulate dependent types,

and then single substitutions are just there to set up Σ, Π and U, and these types are enough to bootstrap the parallel substitution calculus.

Problem 5.1. *Every CwF (with type formers* $\Pi, \dots, \Sigma$*) is an SSC (with the same type formers, see Definition 2.2).*

Construction. Most operations are the same, we set $\gamma^+ := (\gamma \circ \mathsf{p}, \mathsf{q})$ and $\langle a \rangle := (\mathsf{id}, a)$. All equations are derivable. □

The following construction is also known as the standard model, metacircular interpretation [9], contextualisation [18]. Following [36] we call it termification, as most sorts in the new model are terms in the old model.

Problem 5.2 (Termification)**.** *From a model of SSC (with* $\Pi, \dots, \Sigma$*, see Definition 2.2), we define a CwF (with the same type formers).*

Construction. We define iterated lifting $\mathsf{Lift}^k : \mathsf{Ty}\,\Gamma\,i \to \mathsf{Ty}\,\Gamma\,(k{+}i)$ by induction on k, together with $\mathsf{un}^k : \mathsf{Tm}\,(\mathsf{Lift}^k\,A) \cong \mathsf{Tm}\,A : \mathsf{mk}^k$. The category part of the CwF is given by types in the empty context and functions between them, suitably lifted (on the left hand side of := there is the component in the new CwF-model, on the right hand side the components refer to the old SSC-model):

$$\begin{aligned}\mathsf{Con} &:= (i : \mathbb{N}) \times \mathsf{Ty}\ \diamond\ i\\ \mathsf{Sub}\,\Delta\,\Gamma &:= \mathsf{Tm}\ \diamond\ (\mathsf{Lift}^{\Gamma-\Delta}\,\Delta \Rightarrow \mathsf{Lift}^{\Delta-\Gamma}\,\Gamma)\end{aligned}$$

In the definition of Sub, we did not write the projections for Con, so Δ can mean $\Delta_{.1}$ or $\Delta_{.2}$, and we used the truncating subtraction of natural numbers. Composition of substitutions is quite involved, but it is just function composition written with explicit weakenings and appropriate (un)liftings. The category laws hold. The empty context is modelled by ⊤, its η law holds via η for ⊤.

$$\gamma \circ \delta := \mathsf{lam}\Big(\mathsf{mk}^{\Theta-\Gamma}\,\Big(\mathsf{un}^{\Delta-\Gamma}\,\Big(\gamma[\mathsf{p}]\cdot\mathsf{mk}^{\Gamma-\Delta}\,(\mathsf{un}^{\Theta-\Delta}\,(\delta[\mathsf{p}]\cdot\mathsf{mk}^{\Delta-\Theta}\,(\mathsf{un}^{\Gamma-\Theta}\,\mathsf{q})))\Big)\Big)\Big)$$

$$\mathsf{id} := \mathsf{lam}\,\mathsf{q} \qquad \diamond := (0, \top) \qquad \epsilon := \mathsf{lam}\,(\mathsf{mk}^{\Gamma}\,\mathsf{tt})$$

Types are given by functions into U, terms are dependent functions into the type, with lots of lifting adjustments.

$$\mathsf{Ty}\,\Gamma\,i := \mathsf{Tm}\ \diamond\ \big(\mathsf{Lift}^{1+i-\Gamma}\,\Gamma \Rightarrow \mathsf{Lift}^{\Gamma-(1+i)}\,(\mathsf{U}\,i)\big)$$

$$\mathsf{Tm}\,\Gamma\,A := \mathsf{Tm}\ \diamond\ \Big(\Pi\,(\mathsf{Lift}^{i-\Gamma}\,\Gamma)\,\Big(\mathsf{Lift}^{\Gamma-i}\,\big(\mathsf{El}\ \big(\mathsf{un}^{\Gamma-(1+i)}\,(A[\mathsf{p}]\cdot\mathsf{mk}^{1+i-\Gamma}\,(\mathsf{un}^{i-\Gamma}\,\mathsf{q}))\big)\big)\Big)\Big)$$

To make the notation readable, from now on, we will not write the lifting decorations or universe from now on. We repeat the previous definitions again without writing decorations, but all of what follows can be defined properly.

$$\begin{aligned}\mathsf{Con} &:= \mathsf{Ty}\,\diamond\\ \mathsf{Sub}\,\Delta\,\Gamma &:= \mathsf{Tm}\ \diamond\ (\Delta \Rightarrow \Gamma)\\ \gamma \circ \delta &:= \mathsf{lam}\,(\gamma[\mathsf{p}]\cdot(\delta[\mathsf{p}]\cdot\mathsf{q}))\\ \mathsf{id} &:= \mathsf{lam}\,\mathsf{q}\\ \diamond &:= \top\\ \epsilon &:= \mathsf{lam}\,\mathsf{tt}\\ \mathsf{Ty}\,\Gamma &:= \mathsf{Tm}\ \diamond\ (\Gamma \Rightarrow \mathsf{U})\\ A[\gamma] &:= \mathsf{lam}\,(A[\mathsf{p}]\cdot(\gamma[\mathsf{p}]\cdot\mathsf{q}))\\ \mathsf{Tm}\,\Gamma\,A &:= \mathsf{Tm}\ \diamond\ (\Pi\,\Gamma\,(\mathsf{El}\,(A[\mathsf{p}]\cdot\mathsf{q})))\\ a[\gamma] &:= \mathsf{lam}\,(a[\mathsf{p}]\cdot(\gamma[\mathsf{p}]\cdot\mathsf{q}))\\ \Gamma \triangleright A &:= \Sigma\,\Gamma\,(\mathsf{El}\,(A[\mathsf{p}]\cdot\mathsf{q}))\\ (\gamma, a) &:= \mathsf{lam}\,(\gamma[\mathsf{p}]\cdot\mathsf{q}, a[\mathsf{p}]\cdot\mathsf{q})\\ \mathsf{p} &:= \mathsf{lam}\,(\mathsf{fst}\,\mathsf{q})\\ \mathsf{q} &:= \mathsf{lam}\,(\mathsf{snd}\,\mathsf{q})\end{aligned}$$

Context extension is given by Σ types and pairing/projections by pairing/projections of Σ. All the CwF equations hold, for example we derive the functor law for type substitution as follows.

$$\begin{aligned}&A[\gamma\circ\delta] && =\\ &\mathsf{lam}\,(A[\mathsf{p}]\cdot(\mathsf{lam}\,(\gamma[\mathsf{p}]\cdot(\delta[\mathsf{p}]\cdot\mathsf{q}))[\mathsf{p}]\cdot\mathsf{q})) && =(\mathsf{lam}[])\\ &\mathsf{lam}\,(A[\mathsf{p}]\cdot(\mathsf{lam}\,(\gamma[\mathsf{p}][\mathsf{p}^+]\cdot(\delta[\mathsf{p}][\mathsf{p}^+]\cdot\mathsf{q}))\cdot\mathsf{q})) && =(\Pi\beta,\cdot[])\\ &\mathsf{lam}\,(A[\mathsf{p}]\cdot(\gamma[\mathsf{p}][\mathsf{p}^+][\langle\mathsf{q}\rangle]\cdot(\delta[\mathsf{p}][\mathsf{p}^+][\langle\mathsf{q}\rangle]\cdot\mathsf{q}))) && =([\mathsf{p}][^+])\\ &\mathsf{lam}\,(A[\mathsf{p}]\cdot(\gamma[\mathsf{p}][\mathsf{p}][\langle\mathsf{q}\rangle]\cdot(\delta[\mathsf{p}][\mathsf{p}][\langle\mathsf{q}\rangle]\cdot\mathsf{q}))) && =([\mathsf{p}][\langle\rangle])\\ &\mathsf{lam}\,(A[\mathsf{p}]\cdot(\gamma[\mathsf{p}]\cdot(\delta[\mathsf{p}]\cdot\mathsf{q}))) && =(\Pi\beta)\\ &\mathsf{lam}\,(\mathsf{lam}\,(A[\mathsf{p}][\mathsf{p}^+]\cdot(\gamma[\mathsf{p}][\mathsf{p}^+]\cdot\mathsf{q}))\cdot(\delta[\mathsf{p}]\cdot\mathsf{q})) && =(\mathsf{lam}[])\\ &\mathsf{lam}\,(\mathsf{lam}\,(A[\mathsf{p}]\cdot(\gamma[\mathsf{p}]\cdot\mathsf{q}))[\mathsf{p}]\cdot(\delta[\mathsf{p}]\cdot\mathsf{q})) && =\\ &A[\gamma][\delta] && \qquad (5)\end{aligned}$$

Type formers are added by adjusting them to handle contexts built up by Σ types, for example,

$$\Pi\,A\,B := \mathsf{lam}\,(\mathsf{c}\,(\Pi\,(\mathsf{El}\,(A[\mathsf{p}]\cdot\mathsf{q}))\,(\mathsf{El}\,(B[\mathsf{p}][\mathsf{p}]\cdot(\mathsf{q}[\mathsf{p}],\mathsf{q})))))$$

and $\mathsf{lam}\,b := \mathsf{lam}\,(\mathsf{lam}\,(b[\mathsf{p}][\mathsf{p}]\cdot(\mathsf{q}[\mathsf{p}],\mathsf{q})))$. All substitution laws and β/η-laws hold. □

The notion of equivalence between CwFs that we use is contextual isomorphism [17]: a weak CwF-morphism (pseudomorphism) which is bijective on types and terms. Note that a conextual isomorphism preserves all type formers specified by universal properties.

Problem 5.3. *Given a CwF (with type formers) M, let M' denote the CwF obtained by first seeing it as an SSC and then termifying it. There is a contextual isomorphism between M' and M.*

Construction. We denote the components of the contextual isomorphism F as follows.

$$F : \mathsf{Con}_{M'} \to \mathsf{Con}_M$$
$$F : \mathsf{Sub}_{M'}\,\Delta\,\Gamma \to \mathsf{Sub}_M\,(F\,\Delta)\,(F\,\Gamma)$$
$$F : \mathsf{Ty}_{M'}\,\Gamma \cong \mathsf{Ty}_M\,(F\,\Gamma)$$
$$F : \mathsf{Tm}_{M'}\,\Gamma\,A \cong \mathsf{Tm}_M\,(F\,\Gamma)\,(F\,A)$$

We define them in the same order as follows, omitting the M subscripts.

$$F : \mathsf{Ty}\,\diamond \to \mathsf{Con}$$
$$F\,\Gamma := \diamond \triangleright \Gamma$$
$$F : \mathsf{Tm}\,\diamond\,(\Delta \Rightarrow \Gamma) \to \mathsf{Sub}\,(\diamond \triangleright \Delta)(\diamond \triangleright \Gamma)$$
$$F\,\gamma := (\mathsf{p}, \gamma[\mathsf{p}] \cdot \mathsf{q})$$
$$F : \mathsf{Tm}\,\diamond\,(\Gamma \Rightarrow \mathsf{U}) \cong \mathsf{Tm}\,(\diamond \triangleright \Gamma)\,\mathsf{U} \cong \mathsf{Ty}\,(\diamond \triangleright \Gamma)$$
$$F\,A := \mathsf{El}\,(A[\mathsf{p}] \cdot \mathsf{q})$$
$$F : \mathsf{Tm}\,\diamond\,(\Pi\,\Gamma\,(\mathsf{El}\,(A[\mathsf{p}] \cdot \mathsf{q}))) \cong \mathsf{Tm}\,(\diamond \triangleright \Gamma)\,(\mathsf{El}\,(A[\mathsf{p}] \cdot \mathsf{q}))$$
$$F\,a := a[\mathsf{p}] \cdot \mathsf{q}$$

It is easy to check that F preserves the CwF structure, i.e. it is functor, $\epsilon : \mathsf{Sub}\,(F\,\diamond)\,\diamond$ is an isomorphism, $F\,(A[\gamma]) = F\,A[F\,\gamma]$, $F\,(a[\gamma]) = F\,a[F\,\gamma]$ and $(F\,\mathsf{p}, F\,\mathsf{q}) : \mathsf{Sub}\,(F\,(\Gamma \triangleright A))\,(F\,\Gamma \triangleright F\,A)$ is an isomorphism. □

The other roundtrip, that is, starting with an SSC-model, termifying it and comparing the result with the original SSC-model does not provide an isomorphism because we can't define the map $F : \mathsf{Tm}\,\diamond\,(\Delta \Rightarrow \Gamma) \to \mathsf{Sub}\,(\diamond \triangleright \Delta)\,(\diamond \triangleright \Gamma)$ on substitutions. Such a map is not definable only using single substitutions. However, the set of terms and types are still isomorphic after the roundtrip. We leave formulating the right notion of contextual isomorphism for SSCs as future work.

6 Conclusions and further work

Why are we still looking at alternative ways of defining type theory? New definitions might be closer to actual implementations, suggest new implementation techniques, they might be better pedagogically, have more models, or contribute to a coherent definition. We elaborate on the latter.

Defining a useable syntax of type theory internally to homotopy type theory (HoTT [48]) is an open problem [42, 50]: the (h)set-truncated syntax can be formalised as a QIIT, but then we cannot even define the Type-interpretation, that is, a function from the syntax which interprets closed types in the metatheoretic universe Type (rather than an inductive recursive universe as in Subsection 2.2 or [9, Section 4]). In two-level type theory the syntax of type theory can be defined using infinitely many coherene rules [43]. For example, for substitution Sub, we have an equation in the syntax on the associativity of composition of substitutions. If we truncate our syntax to be a set, then this is enough. If we don't truncate, there are two different ways of proving $((\gamma \circ \delta) \circ \theta) \circ \xi = \gamma \circ (\delta \circ (\theta \circ \xi))$, and nothing forces these to be the same, so in such a (wild) syntax substitutions don't form a set, hence they don't have decidable equality [32]. The solution is to add the pentagon law, but then this introduces new higher equalities, which require more coherence laws, so in the end we have an infinite sequence of higher coherence laws that need to be part of the syntax. Our single substitution calculus does not have composition for Sub, so there is no need for higher coherence laws like associativity. However as it currently stands, our syntax is not coherent, it needs set-truncation when working in HoTT. For example, there are two different ways to prove the following equation:

$$(\mathsf{c}\,\mathsf{U})[\mathsf{p}][\langle a\rangle] \overset{[\mathsf{p}][\langle\rangle]}{=} \mathsf{c}\,\mathsf{U}$$
$$(\mathsf{c}\,\mathsf{U})[\mathsf{p}][\langle a\rangle] \overset{\mathsf{c}[]}{=} \mathsf{c}\,(\mathsf{U}[\mathsf{p}][\langle a\rangle]) \overset{\mathsf{U}[]}{=} \mathsf{c}\,\mathsf{U}$$

In a coherent version of our syntax we either need to state that these two proofs are equal, or we have to remove one of them. We can remove the first one by limiting $[\mathsf{p}][\langle\rangle]$ to only apply to variables (this needs the introduction of a new sort for variables). If we do this, the minimisation of Section 4 does not work anymore, so we need the extra four equations for types ($[\mathsf{p}][^+]$, $[\mathsf{p}][\langle\rangle]$, $[\langle\rangle][]$, $[\mathsf{p}^+][\langle\mathsf{q}\rangle]$). These equations on types make the syntax non-coherent again. A possible way out of this would be to make the equations that depend on one of these conditional. But then the methods of Section 3 don't work anymore for re-proving the needed admissible equations. Also, if we only have the rule $[\mathsf{p}][^+]$ for variables, we cannot redo the termification of Section 5, e.g. it is not clear how to prove equation (5).

In this paper we unfolded a particular example of the SOGAT → single substitution GAT translation [37]. We showed that for this theory, the syntax is isomorphic to the parallel syntax, and we conjecture that our method works for any SOGAT. For our particular SOGAT, we minimised the single substitution syntax, and we even have that any single substitution model is equivalent to a parallel model in a weak sense. This shows that if we have enough type formers, the substitution calculus can be very weak, and still enough to bootstrap the usual theory. It needs further investigation whether we can remove even more structural rules, and still obtain isomorphic syntaxes. We would also like to understand what a good notion of weak equivalence is for single substitution models.

References

[1] Andreas Abel. Normalization by evaluation: dependent types and impredicativity, 2013. Habilitation thesis, Ludwig-Maximilians-Universität München.

[2] Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Decidability of conversion for type theory in type theory. *Proc. ACM Program. Lang.*, 2(POPL):23:1–23:29, 2018.

[3] Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Pierre-Marie Pédrot, and Loïc Pujet. Martin-Löf à la Coq. In Amin Timany, Dmitriy Traytel, Brigitte Pientka, and Sandrine Blazy, editors, *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024*, pages 230–245. ACM, 2024.

[4] Benedikt Ahrens, Jacopo Emmenegger, Paige Randall North, and Egbert Rijke. B-systems and c-systems are equivalent. *The Journal of Symbolic Logic*, page 1–9, 2023.

[5] Benedikt Ahrens, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. Categorical structures for type theory in univalent foundations. *Log. Methods Comput. Sci.*, 14(3), 2018.

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

[7] Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, and Nicolas Tabareau. Setoid type theory—a syntactic translation. In Graham Hutton, editor, *Mathematics of Program Construction*, pages 155–196, Cham, 2019. Springer International Publishing.

[8] Thorsten Altenkirch, Paolo Capriotti, and Nicolai Kraus. Extending homotopy type theory with strict equality. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[9] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29. ACM, 2016.

[10] Thorsten Altenkirch and Ambrus Kaposi. Normalisation by Evaluation for Type Theory, in Type Theory. *Logical Methods in Computer Science*, Volume 13, Issue 4, October 2017.

[11] Thorsten Altenkirch and Ambrus Kaposi. Coherent categories with families. In Patrick Bahr and Rasmus Ejlers Møgelberg, editors, *30th International Conference on Types for Proofs and Programs (TYPES 2024).* Copenhagen, 2024.

[12] Thorsten Altenkirch, Ambrus Kaposi, Artjoms Sinkarovs, and Tamás Végh. Combinatory logic and lambda calculus are equal, algebraically. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, volume 260 of *LIPIcs*, pages 24:1–24:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[13] Thorsten Altenkirch and Bernhard Reus. Monadic presentations of lambda terms using generalized inductive types. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, volume 1683 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 1999.

[14] Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications. *Math. Struct. Comput. Sci.*, 33(8):688–743, 2023.

[15] Steve Awodey. Natural models of homotopy type theory. *Math. Struct. Comput. Sci.*, 28(2):241–286, 2018.

[16] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Proofs for free - parametricity for dependent types. *J. Funct. Program.*, 22(2):107–152, 2012.

[17] Rafaël Bocquet. External univalence for second-order generalized algebraic theories. *CoRR*, abs/2211.07487, 2022.

[18] Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. For the metatheory of type theory, internal sconing is enough. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, volume 260 of *LIPIcs*, pages 18:1–18:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[19] N. G. De Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *INDAG. MATH*, 34:381–392, 1972.

[20] Guillaume Brunerie and Menno de Boer. Formalization of the initiality conjecture, 2020.

[21] John Cartmell. Generalised algebraic theories and contextual categories. *Ann. Pure Appl. Log.*, 32:209–243, 1986.

[22] Simon Castellan, Pierre Clairambault, and Peter Dybjer. *Categories with Families: Unityped, Simply Typed, and Dependently Typed*, pages 135–180. Springer International Publishing, Cham, 2021.

[23] James Chapman. Type theory should eat itself. *Electronic Notes in Theoretical Computer Science*, 228:21–36, January 2009.

[24] Jesper Cockx, Nicolas Tabareau, and Théo Winterhalter. The taming of the rew: a type theory with computational assumptions. *Proc. ACM Program. Lang.*, 5(POPL):1–29, 2021.

[25] Thierry Coquand. Presheaf model of type theory. https://www.cse.chalmers.se/~coquand/presheaf.pdf, 2018.

[26] Thierry Coquand. Generalised algebraic presentation of type theory. https://www.cse.chalmers.se/~coquand/cwf2.pdf, 2020.

[27] Thierry Coquand, Simon Huber, and Anders Mörtberg. On higher inductive types in cubical type theory. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 255–264. ACM, 2018.

[28] Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*, volume 1158 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 1995.

[29] Thomas Ehrhard. *Une sémantique catégorique des types dépendents.* PhD thesis, Université Paris VII, 1988.

[30] Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional proof-irrelevance without K. *Proc. ACM Program. Lang.*, 3(POPL):3:1–3:28, 2019.

[31] Robert Harper, Furio Honsell, and Gordon D. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, 1993.

[32] Michael Hedberg. A coherence theorem for Martin-Löf's type theory. *Journal of Functional Programming*, 8(04):413–436, 1998.

[33] Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 204–213. IEEE Computer Society, 1999.

[34] Ambrus Kaposi, Simon Huber, and Christian Sattler. Gluing for Type Theory. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, volume 131 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[35] Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL):2:1–2:24, 2019.

[36] Ambrus Kaposi, András Kovács, and Nicolai Kraus. Shallow embedding of type theory is morally correct. In Ugo de' Liguoro and Stefano Berardi, editors, *26th International Conference on Types for Proofs and Programs, TYPES 2020.* University of Turin, 2020.

[37] Ambrus Kaposi and Szumi Xie. Second-order generalised algebraic theories: Signatures and first-order semantics. In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and*

Deduction, FSCD 2024, July 10-13, 2024, Tallinn, Estonia, volume 299 of *LIPIcs*, pages 10:1–10:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[38] Ambrus Kaposi and Zongpu Xie. Quotient inductive-inductive types in the setoid model. In Henning Basold, editor, *27th International Conference on Types for Proofs and Programs, TYPES 2021*. Universiteit Leiden, 2021.

[39] Amrbus Kaposi and Norbert Luksa. A calculus of single substitutions for simple type theory. https://bitbucket.org/akaposi/tel_stt/raw/master/paper.pdf, 2020.

[40] András Kovács. Generalized universe hierarchies and first-class universe levels. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[41] András Kovács. *Type-Theoretic Signatures for Algebraic Theories and Inductive Types*. PhD thesis, Eötvös Loránd University, Hungary, 2022.

[42] Nicolai Kraus. On the role of semisimplicial types. In José Espírito Santo and Luís Pinto, editors, *24th International Conference on Types for Proofs and Programs, TYPES 2018*. University of Minho, 2018.

[43] Nicolai Kraus. Internal ∞-categorical models of dependent type theory : Towards 2ltt eating hott. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021.

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

[45] Hugo Moeneclaey. *Cubical models are cofreely parametric. (Les modèles cubiques sont colibrement paramétriques)*. PhD thesis, Paris Cité University, France, 2022.

[46] Brigitte Pientka and Jana Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In Jürgen Giesl and Reiner Hähnle, editors, *Automated Reasoning*, pages 15–21, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[47] Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

[48] The Univalent Foundations Program. Homotopy type theory: Univalent foundations of mathematics. Technical report, Institute for Advanced Study, 2013.

[49] Loïc Pujet. *Computing with Extensionality Principles in Type Theory. (Calculer avec des Principes d'Extensionnalité en Théorie des Types)*. PhD thesis, University of Nantes, France, 2022.

[50] Mike Shulman. Homotopy type theory should eat itself (but so far, it's too big to swallow), 2014. Blog post on the Homotopy Type Theory website.

[51] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The MetaCoq Project. *J. Autom. Reason.*, 64(5):947–999, 2020.

[52] Jonathan Sterling. *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory*. PhD thesis, Carnegie Mellon University, USA, 2022.

[53] Taichi Uemura. *Abstract and Concrete Type Theories*. PhD thesis, University of Amsterdam, 2021.

[54] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical agda: A dependently typed programming language with univalence and higher inductive types. *J. Funct. Program.*, 31:e8, 2021.

[55] Vladimir Voevodsky. B-systems, 2014.

[56] Théo Winterhalter. *Formalisation and meta-theory of type theory*. PhD thesis, Université de Nantes, 2020.

A GAT presentation of (1)

Definition 2.1 in full detail:
The core substitution calculus:

$$
\begin{array}{ll}
\mathsf{Con} & : \mathsf{Set} \\
\mathsf{Ty} & : \mathsf{Con} \to \mathbb{N} \to \mathsf{Set} \\
\diamond & : \mathsf{Con} \\
- \triangleright - & : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Con} \\
\mathsf{Sub} & : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\
\mathsf{Tm} & : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Set} \\
\mathsf{p} & : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma \\
\langle - \rangle & : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright A) \\
-^{+} & : (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Sub}\,(\Delta \triangleright A[\gamma])\,(\Gamma \triangleright A) \\
-[-] & : \mathsf{Ty}\,\Gamma\,i \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta\,i \\
-[-] & : \mathsf{Tm}\,\Gamma\,A \to (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma]) \\
\mathsf{q} & : \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}]) \\
[\mathsf{p}][^{+}] & : B[\mathsf{p}][\gamma^{+}] = B[\gamma][\mathsf{p}] \\
[\mathsf{p}][^{+}] & : b[\mathsf{p}][\gamma^{+}] = b[\gamma][\mathsf{p}] \\
\mathsf{q}[^{+}] & : \mathsf{q}[\gamma^{+}] = \mathsf{q} \\
[\mathsf{p}][\langle\rangle] & : B[\mathsf{p}][\langle a \rangle] = B \\
[\mathsf{p}][\langle\rangle] & : b[\mathsf{p}][\langle a \rangle] = b \\
\mathsf{q}[\langle\rangle] & : \mathsf{q}[\langle a \rangle] = a \\
[\langle\rangle][] & : B[\langle a \rangle][\gamma] = B[\gamma^{+}][\langle a[\gamma] \rangle] \\
[\mathsf{p}^{+}][\langle\mathsf{q}\rangle] & : B[\mathsf{p}^{+}][\langle\mathsf{q}\rangle] = B
\end{array}
$$

Rules for individual type formers.

$$
\begin{array}{ll}
\Pi & : (A : \mathsf{Ty}\,\Gamma\,i) \to \mathsf{Ty}\,(\Gamma \triangleright A)\,i \to \mathsf{Ty}\,\Gamma\,i \\
\Pi[] & : (\Pi\,A\,B)[\gamma] = \Pi\,(A[\gamma])\,(B[\gamma^{+}]) \\
\mathsf{lam} & : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \\
\mathsf{lam}[] & : (\mathsf{lam}\,b)[\gamma] = \mathsf{lam}\,(b[\gamma^{+}]) \\
- \cdot - & : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to (a : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,(B[\langle a \rangle]) \\
\cdot[] & : (t \cdot a)[\gamma] = (t[\gamma]) \cdot (a[\gamma]) \\
\Pi\beta & : \mathsf{lam}\,b \cdot a = b[\langle a \rangle] \\
\Pi\eta & : t = \mathsf{lam}\,(t[\mathsf{p}] \cdot \mathsf{q}) \\
\mathsf{U} & : (i : \mathbb{N}) \to \mathsf{Ty}\,\Gamma\,(1 + i) \\
\mathsf{U}[] & : (\mathsf{U}\,i)[\gamma] = \mathsf{U}\,i \\
\mathsf{El} & : \mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i) \to \mathsf{Ty}\,\Gamma\,i \\
\mathsf{El}[] & : (\mathsf{El}\,\hat{A})[\gamma] = \mathsf{El}\,(\hat{A}[\gamma]) \\
\mathsf{c} & : \mathsf{Ty}\,\Gamma\,i \to \mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i) \\
\mathsf{c}[] & : (\mathsf{c}\,A)[\gamma] = \mathsf{c}\,(A[\gamma]) \\
\mathsf{U}\beta & : \mathsf{El}\,(\mathsf{c}\,A) = A \\
\mathsf{U}\eta & : \mathsf{c}\,(\mathsf{El}\,\hat{A}) = \hat{A} \\
\mathsf{Lift} & : \mathsf{Ty}\,\Gamma\,i \to \mathsf{Ty}\,\Gamma\,(1 + i) \\
\mathsf{Lift}[] & : (\mathsf{Lift}\,A)[\gamma] = \mathsf{Lift}\,(A[\gamma]) \\
\mathsf{mk} & : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,(\mathsf{Lift}\,A) \\
\mathsf{mk}[] & : (\mathsf{mk}\,a)[\gamma] = \mathsf{mk}\,(a[\gamma]) \\
\mathsf{un}[] & : (\mathsf{un}\,a)[\gamma] = \mathsf{un}\,(a[\gamma]) \\
\mathsf{un} & : \mathsf{Tm}\,\Gamma\,(\mathsf{Lift}\,A) \to \mathsf{Tm}\,\Gamma\,A \\
\mathsf{Lift}\beta & : \mathsf{un}\,(\mathsf{mk}\,a) = a \\
\mathsf{Lift}\eta & : \mathsf{mk}\,(\mathsf{un}\,a) = a
\end{array}
$$

B Minimised GAT presentation of (1)

Definition 4.1 in full detail (omitting Lift):

$$
\begin{array}{ll}
\mathsf{Con} & : \mathsf{Set} \\
\mathsf{Ty} & : \mathsf{Con} \to \mathbb{N} \to \mathsf{Set} \\
\diamond & : \mathsf{Con} \\
- \triangleright - & : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Con} \\
\mathsf{Sub} & : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\
\mathsf{Tm} & : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Set} \\
\mathsf{p} & : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma \\
\langle - \rangle & : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright A) \\
-^{+} & : (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Sub}\,(\Delta \triangleright A[\gamma])\,(\Gamma \triangleright A) \\
-[-] & : \mathsf{Ty}\,\Gamma\,i \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta\,i \\
-[-] & : \mathsf{Tm}\,\Gamma\,A \to (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma]) \\
\mathsf{q} & : \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}]) \\
[\mathsf{p}][^{+}]' & : (e : B[\mathsf{p}][\gamma^{+}] = B[\gamma][\mathsf{p}]) \to b[\mathsf{p}][\gamma^{+}] =_e b[\gamma][\mathsf{p}] \\
\mathsf{q}[^{+}]' & : (e : B[\mathsf{p}][\gamma^{+}] = B[\gamma][\mathsf{p}]) \to \mathsf{q}[\gamma^{+}] =_e \mathsf{q} \\
[\mathsf{p}][\langle\rangle]' & : (e : B[\mathsf{p}][\langle a \rangle] = B) \to b[\mathsf{p}][\langle a \rangle] =_e b \\
\mathsf{q}[\langle\rangle]' & : (e : B[\mathsf{p}][\langle a \rangle] = B) \to \mathsf{q}[\langle a \rangle] =_e a \\
\Pi & : (A : \mathsf{Ty}\,\Gamma\,i) \to \mathsf{Ty}\,(\Gamma \triangleright A)\,i \to \mathsf{Ty}\,\Gamma\,i \\
\Pi[] & : (\Pi\,A\,B)[\gamma] = \Pi\,(A[\gamma])\,(B[\gamma^{+}]) \\
\mathsf{lam} & : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \\
\mathsf{lam}[] & : (\mathsf{lam}\,b)[\gamma] = \mathsf{lam}\,(b[\gamma^{+}]) \\
- \cdot - & : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to (a : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,(B[\langle a \rangle]) \\
\cdot[]' & : (e : B[\langle a \rangle][\gamma] = B[\gamma^{+}][\langle a[\gamma] \rangle]) \to \\
 & \quad (t \cdot a)[\gamma] =_e (t[\gamma]) \cdot (a[\gamma]) \\
\Pi\eta' & : (e : B[\mathsf{p}^{+}][\langle\mathsf{q}\rangle] = B) \to t =_e \mathsf{lam}\,(t[\mathsf{p}] \cdot \mathsf{q}) \\
\Pi\beta' & : (e : B[\mathsf{p}^{+}][\langle\mathsf{q}\rangle] = B) \to (\mathsf{lam}\,b)[\mathsf{p}] \cdot \mathsf{q} =_e b \\
\mathsf{U} & : (i : \mathbb{N}) \to \mathsf{Ty}\,\Gamma\,(1 + i) \\
\mathsf{U}[] & : (\mathsf{U}\,i)[\gamma] = \mathsf{U}\,i \\
\mathsf{El} & : \mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i) \to \mathsf{Ty}\,\Gamma\,i \\
\mathsf{El}[] & : (\mathsf{El}\,\hat{A})[\gamma] = \mathsf{El}\,(\hat{A}[\gamma]) \\
\mathsf{c} & : \mathsf{Ty}\,\Gamma\,i \to \mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i) \\
\mathsf{c}[] & : (\mathsf{c}\,A)[\gamma] = \mathsf{c}\,(A[\gamma]) \\
\mathsf{U}\beta & : \mathsf{El}\,(\mathsf{c}\,A) = A \\
\mathsf{U}\eta & : \mathsf{c}\,(\mathsf{El}\,\hat{A}) = \hat{A}
\end{array}
$$