Type theory with single substitutions*

Ambrus Kaposi
Eötvös Loránd University (ELTE)
Budapest, Hungary
akaposi@inf.elte.hu

Szumi Xie
Eötvös Loránd University (ELTE)
Budapest, Hungary
szumi@inf.elte.hu

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 the other equations. 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: there are more models of the former, but the syntaxes are equivalent. If we have some additional type formers, we show that an SSC model gives rise to a CwF.

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]).

In this paper, we will answer this question by defining type theory as a generalised algebraic theory (GAT [22]). GATs are multi-sorted algebraic theories where later sorts can be indexed by previous sorts. An example is categories where the sort of morphisms is double-indexed over objects. The GAT presentation of type theory is intrinsic [14, 10] rather than extrinsic [7, 1, 3]: this means that we only consider well-formed, well-scoped and well-typed abstract syntax trees, there are no meaningless terms; in the GAT presentation, the conversion relation is expressed by equations of the algebraic theory.

The GAT presentation has the following advantages:

- In type theory, typing depends on conversion: in an extrinsic presentation, this is expressed by the rule deriving $\Gamma \vdash t : B$ from $\Gamma \vdash A \sim B$ and $\Gamma \vdash t : A$. When conversion $\sim$ is equality, this rule holds by the general properties of equality; and we also avoid lots of boilerplate stating that e.g. conversion is a congruence with respect to all operations (for an alternative intrinsic presentation with explicit conversion relation see [24]).
- Syntax and semantics are concordant: the syntax is simply the initial model, which always exists for any GAT and is called a quotient inductive-inductive type [37]. The fact that the syntax is quotiented means that any function defined on it needs to respect the equations. For example, it is not possible to define a printing function which distinguishes the terms $(\lambda x.x)\,(\lambda y.y)$ and $\lambda y.y$, as

*This research was supported by the HOTT ERC Grant 101170308.

Submitted to:
LFMTP 2025

these are convertible. However, it is still possible to define normalisation [11] and typechecking [35] at this level of abstraction.

- The GAT approach is also more abstract than extrinsic approaches, which means that we don't have to make ad-hoc choices. For example, we do not have to decide whether we want function space à la Curry or à la Church; whether we work with paranoid or economic typing rules [56, Section 5]; whether we have explicit or recursively defined substitution (instantiation of variables by terms, written $t[x \mapsto u]$ or $t[u/x]$). These choices are forced by the notion of GAT: the domain of a lambda appears in its type, so we are always à la Church; algebraic theories don't allow recursively defined operators, so instantiation is an explicit operation; and so on.

However, even at the GAT level of abstraction, there are some choices to be made about the instantiation operation. It is usually convenient to use parallel substitutions, which means that the instantiation operation replaces all variables in a context by terms in another context *at the same time*, that is, $t[x_0 \mapsto u_0, \dots, x_n \mapsto u_n]$ where $x_0, \dots, x_n$ are the variables in the context of t. We call $(x_0 \mapsto u_0, \dots, x_n \mapsto u_n)$ a parallel substitution, where the terms $u_0, \dots, u_n$ are all in the same context. Parallel substitutions are very natural in the algebraic setting because they form the morphisms in a category where objects are contexts (lists of types), then types/terms are presheaves, context extension is a representability condition, so the equations of the algebraic theory are forced by the categorical structure. Some examples of algebraic definitions of type theory using parallel substitutions ranging from the more syntactic to the more semantic: Ehrhard's notion [31, 28], categories with families (CwF [30, 23]), natural models [15], contextual categories [22], C-systems [6], locally cartesian closed categories [25], path categories [16].

In traditional extrinsic presentations (e.g. [7, 47]) instantiation replaces free occurrences of a single variable x with a term u as in the notation $t[x \mapsto u]$. A substitution consists of a variable name (a pointer into a context) and a term. Substitutions cannot be composed, thus they do not form a category. To bridge the gap between the intrinsic parallel substitution world and the extrinsic single substitution world, Voevodsky introduced B-systems [4] which are an algebraic description of a single substitution calculus. B-systems involve complex rules describing telescopes, weakenings and substitutions under telescopes, with several equations.

In this paper, we introduce a new single substitution calculus (SSC) which is simpler and more minimalistic than existing algebraic approaches. The instantiation operation $-[-]$ takes either a single substitution or a single weakening as an argument, and there are eight equations explaining how these behave: four of these say how to instantiate variables and the other four equations are needed to typecheck the first four. The SSC shows that to explain type theory in an algebraic way, there is no need for categories, parallel substitutions or weakenings, empty substitution, telescopes, or combinations of these. All our operations are easy to motivate, we showcase this in Section 2 which is a tutorial introduction to our theory. We believe that our SSC is in a sweet spot: our notations are close to traditional extrinsic notations, but they are algebraic, thus come with a well behaved model theory, and can be easily related to more semantic descriptions.

Our SSC 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 $b : \mathsf{Tm}\,(\Gamma \triangleright A)\,B$ 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 [17], which do not hold in an arbitrary model, but they hold in the syntax. In this paper, we show that the *syntaxes* of SSC and the corresponding parallel substitution calculus (CwF-based calculus) are isomorphic. All CwF-based models are models of SSC, but not necessarily the other way. While B-systems are equivalent to contextual CwFs (and

C-systems [4]), there are more SSC models than CwF-based models. This relationship is like 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 [13]. Another example is the relationship between monoids over a set X and nil/cons algebras over X: 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 X-lists (the syntax of nil/cons algebras over X).

We introduce a technology for proving admissible rules. We define α-normal forms and show that every term admits an α-normal form. Thus, proving something for α-normal forms implies it for all terms. Just as β-normal forms don't distinguish between β-convertible terms, α-normal forms don't include explicit instantiations. α-normal forms are not like ordinary β/η-normal forms because they are quotiented by the computation/uniqueness rules like β/η for function space. We use induction on α-normal forms to define parallel substitutions and prove all CwF rules in the syntax of SSC.

Our SSC has 8 equations, and these are enough to describe type theories with arbitrary choice of type formers (see the paragraph on SOGATs in Subsection 1.1). In the particular case of type theory with a Coquand-universe and Π types, we can do better: 4 (conditional) equations are enough, because we can derive the other 4 equations using the fact that every type is represented by a term.

If in addition to having a universe and Π types, our theory also supports unit and Σ types, we can actually derive a CwF-based model from an SSC-model. The idea is that Σ types can represent contexts, functions between Σ types represent parallel substitutions, functions into the universe represent dependent types. Then the SSC is just there for bootstrapping purposes: SSC formulates Σ, Π, U, and then we use these to define a parallel substitution calculus. In summary, although SSC models without any type formers are weaker than CwFs, if we equip them with enough type formers, the SSC model gives rise to a CwF. Doing a roundtrip CwF $\longrightarrow$ SSC $\longrightarrow$ CwF results in a contextually isomorphic CwF.

All of the constructions in this paper can be understood as happening internal to extensional type theory, and most results were formalised in Agda (see Subsection 1.2). Some results in this paper were presented at the TYPES 2024 conference [41].

In summary, our contributions are (in the same order as the structure of the paper):

Section 2 A new generalised algebraic presentation of type theory in the form of a minimalistic single substitution calculus. Our calculus does not feature parallel substitutions/weakenings, empty substitution, telescopes. We present our calculus in an easy-to-understand way where all operations are well-motivated.

Section 3 The α-normalisation technique which shows that the syntax of our single substitution calculus is isomorphic to the syntax of the CwF-based theory.

Section 4 For type theory with Π, U, a minimised presentation of the equations which results in an isomorphic theory.

Section 5 For type theory with Π, U, $\top$, Σ, we show that a CwF structure is derivable.

1.1 Related work

B-systems. B-systems introduced by Voevodsky [54] are an intrinsic, essentially algebraic presentation of type theory using single substitutions. B-systems relate to our SSC as essentially algebraic theories relate to generalised algebraic theories, or sets with a map into I relate to indexed families over I [22, 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. A further difference is that contexts in B-systems are inductively defined (we call such models *contextual*). However in the syntax of our theory, all the rules

of B-systems are admissible. We describe the relationship in more detail in Appendix A. Kaposi and Luksa [39] defined a telescopic SSC 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. If we have function space, the same equivalence holds for the simply typed version of our substitution calculus (𝓉𝒻).

Ehrhard's calculus. The first generalised algebraic presentation of type theory was Ehrhard's calculus [31, 28] which featured a parallel substitution calculus with p, $-^+$ and $\langle - \rangle$ operations, just like our calculus. Our single substitution calculus is essentially Ehrhard's calculus with the categorical composition and identity operations removed. Categories with families (CwFs [30, 23]) feature p, q, $(-,-)$ operations which make it more apparent that substitutions are lists of terms. CwFs are equivalent to Ehrhard's calculus and the natural models of Awodey [15]. Contextual versions of these are equivalent to contextual categories/C-systems [22, 6], and B-systems [4]. For the precise statements of equivalences, see e.g. [5].

Formalisations of type theory. Most computer formalisations of type theory are extrinsic [2, 3, 49]: 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 [24, 8]: terms are intrinsically well-typed, but conversion is still an explicit relation. For formalising the GAT-level syntax, one needs a stronger metatheory than plain Agda or Coq: the metatheory has to support quotient inductive-inductive types (QIITs [37]), in other words, initial models of GATs. Altenkirch and Kaposi [10] formalised the syntax of type theory using postulated QIITs in Agda, together with parametricity and normalisation [11] proofs. Brunerie and de Boer [21] constructed the initial contextual category in Agda using postulated quotients. Although we don't have a general proof, we have experimental evidence that QIITs are supported by the cubical set model [29] and the setoid model [42], thus Cubical Agda [53] and setoid/observational type theory [9, 48] support QIITs. For example, in Cubical Agda, set-truncated and groupoid-truncated syntaxes of type theory have been formalised [12]. Without QIITs, tricks such as shallowly embedding the syntax can be used to formalise metatheoretic results about type theory such as gluing [36], and its special cases canonicity and parametricity [38].

SOGATs We can define languages with binders as second-order theories following higher-order abstract syntax [34] and logical frameworks [33, 46]. Second-order generalised algebraic theories (SOGATs [52, 40]) can be used to present type theory more abstractly than GATs: SOGATs abstract over the exact definition of the substitution calculus (whether substitutions are single or parallel; whether we have CwF-style or Ehrhard-style operations): contexts, substitutions, instantiations are simply modelled by the metatheoretic function space. For example, a type theory with Π, Coquand-universes [27] and a universe level lifting operation is described by the following SOGAT (where $\cong$ means isomorphism):

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

Second-order models of SOGATs are not well-behaved, as there is no good notion of homomorphism between them [40, bottom of page 5]. This is why Kaposi and Xie [40] translate SOGATs to GATs: a model of a SOGAT then is a model of the GAT that we obtain by their translation. GATs come with well-behaved metatheory: a category of models with initial model (syntax), (co)free models [43, 45], and so on. However, the translation is not unique: Kaposi and Xie [40] define two different translations, one

based on parallel substitutions and one based on single substitutions. If we apply their parallel translation to the SOGAT

$$\mathsf{Ty} : \mathsf{Set} \qquad \mathsf{Tm} : \mathsf{Ty} \to \mathsf{Set} \tag{2}$$

with only two sorts and no operations or equations, we obtain the GAT of categories with families (CwFs). If we apply their single substitution translation to the SOGAT (1), we obtain Definition 1 (also listed in Appendix B). The current paper can be seen as taking a big enough type theory (described as a SOGAT), and investigating how the result of its single substitution translation relates to the result of its parallel translation (the latter first-order theory is quite well-known).

There are techniques to prove properties of type theories without choosing between parallel or single substitutions, and staying at the SOGAT level of abstraction: Synthethic Tait Computability [50] and internal sconing [19].

1.2 Metatheory and formalisation

Our metatheory is observational type theory [48] with quotient inductive-inductive types (QIITs). On paper we usually omit writing coercions, so our notation is close to extensional type theory. Definition 1 and Section 3 of this paper are formalised in the proof assistant Agda. The formalisation is available online (). The Agda logos next to definitions/theorems are links to their formalised counterparts. We use a strict Prop-valued [32] equality type with postulated coercion rule (transport rule) and postulated QIITs with computation rules added using rewrite rules [26]. Thus, we work in a setting of uniqueness of identity proofs (UIP), and our metatheory is incompatible with homotopy type theory [51]. Our notation on paper is close to Agda's: we write Π types as $(x : A) \to B$, Σ types as $(x : A) \times B$, we use implicit arguments and overloading extensively, for example instantiation $-[-]$ is overloaded for types and terms. For isomorphisms, we use the notation $f : X \cong Y : g$ meaning $f : X \to Y$ and $g : Y \to X$ together with a β equality $f\,(g\,y) = y$ for all y, and an η equality saying $g\,(f\,x) = x$ for all x (we think of f as a destructor and g as a constructor). Following Voevodsky [55], we call a proof relevant theorem a *problem* and its proof a *construction*. The words theorem, lemma, proof refer to propositions.

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.

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.

$$\mathsf{Con} : \mathsf{Set} \qquad \mathsf{Ty} : \mathsf{Con} \to \mathbb{N} \to \mathsf{Set} \qquad \mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Set}$$

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 preceding the type.

$$\diamond : \mathsf{Con} \qquad - \triangleright - : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\, i \to \mathsf{Con}$$

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, first of all we need an operation providing the last variable in the context. This is the zero De Bruijn index [20], which we denote 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. For now, Sub only has the single element p and we introduce the operation $-[-]$ called *instantiation*. Elements of Sub will later be called substitutions, hence the name, but for now, we only have single end-of context weakenings in Sub. The context Δ is called the domain, Γ the codomain of a $\mathsf{Sub}\,\Delta\,\Gamma$.

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

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 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.

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

The second and third equations only make sense because of the first equation. The phenomenon that later equations/operations depend on previous equations is common in GATs.

Now that we have $[\mathsf{p}][^+]$ for types, we can derive its instantiation rule by $(A \Rightarrow B)[\gamma] = \Pi A\,(B[\mathsf{p}])[\gamma] \overset{\Pi[]}{=} \Pi\,(A[\gamma])\,(B[\mathsf{p}][\gamma^+]) \overset{[\mathsf{p}][^+]}{=} \Pi\,(A[\gamma])\,(B[\gamma][\mathsf{p}]) = (A[\gamma]) \Rightarrow (B[\gamma])$.

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 $\Pi[]$.

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

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 $\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}])) : \mathsf{Tm}\,\Gamma\,\big(\Pi A\ (B \Rightarrow (A[\mathsf{p}]))\big)$ 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.

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

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, as the equations relating substitutions are defined for their action on types and terms. An element of Sub is either a single weakening p lifted a finite number of times, or a single substitution lifted a finite number 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.

$$[\langle\rangle][] : B[\langle a\rangle][\gamma] = B[\gamma^+][\langle a[\gamma]\rangle] \qquad \cdot[] : (t\cdot a)[\gamma] = (t[\gamma])\cdot(a[\gamma])$$

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.

$$[\mathsf{p}][\langle\rangle] : B[\mathsf{p}][\langle a\rangle] = B \qquad [\mathsf{p}][\langle\rangle] : b[\mathsf{p}][\langle a\rangle] = b \qquad \mathsf{q}[\langle\rangle] : \mathsf{q}[\langle a\rangle] = a$$

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 (we mentioned the term version of this equation in the introduction).

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

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.

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

Finally, we add the rules for moving types one level up. Because Π needs two types at the same level, without lifting, we cannot even define the polymorphic identity function.

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

Note that we don't have definitional commutation of type formers and Lift such as $\mathsf{Lift}\,(A \Rightarrow B) = (\mathsf{Lift}\,A \Rightarrow \mathsf{Lift}\,B)$. It however holds as a definitional isomorphism (see Subsection 2.1). This adds extra administration when using our theory, but makes stating our theory simpler, and includes more models. The situation is similar to the correspondence between $\mathsf{Ty}\,\Gamma\,i$ and $\mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i)$ which could have been stated as an equality (making universes à la Russell).

Definition 1 (Single substitution calculus with Π, U, Lift (ツ))**.** *This concludes the definition of our basic type theory. For reference, we list the same rules again in Appendix B.*

As a sanity check for our notion of SSC-model, we defined the standard (metacircular/set/type) model [10] of the SSC-based calculus in Agda (ツ).

The theory can be extended with new type and term formers in the same fashion: each operation has to be indexed over contexts and come with a substitution rule. There is no need to add more structural (substitution calculus) rules.[1] For example, we show how to extend Definition 1 with $\top$ and Σ types.

Definition 2 (Single substitution calculus with Π, U, Lift, $\top$, and Σ types)**.** *We extend Definition 1 with the following rules.*

$$\top : \mathsf{Ty}\,\Gamma\,0 \qquad \top[] : \top[\gamma] = \top \qquad \top\eta : t = \mathsf{tt} \qquad \mathsf{tt} : \mathsf{Tm}\,\Gamma\,\top \qquad \mathsf{tt}[] : \mathsf{tt}[\gamma] = \mathsf{tt}$$

$$\Sigma \quad : (A : \mathsf{Ty}\,\Gamma\,i) \to \mathsf{Ty}\,(\Gamma \triangleright A)\,i \to \mathsf{Ty}\,\Gamma\,i \qquad \Sigma[] : (\Sigma A B)[\gamma] = \Sigma\,(A[\gamma])\,(B[\gamma^+])$$

$$\mathsf{fst}, \mathsf{snd} : \mathsf{Tm}\,\Gamma\,(\Sigma A B) \cong (a : \mathsf{Tm}\,\Gamma A) \times \mathsf{Tm}\,\Gamma\,(B[\langle a \rangle]) : -,- \qquad ,[] : (a,b)[\gamma] = (a[\gamma], b[\gamma])$$

To save space, we compressed the introduction, elimination, β and η rules of Σ into an isomorphism, and did not list substitution law for fst, snd *as they are derivable (see below).*

2.1 Examples

In this subsection, we show how to use the above defined calculus, and illustrate its deficiencies: some important equations are not derivable in any model, only admissible in the syntax. These equations will be proven in Section 3.

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

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

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. Intrinsically typed terms are the same as derivation trees, we illustrate this by deriving this term as follows.

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

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, which is why we had to lift $\mathsf{El}\,\mathsf{q}$. Another subtlety is happening when deriving the third implicit argument of q, because we have to coerce q along the equality $\mathsf{U}[]$:

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

[1] In [40], the same structural rules suffice to describe the first-order theory for any SOGAT.

When working informally (or in extensional type theory), we don't write coercions.

Given a type $A : \mathsf{Ty}\,\diamond\,0$, and $a : \mathsf{Tm}\,\diamond\,A$, we have $\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q})\cdot \mathsf{c}A\cdot a \overset{\Pi\beta}{=} \mathsf{lam}\,\mathsf{q}[\langle \mathsf{c}A\rangle]\cdot a \overset{\mathsf{lam}[]}{=} \mathsf{lam}\,(\mathsf{q}[\langle \mathsf{c}A\rangle^+])\cdot a \overset{[\mathsf{q}][^+]}{=} \mathsf{lam}\,\mathsf{q}\cdot a \overset{\Pi\beta}{=} \mathsf{q}[\langle a\rangle] \overset{\mathsf{q}[\langle\rangle]}{=} a$. 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: $\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q})[\mathsf{p}] : \mathsf{Tm}\,(\diamond\triangleright C)\,(\Pi\,(\mathsf{U}\,0)\,(\mathsf{Lift}\,(\mathsf{El}\,\mathsf{q})\Rightarrow \mathsf{Lift}\,(\mathsf{El}\,\mathsf{q}))[\mathsf{p}])$. But we have $\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q})[\mathsf{p}] \overset{\mathsf{lam}[]\,2\mathrm{x}}{=} \mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}^{++}])) \overset{\mathsf{q}[^+]}{=} \mathsf{lam}\,(\mathsf{lam}\,\mathsf{q})$ (with different implicit arguments for the lams on the left and right hand side), so we are able to apply $\mathsf{c}A'$ and a' just as before. What we cannot derive is to directly weaken a term in context $\diamond$ to an arbitrary context Γ, where Γ is a metavariable. If we know that Γ has length n, then we can do the weakening by applying $-[\mathsf{p}]$ to it n times. If we work in the syntax, then we can derive such weakenings (and more) using the methods of Section 3.

The operations for U and Lift can be written more concisely as the following isomorphisms:

$$\mathsf{El} : \mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i) \cong \mathsf{Ty}\,\Gamma\,i : \mathsf{c} \qquad\qquad \mathsf{un} : \mathsf{Tm}\,\Gamma\,(\mathsf{Lift}\,A) \cong \mathsf{Tm}\,\Gamma\,A : \mathsf{mk}$$

The $\mathsf{U}\beta$, $\mathsf{U}\eta$ and $\mathsf{Lift}\beta$, $\mathsf{Lift}\eta$ rules express that the round-trips are identities. These isomorphisms are also *natural*, or compatible with substitutions: these are expressed by the equations $\mathsf{El}[]$, $\mathsf{un}[]$, $\mathsf{c}[]$, $\mathsf{mk}[]$.

Σ types were introduced by a similar isomorphism, and we give names to the round-trips equations according to our convention introduced in Section 1.2: $\Sigma\beta_1 : \mathsf{fst}\,(a,b) = a$, $\Sigma\beta_2 : \mathsf{snd}\,(a,b) = b$ and $\Sigma\eta : w = (\mathsf{fst}\,w, \mathsf{snd}\,w)$. Naturality for Σ was only stated in one direction $(,[])$, naturalities in the other direction are derivable, for fst this is $(\mathsf{fst}\,w)[\gamma] \overset{\Sigma\beta_1}{=} \mathsf{fst}\,\big((\mathsf{fst}\,w)[\gamma], (\mathsf{snd}\,w)[\gamma]\big) \overset{,[]}{=} \mathsf{fst}\,\big((\mathsf{fst}\,w, \mathsf{snd}\,w)[\gamma]\big) \overset{\Sigma\eta}{=} \mathsf{fst}\,(w[\gamma])$. For Π types, we construct a similar isomorphism $(\lambda t.\, t[\mathsf{p}]\cdot\mathsf{q}) : \mathsf{Tm}\,\Gamma\,(\Pi A\,B) \cong \mathsf{Tm}\,(\Gamma\triangleright A)\,B : \mathsf{lam}$. The first round-trip equality is proven as $\beta : (\mathsf{lam}\,t)[\mathsf{p}]\cdot\mathsf{q} \overset{\mathsf{lam}[]}{=} \mathsf{lam}\,(t[\mathsf{p}^+])\cdot\mathsf{q} \overset{\Pi\beta}{=} t[p^+][\langle\mathsf{q}\rangle] \overset{(7)}{=} t$. The last step is the admissible equation (7) from Section 3. As above with weakening into an arbitrary context, this equation holds in the syntax, but not in an arbitrary model. The other round-trip is exactly $\Pi\eta : \mathsf{lam}\,(t[\mathsf{p}]\cdot\mathsf{q}) = t$.

We construct another isomorphism between terms depending on a variable and terms depending on the lifted version of the same variable:

$$(\lambda t.\, t[\mathsf{p}^+][\langle \mathsf{un}\,\mathsf{q}\rangle]) : \mathsf{Tm}\,(\Gamma\triangleright A)\,B \cong \mathsf{Tm}\,(\Gamma\triangleright \mathsf{Lift}\,A)\,(B[\mathsf{p}^+][\langle\mathsf{q}\rangle]) : (\lambda t.\, t[\mathsf{p}^+][\langle \mathsf{mk}\,\mathsf{q}\rangle]) \tag{3}$$

To derive the round-trips, we again need admissible equations. We prove the first round-trip equality in Appendix B, the other is analogous. Once we have all the above isomorphisms, we can compose them to obtain commutation of Lift and Π:

$$\mathsf{Tm}\,\Gamma\,\big(\mathsf{Lift}\,(\Pi A\,B)\big) \overset{\mathsf{Lift}}{\cong} \mathsf{Tm}\,\Gamma\,(\Pi A\,B) \overset{\Pi}{\cong} \mathsf{Tm}\,(\Gamma\triangleright A)\,B \overset{\mathsf{Lift}}{\cong} \mathsf{Tm}\,(\Gamma\triangleright A)\,(\mathsf{Lift}\,B) \overset{\text{lifted var}}{\cong}$$
$$\mathsf{Tm}\,(\Gamma\triangleright \mathsf{Lift}\,A)\,(\mathsf{Lift}\,B[\mathsf{p}^+][\langle \mathsf{un}\,\mathsf{q}\rangle]) \overset{\Pi}{\cong} \mathsf{Tm}\,\Gamma\,\big(\Pi\,(\mathsf{Lift}\,A)\,(\mathsf{Lift}\,B[\mathsf{p}^+][\langle \mathsf{un}\,\mathsf{q}\rangle])\big)$$

We hope that the above examples demonstrated the need for equations (4)–(7) which we will prove in the next section.

3 Admissible equations

In this section we show that our SSC-based syntax is isomorphic to the CwF-based syntax with the same type formers.[2]

[2]Using the two translations from SOGATs to GATs [40], we can rephrase the contents of this section as follows: the parallel and single substitution syntax of the SOGAT (1) are isomorphic.

Definition 3 (CwF (see 😉))**.** *A* category with $\mathbb{N}$-many families *is defined as a category (objects* 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*).*

In the rest of this section, we work with the SSC syntax denoted Con, Sub, Ty, Tm, and so on, towards the goal of defining all CwF operations and equations.

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.

$$B[\mathsf{p}^{+^n}][(\gamma^+)^{+^n}]=B[\gamma^{+^n}][\mathsf{p}^{+^n}] \quad (4) \qquad B[\langle a\rangle^{+^n}][\gamma^{+^n}]=B[(\gamma^+)^{+^n}][\langle a[\gamma]\rangle^{+^n}] \quad (6)$$

$$B[\mathsf{p}^{+^n}][\langle a\rangle^{+^n}]=B \quad (5) \qquad B[(\mathsf{p}^+)^{+^n}][\langle \mathsf{q}\rangle^{+^n}]=B \quad (7)$$

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 equations (4)–(7), we define telescopes as an inductive type together with a recursive operation to append telescopes to contexts (😉).

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

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

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

If we try to prove the equations in (4)–(7) 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.
- $f\cdot a:\mathsf{Tm}\,\Gamma\,(B[\langle a\rangle])$ 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). We truncate the α-normal predicate to be a proposition to allow interpreting into it from the quotiented syntax. Without truncation, interpretation into α-normal forms would expose differences between β/η-equal terms (and such an interpretation is not definable).

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 structural. 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 4 (🙂)**.** *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 equations (4)–(7), 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 category laws as we will not compare Sub^*s directly for equality. All instantiation operations, lifting operations, and substitution rules are redefined for Sub^* (🙂).

Lemma 5. *Given $\gamma_0, \gamma_1 : \mathsf{Sub}^* \Delta \Gamma$, if for any $A : \mathsf{Ty}\,\Gamma$, $A[\gamma_0] = A[\gamma_1]$, and for any variable $x : \mathsf{Tm}\,\Gamma A$, $x[\gamma_0] = x[\gamma_1]$, 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 on 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 are 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 lists of terms. It seems to be difficult to define the instantiation operation for Tms directly, so we also define a map into

Sub^*, and reuse the instantiation operation of Sub^* to convert it into a sequence of instantiations of single substitutions (🙁).

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

The computation of instantiating with Tms (🙁) is illustrated below. On the right-hand side, we first weaken B to the context in which the a_i's live, then replace the variables of B one by one using the a_i's, which are well-typed because of the previous weakening.

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

All CwF operations and equations can be defined with Tms by induction, using Lemma 5 to avoid further induction on the syntax (🙁).

Problem 6 (🙁)**.** *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.*

Construction.

$\Rightarrow$ By recursion on the syntax, SSC operations can be trivially interpreted by CwF operations.

$\Leftarrow$ By recursion on the syntax, using Tms to interpret parallel substitutions.

The round-trips are proven by induction on the syntax. □

This also implies that the initial SSC model is the initial CwF.

4 Minimisation

In this section we show that if have 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 (here B is a type and b is a term of type B): $B[\mathsf{p}][\gamma^+] = B[\gamma][\mathsf{p}]$ and $b[\mathsf{p}][\gamma^+] =_{[\mathsf{p}][^+]} b[\gamma][\mathsf{p}]$. 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 $[\mathsf{p}][^+]' : (e : B[\mathsf{p}][\gamma^+] = B[\gamma][\mathsf{p}]) \to b[\mathsf{p}][\gamma^+] =_e b[\gamma][\mathsf{p}]$. That is, for any type B for which $B[\mathsf{p}][\gamma^+] = B[\gamma][\mathsf{p}]$, we have the equation (suitably over the input equation) for any b. 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 $B : \mathsf{Ty}\,\Gamma\,i$, $B[\mathsf{p}][\gamma^+] \overset{\mathsf{U}\beta}{=} \mathsf{El}\,(\mathsf{c}B)[\mathsf{p}][\gamma^+] \overset{\mathsf{El}[]}{=} \mathsf{El}\,((\mathsf{c}B)[\mathsf{p}][\gamma^+]) \overset{[\mathsf{p}][^+]'\mathsf{U}[]}{=} \mathsf{El}\,((\mathsf{c}B)[\gamma][\mathsf{p}]) \overset{\mathsf{El}[]}{=} \mathsf{El}\,(\mathsf{c}B)[\gamma][\mathsf{p}] \overset{\mathsf{U}\beta}{=} B[\gamma][\mathsf{p}]$. Thus, as the input of equation $[\mathsf{p}][^+]'$ holds all the time, we get this equation for every term. 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 $\cdot[]$. A conditional version of $\cdot[]$ is $\cdot[]' : (e : B[\langle a\rangle][\gamma] = B[\gamma^+][\langle a[\gamma]\rangle]) \to (t \cdot a)[\gamma] =_e (t[\gamma]) \cdot (a[\gamma])$. 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 $B : \mathsf{Ty}\,(\Gamma \triangleright A)$: $B[\langle a\rangle][\gamma] \overset{\mathsf{U}\beta}{=} \mathsf{El}\,(\mathsf{c}\,(B[\langle a\rangle][\gamma])) \overset{\mathsf{c}[]}{=} \mathsf{El}\,((\mathsf{c}B)[\langle a\rangle][\gamma]) \overset{\Pi\beta}{=} \mathsf{El}\,((\mathsf{lam}\,(\mathsf{c}B) \cdot a)[\gamma]) \overset{\cdot[]'\mathsf{U}[]}{=} \mathsf{El}\,((\mathsf{lam}\,(\mathsf{c}B)[\gamma]) \cdot (a[\gamma])) \overset{\mathsf{lam}[]}{=} \mathsf{El}\,(\mathsf{lam}\,(\mathsf{c}B[\gamma^+]) \cdot (a[\gamma])) \overset{\Pi\beta}{=} \mathsf{El}\,(\mathsf{c}B[\gamma^+][\langle a[\gamma]\rangle]) \overset{\mathsf{c}[]}{=} \mathsf{El}\,(\mathsf{c}B)[\gamma^+][\langle a[\gamma]\rangle] \overset{\mathsf{U}\beta}{=} B[\gamma^+][\langle a[\gamma]\rangle]$. Hence, the conditional $\cdot[]'$ equation implies the condition for all types.

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

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

But first we need to verify that the instance of $\Pi\beta$ used when proving $B[\langle a\rangle][\gamma] = B[\gamma^+][\langle a[\gamma]\rangle]$ is derivable from $\Pi\beta'$. Assuming $\hat{B} : \mathsf{Tm}\,(\Gamma \triangleright A)\,\mathsf{U}$, we argue $\Pi\beta^{\mathsf{U}} : \mathsf{lam}\,\hat{B} \cdot a \overset{\mathsf{q}[\langle\rangle]}{=} \mathsf{lam}\,\hat{B} \cdot (\mathsf{q}[\langle a\rangle]) \overset{[\mathsf{p}][\langle\rangle]}{=} ((\mathsf{lam}\,\hat{B})[\mathsf{p}][\langle a\rangle]) \cdot (\mathsf{q}[\langle a\rangle]) \overset{\cdot[]'\mathsf{U}[]}{=} ((\mathsf{lam}\,\hat{B})[\mathsf{p}] \cdot \mathsf{q})[\langle a\rangle] \overset{\Pi\beta'\mathsf{U}[]}{=} \hat{B}[\langle a\rangle]$. Now, for any type $B : \mathsf{Ty}\,(\Gamma \triangleright A)$, we derive the input of $\Pi\beta'/\Pi\eta'$ by $B[\mathsf{p}^+][\langle\mathsf{q}\rangle] \overset{\mathsf{U}\beta}{=} \mathsf{El}\,(\mathsf{c}B)[\mathsf{p}^+][\langle\mathsf{q}\rangle] \overset{\mathsf{El}[]}{=} \mathsf{El}\,(\mathsf{c}B[\mathsf{p}^+][\langle\mathsf{q}\rangle]) \overset{\Pi\beta^{\mathsf{U}}}{=} \mathsf{El}\,(\mathsf{lam}\,(\mathsf{c}B[\mathsf{p}^+]) \cdot \mathsf{q}) \overset{\mathsf{lam}[]}{=} \mathsf{El}\,(\mathsf{lam}\,(\mathsf{c}B)[\mathsf{p}] \cdot \mathsf{q}) \overset{\Pi\beta'\mathsf{U}[]}{=} \mathsf{El}\,(\mathsf{c}B) \overset{\mathsf{U}\beta}{=} B$. We summarise this section by formally stating the above.

Definition 7 (Minimised single substitution calculus with Π and U)**.** *This section defined the minimised version of Definition 1, relying essentially on the presence of universes and Π types. For reference, we list all the rules of the minimised calculus in Appendix C.*

Problem 8. *The GATs of Definition 7 and Definition 1 are isomorphic, in particular, all equations are interderivable.*

Construction. 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 a CwF structure is derivable in any model. The idea is that (i) using Σ types we emulate contexts; (ii) using functions between these Σ types we emulate parallel substitutions; (iii) using the functions into the universe we emulate dependent types. 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 9. *Every CwF (with type formers $\Pi, \ldots, \Sigma$) gives rise to an SSC (with the same type formers).*

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 [10], contextualisation [19]. Following [38] we call it termification, as most sorts in the new model are given by the sort of terms in the old model.

Problem 10 (Termification)**.** *From a model of SSC (with $\Pi, \ldots, \Sigma$, see Definition 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):

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

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 $\top$, its η law holds via η for $\top$.

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

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

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

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

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

To make the notation readable, from now on, we will not write the lifting decorations or universe levels. We repeat the previous definitions again without writing decorations.

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

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 in Appendix D. 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. □

In Appendix D, we show that the roundtrip CwF $\longrightarrow$ SSC $\xrightarrow{\text{termification}}$ CwF results in a contextually isomorphic CwF.

6 Conclusions and further work

We described type theory in a minimalistic way, without referring to categories or parallel substitutions. Our presentation has pedagogical value, as illustrated in Section 2. It is amazing that any type theory can be described with such a minimal substitution calculus: the lifted equations (4)–(7) are not required. We use them when proving properties of the syntax (such as normalisation), but they are admissible.

In the future, we would like to investigate whether the calculus can be minimised even more, e.g. to the degree that there is at most one proof for any equation. This would be interesting for a possible coherent syntax of type theory avoiding the need for truncation in the setting of homotopy type theory.

References

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

[2] Andreas Abel, Joakim Öhman & Andrea Vezzosi (2018): *Decidability of conversion for type theory in type theory*. *Proc. ACM Program. Lang.* 2(POPL), pp. 23:1–23:29, doi:10.1145/3158111.

[3] Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Pierre-Marie Pédrot & Loïc Pujet (2024): *Martin-Löf à la Coq*. In Amin Timany, Dmitriy Traytel, Brigitte Pientka & Sandrine Blazy, editors: *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024*, ACM, pp. 230–245, doi:10.1145/3636501.3636951.

[4] Benedikt Ahrens, Jacopo Emmenegger, Paige Randall North & Egbert Rijke (2023): *B-systems and C-systems are equivalent*. *The Journal of Symbolic Logic*, p. 1–9, doi:10.1017/jsl.2023.41.

[5] Benedikt Ahrens, Peter LeFanu Lumsdaine & Paige Randall North (2024): *Comparing Semantic Frameworks for Dependently-Sorted Algebraic Theories*. In Oleg Kiselyov, editor: *Programming Languages and Systems - 22nd Asian Symposium, APLAS 2024, Kyoto, Japan, October 22-24, 2024, Proceedings*, *Lecture Notes in Computer Science* 15194, Springer, pp. 3–22, doi:10.1007/978-981-97-8943-6_1.

[6] Benedikt Ahrens, Peter LeFanu Lumsdaine & Vladimir Voevodsky (2018): *Categorical structures for type theory in univalent foundations*. *Log. Methods Comput. Sci.* 14(3), doi:10.23638/LMCS-14(3:18)2018.

[7] Thorsten Altenkirch (1993): *Constructions, Inductive Types and Strong Normalization*. Ph.D. thesis, University of Edinburgh.

[8] Thorsten Altenkirch (2017): *From setoid hell to homotopy heaven? The role of extensionality in Type Theory*. In Ambrus Kaposi, editor: *23rd International Conference on Types for Proofs and Programs, TYPES 2017*, Eötvös Loránd University. Available at `https://types2017.elte.hu/proc.pdf#page=20`.

[9] Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi & Nicolas Tabareau (2019): *Setoid Type Theory - A Syntactic Translation*. In Graham Hutton, editor: *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings*, *Lecture Notes in Computer Science* 11825, Springer, pp. 155–196, doi:10.1007/978-3-030-33636-3_7.

[10] Thorsten Altenkirch & Ambrus Kaposi (2016): *Type theory in type theory using quotient inductive types*. In Rastislav Bodík & 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*, ACM, pp. 18–29, doi:10.1145/2837614.2837638.

[11] Thorsten Altenkirch & Ambrus Kaposi (2017): *Normalisation by Evaluation for Type Theory, in Type Theory*. *Logical Methods in Computer Science* Volume 13, Issue 4, doi:10.23638/LMCS-13(4:1)2017. Available at `https://lmcs.episciences.org/4005`.

[12] Thorsten Altenkirch & Ambrus Kaposi (2024): *Coherent Categories with Families*. In Patrick Bahr & Rasmus Ejlers Møgelberg, editors: *30th International Conference on Types for Proofs and Programs (TYPES 2024)*, Copenhagen. Available at `https://types2024.itu.dk/abstracts.pdf#page=78`.

[13] Thorsten Altenkirch, Ambrus Kaposi, Artjoms Sinkarovs & Tamás Végh (2023): *Combinatory Logic and Lambda Calculus Are Equal, Algebraically*. In Marco Gaboardi & Femke van Raamsdonk, editors: *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, *LIPIcs* 260, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 24:1–24:19, doi:10.4230/LIPICS.FSCD.2023.24.

[14] Thorsten Altenkirch & Bernhard Reus (1999): *Monadic Presentations of Lambda Terms Using Generalized Inductive Types*. In Jörg Flum & 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*, *Lecture Notes in Computer Science* 1683, Springer, pp. 453–468, doi:10.1007/3-540-48168-0_32.

[15] Steve Awodey (2018): *Natural models of homotopy type theory*. Math. Struct. Comput. Sci. 28(2), pp. 241–286, doi:10.1017/S0960129516000268.

[16] Benno van den Berg (2018): *Path Categories and Propositional Identity Types*. ACM Trans. Comput. Log. 19(2), pp. 15:1–15:32, doi:10.1145/3204492.

[17] Jean-Philippe Bernardy, Patrik Jansson & Ross Paterson (2012): *Proofs for free - Parametricity for dependent types*. J. Funct. Program. 22(2), pp. 107–152, doi:10.1017/S0956796812000056.

[18] Rafaël Bocquet (2022): *External univalence for second-order generalized algebraic theories*. CoRR abs/2211.07487, doi:10.48550/ARXIV.2211.07487. arXiv:2211.07487.

[19] Rafaël Bocquet, Ambrus Kaposi & Christian Sattler (2023): *For the Metatheory of Type Theory, Internal Sconing Is Enough*. In Marco Gaboardi & Femke van Raamsdonk, editors: *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, *LIPIcs* 260, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 18:1–18:23, doi:10.4230/LIPICS.FSCD.2023.18.

[20] N. G. de Bruijn (1972): *Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem*. Indagationes Mathematicae (Proceedings) 75(5), pp. 381–392, doi:10.1016/1385-7258(72)90034-0. Available at `https://www.sciencedirect.com/science/article/pii/1385725872900340`.

[21] Guillaume Brunerie & Menno de Boer (2020): *Formalization of the initiality conjecture*. Available at `https://github.com/guillaumebrunerie/initiality`.

[22] John Cartmell (1986): *Generalised algebraic theories and contextual categories*. Ann. Pure Appl. Log. 32, pp. 209–243, doi:10.1016/0168-0072(86)90053-9.

[23] Simon Castellan, Pierre Clairambault & Peter Dybjer (2021): *Categories with Families: Unityped, Simply Typed, and Dependently Typed*, pp. 135–180. Springer International Publishing, Cham, doi:10.1007/978-3-030-66545-6_5.

[24] James Chapman (2009): *Type Theory Should Eat Itself*. Electronic Notes in Theoretical Computer Science 228, pp. 21–36, doi:10.1016/j.entcs.2008.12.114.

[25] Pierre Clairambault & Peter Dybjer (2014): *The biequivalence of locally cartesian closed categories and Martin-Löf type theories*. Math. Struct. Comput. Sci. 24(6), doi:10.1017/S0960129513000881.

[26] Jesper Cockx, Nicolas Tabareau & Théo Winterhalter (2021): *The taming of the rew: a type theory with computational assumptions*. Proc. ACM Program. Lang. 5(POPL), pp. 1–29, doi:10.1145/3434341.

[27] Thierry Coquand (2018): *Presheaf model of type theory*. `https://www.cse.chalmers.se/~coquand/presheaf.pdf`.

[28] Thierry Coquand (2020): *Generalised algebraic presentation of type theory*. `https://www.cse.chalmers.se/~coquand/cwf2.pdf`.

[29] Thierry Coquand, Simon Huber & Anders Mörtberg (2018): *On Higher Inductive Types in Cubical Type Theory*. In Anuj Dawar & Erich Grädel, editors: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, ACM, pp. 255–264, doi:10.1145/3209108.3209197.

[30] Peter Dybjer (1995): *Internal Type Theory*. In Stefano Berardi & Mario Coppo, editors: *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*, *Lecture Notes in Computer Science* 1158, Springer, pp. 120–134, doi:10.1007/3-540-61780-9_66.

[31] Thomas Ehrhard (1988): *Une sémantique catégorique des types dépendents*. Ph.D. thesis, Université Paris VII.

[32] Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau & Nicolas Tabareau (2019): *Definitional proof-irrelevance without K*. Proc. ACM Program. Lang. 3(POPL), pp. 3:1–3:28, doi:10.1145/3290316.

[33] Robert Harper, Furio Honsell & Gordon D. Plotkin (1993): *A Framework for Defining Logics*. J. ACM 40(1), pp. 143–184, doi:10.1145/138027.138060.

[34] Martin Hofmann (1999): *Semantical Analysis of Higher-Order Abstract Syntax*. In: *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, IEEE Computer Society, pp. 204–213, doi:10.1109/LICS.1999.782616.

[35] Ambrus Kaposi (2018): *Formalisation of type checking into algebraic syntax*. `https://bitbucket.org/akaposi/tt-in-tt/src/master/Typecheck.agda`.

[36] Ambrus Kaposi, Simon Huber & Christian Sattler (2019): *Gluing for Type Theory*. In Herman Geuvers, editor: *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 131, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 25:1–25:19, doi:10.4230/LIPIcs.FSCD.2019.25. Available at `http://drops.dagstuhl.de/opus/volltexte/2019/10532`.

[37] Ambrus Kaposi, András Kovács & Thorsten Altenkirch (2019): *Constructing quotient inductive-inductive types*. *Proc. ACM Program. Lang.* 3(POPL), pp. 2:1–2:24, doi:10.1145/3290315.

[38] Ambrus Kaposi, András Kovács & Nicolai Kraus (2019): *Shallow Embedding of Type Theory is Morally Correct*. In Graham Hutton, editor: *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings*, *Lecture Notes in Computer Science* 11825, Springer, pp. 329–365, doi:10.1007/978-3-030-33636-3_12.

[39] Ambrus Kaposi & Norbert Luksa (2020): *A calculus of single substitutions for simple type theory*. `https://bitbucket.org/akaposi/tel_stt/raw/master/paper.pdf`.

[40] Ambrus Kaposi & Szumi Xie (2024): *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*, *LIPIcs* 299, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 10:1–10:24, doi:10.4230/LIPICS.FSCD.2024.10.

[41] Ambrus Kaposi & Szumi Xie (2024): *Type theory in type theory using single substitutions*. In Patrick Bahr & Rasmus Ejlers Møgelberg, editors: *30th International Conference on Types for Proofs and Programs (TYPES 2024)*, Copenhagen. Available at `https://types2024.itu.dk/abstracts.pdf#page=80`.

[42] Ambrus Kaposi & Zongpu Xie (2021): *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. Available at `https://types21.liacs.nl/download/quotient-inductive-inductive-types-in-the-setoid-model/`.

[43] András Kovács (2022): *Type-Theoretic Signatures for Algebraic Theories and Inductive Types*. Ph.D. thesis, Eötvös Loránd University, Hungary. Available at `https://arxiv.org/pdf/2302.08837.pdf`.

[44] Per Martin-Löf (1975): *An Intuitionistic Theory of Types: Predicative Part*. In H.E. Rose & J.C. Shepherdson, editors: *Logic Colloquium '73*, *Studies in Logic and the Foundations of Mathematics* 80, Elsevier, pp. 73–118, doi:10.1016/S0049-237X(08)71945-1. Available at `https://www.sciencedirect.com/science/article/pii/S0049237X08719451`.

[45] Hugo Moeneclaey (2022): *Cubical models are cofreely parametric. (Les modèles cubiques sont colibrement paramétriques)*. Ph.D. thesis, Paris Cité University, France. Available at `https://tel.archives-ouvertes.fr/tel-04435596`.

[46] Brigitte Pientka & Jana Dunfield (2010): *Beluga: A Framework for Programming and Reasoning with Deductive Systems (System Description)*. In Jürgen Giesl & Reiner Hähnle, editors: *Automated Reasoning*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 15–21, doi:10.1007/978-3-642-14203-1_2.

[47] Benjamin C. Pierce (2002): *Types and programming languages*. MIT Press.

[48] Loïc Pujet (2022): *Computing with Extensionality Principles in Type Theory. (Calculer avec des Principes d'Extensionnalité en Théorie des Types)*. Ph.D. thesis, University of Nantes, France. Available at `https://hal.archives-ouvertes.fr/tel-03923041`.

[49] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau & Théo Winterhalter (2020): *The MetaCoq Project*. *J. Autom. Reason.* 64(5), pp. 947–999, doi:10.1007/S10817-019-09540-0.

[50] Jonathan Sterling (2022): *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory*. Ph.D. thesis, Carnegie Mellon University, USA, doi:10.1184/R1/19632681.V1.

[51] The Univalent Foundations Program (2013): *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study.

[52] Taichi Uemura (2021): *Abstract and Concrete Type Theories*. Ph.D. thesis, University of Amsterdam.

[53] Andrea Vezzosi, Anders Mörtberg & Andreas Abel (2021): *Cubical Agda: A dependently typed programming language with univalence and higher inductive types*. *J. Funct. Program.* 31, p. e8, doi:10.1017/S0956796821000034.

[54] Vladimir Voevodsky (2014): *B-systems*. arXiv:1410.5389.

[55] Vladimir Voevodsky (2015): *A C-system defined by a universe category*. arXiv:1409.7925.

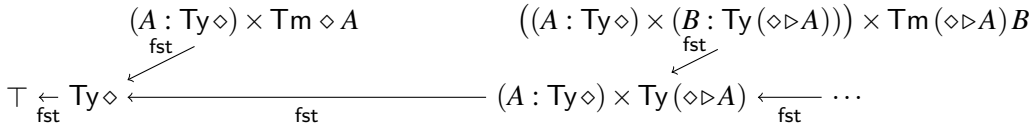
[56] Théo Winterhalter (2017): *A Restricted Version of Reflection Compatible with Univalent Homotopy Type Theory*. Available at `https://inria.hal.science/hal-01626651`. Mémoire étudiant.

A Detailed comparison of our SSC-calculus and B-systems

In this section, we describe the relationship between B-systems [4] and our Definition 1 in 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.

Using our notation, this is the following data ($\diamond$ is the empty context, $\triangleright$ is context extension).



The substitution operation $\mathbb{S}$ for an $x : \tilde{B}_{n+1}$ is a homomorphism of the slice B-frames $\mathbb{B}/\partial(x) \to \mathbb{B}/\mathsf{ft}(\partial(x))$. In our notation, $x = (\Gamma, A, a)$ where $a : \mathsf{Tm}\,\Gamma A$, and $\mathbb{S}$ corresponds to the following functions for Ty (a map 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 include the lifted versions of these 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 U and Π, we reduce the needed equations even more, see Section 4.

B Listings for Section 2

In Section 2, we presented the single substitution syntax interleaved with explanations (Definition 1), here we list all the rules in one place, for reference.

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.

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

We prove one of the round-trips in the isomorphism for lifting variables (3), the other is analogous:

$$
\begin{array}{ll}
t[\mathsf{p}^{+}][\langle\mathsf{un}\,\mathsf{q}\rangle][\mathsf{p}^{+}][\langle\mathsf{mk}\,\mathsf{q}\rangle] & =(6) \\
t[\mathsf{p}^{+}][\mathsf{p}^{++}][\langle\mathsf{un}\,\mathsf{q}[\mathsf{p}^{+}]\rangle][\langle\mathsf{mk}\,\mathsf{q}\rangle] & =(\mathsf{un}[]) \\
t[\mathsf{p}^{+}][\mathsf{p}^{++}][\langle\mathsf{un}\,(\mathsf{q}[\mathsf{p}^{+}])\rangle][\langle\mathsf{mk}\,\mathsf{q}\rangle] & =(\mathsf{q}[^{+}]) \\
t[\mathsf{p}^{+}][\mathsf{p}^{++}][\langle\mathsf{un}\,\mathsf{q}\rangle][\langle\mathsf{mk}\,\mathsf{q}\rangle] & =(6) \\
t[\mathsf{p}^{+}][\mathsf{p}^{++}][\langle\mathsf{mk}\,\mathsf{q}\rangle^{+}][\langle\mathsf{un}\,\mathsf{q}[\langle\mathsf{mk}\,\mathsf{q}\rangle]\rangle] & =(\mathsf{un}[]) \\
t[\mathsf{p}^{+}][\mathsf{p}^{++}][\langle\mathsf{mk}\,\mathsf{q}\rangle^{+}][\langle\mathsf{un}\,(\mathsf{q}[\langle\mathsf{mk}\,\mathsf{q}\rangle])\rangle] & =(\mathsf{q}[\langle\rangle]) \\
t[\mathsf{p}^{+}][\mathsf{p}^{++}][\langle\mathsf{mk}\,\mathsf{q}\rangle^{+}][\langle\mathsf{un}\,(\mathsf{mk}\,\mathsf{q})\rangle] & =(\mathsf{Lift}\beta) \\
t[\mathsf{p}^{+}][\mathsf{p}^{++}][\langle\mathsf{mk}\,\mathsf{q}\rangle^{+}][\langle\mathsf{q}\rangle] & =(4) \\
t[\mathsf{p}^{+}][\mathsf{p}^{+}][\langle\mathsf{mk}\,\mathsf{q}\rangle^{+}][\langle\mathsf{q}\rangle] & =(5) \\
t[\mathsf{p}^{+}][\langle\mathsf{q}\rangle] & =(7) \\
t &
\end{array}
$$

C Listing for Section 4

For reference, we list the rules for the minimised single substitution syntax (Definition 7).

$$
\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])
\end{array}
$$

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

D Listings for Section 5

As an example of the equations in the termification model construction (Problem 10), we derive one of the functor laws for types:

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

We show that the roundtrip CwF $\longrightarrow$ SSC $\xrightarrow{\text{termification}}$ CwF results in a contextually isomorphic CwF. A contextual isomorphism [18] is a weak CwF-morphism (pseudomorphism, i.e. context extension and the empty context are only preserved up to isomorphism) which is bijective on types and terms. Note that a contextual isomorphism preserves all type formers specified by universal properties.

Problem 11. *Given a CwF (with type formers) M, let M' denote the CwF obtained by first seeing it as an SSC model and then termifying it. We construct a contextual isomorphism between M' and M.*

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

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

We define them in the same order, omitting the M subscripts: $F\,\Gamma := \diamond \triangleright \Gamma$, $F\,\gamma := (\mathsf{p}, \gamma[\mathsf{p}] \cdot \mathsf{q})$, $F\,A := \mathsf{El}\,(A[\mathsf{p}] \cdot \mathsf{q})$, $F\,a := a[\mathsf{p}] \cdot \mathsf{q}$. It is easy to check that F preserves the CwF structure, i.e. it is a functor, $\varepsilon : \mathsf{Sub}\,(F \diamond) \diamond$ 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 round-trip, 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 (a Sub is either a lifted weakening or a lifted single substitution, but we don't know anything about the relationship between Δ and Γ). However, the set of terms and types are still isomorphic after the round-trip. We leave formulating the right notion of contextual isomorphism for SSCs as future work.