Admissibility of Substitution for Multimode Type Theory

Joris Ceulemans
DistriNet, KU Leuven, Belgium

Andreas Nuyts
DistriNet, KU Leuven, Belgium

Dominique Devriese
DistriNet, KU Leuven, Belgium

Abstract

Multimode Type theory (MTT) is a generic type theory that can be instantiated with an arbitrary mode theory to model features like parametricity, cohesion and guarded recursion. However, the presence of modalities in MTT significantly complicates the substitution calculus of this system. Moreover, MTT's syntax has explicit substitutions with an axiomatic system – not an algorithm – governing the connection between an explicitly substituted term and the resulting term in which variables have actually been replaced. So far, admissibility of substitution for MTT has only been proved as a consequence of normalisation via normalisation by evaluation. In this paper, we present a proof of admissibility of substitution for MTT that is completely separated from normalisation. To this end, we introduce Substitution-Free Multimode Type Theory (SFMTT): a formulation of MTT without explicit substitutions, but for which we are able to give a structurally recursive substitution algorithm, suitable for implementation in a total programming language or proof assistant. On the usual formulation of MTT, we consider *σ-equality*, the congruence generated solely by equality rules for explicit substitutions. There is a trivial embedding from SFMTT to MTT, and a converse translation that eliminates the explicit substitutions. We prove soundness and completeness with respect to σ-equivalence and thus establish that MTT with σ-equality has computable σ-normal forms, given by the terms of SFMTT.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Modal and temporal logics; Software and its engineering → Syntax

Keywords and phrases dependent type theory, modalities, multimode type theory, explicit substitutions, admissibility of substitution

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Substitution is the operation that replaces variables in a term with other terms. It is a key part in defining the semantics of many programming languages. In a dependent type system, it is even necessary in order to formulate the typing rules, such as the one for dependent function application. However, defining substitution is not as simple as it intuitively may seem.

1.1 Renaming and Substitution in the Simply Typed Lambda Calculus

For example, consider the well-known simply typed lambda calculus. We call $\mathsf{Tm}^{\text{STLC}}(\Gamma \vdash T)$ the set of terms of type T with free variables in context Γ and $\mathsf{Sub}^{\text{STLC}}(\Gamma \to \Delta)$ the set of well-formed (simultaneous) substitutions from Γ to Δ. These substitutions are lists of terms: they contain a term of type T in context Γ for every variable of type T in context Δ. In other words, STLC substitutions are constructed in two ways: $!_\Gamma : \mathsf{Sub}^{\text{STLC}}(\Gamma \to \cdot)$ representing

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the empty list and $\sigma.t : \mathsf{Sub}^{\text{STLC}}(\Gamma \to \Delta, x : T)$ which substitutes variables in Δ according to $\sigma : \mathsf{Sub}^{\text{STLC}}(\Gamma \to \Delta)$ and substitutes $t : \mathsf{Tm}^{\text{STLC}}(\Gamma \vdash T)$ for the variable $x : T$.

Applying a substitution $\sigma : \mathsf{Sub}^{\text{STLC}}(\Gamma \to \Delta)$ to a term $t : \mathsf{Tm}^{\text{STLC}}(\Delta \vdash T)$ should produce a term $t\,[\,\sigma\,] : \mathsf{Tm}^{\text{STLC}}(\Gamma \vdash T)$. This can be defined via recursion on the term t. Some cases are very simple: for variables x the corresponding term is found in σ and for applications $(f\,s)[\,\sigma\,]$ we recurse on the subterms $(f\,[\,\sigma\,])(s\,[\,\sigma\,])$. However, difficulty arises when binders are involved. For lambda terms $(\lambda x.s) : \mathsf{Tm}^{\text{STLC}}(\Delta \vdash T \to S)$ with $s : \mathsf{Tm}^{\text{STLC}}(\Delta, x : T \vdash S)$, the substitution $(\lambda x.s)[\,\sigma\,]$ is defined as $\lambda x.(s\,[\,\sigma^+\,])$ for $\sigma^+ : \mathsf{Sub}^{\text{STLC}}((\Gamma, x : T) \to (\Delta, x : T))$ a version of σ that is lifted to the contexts extended with x. To construct σ^+, we can use the extension constructor above with variable x as term, but then we still need to weaken the terms in σ from context Γ to the extended context $\Gamma, x : T$. A naive definition might implement this weakening of terms $t : \mathsf{Tm}^{\text{STLC}}(\Gamma \vdash A)$ to $\mathsf{Tm}^{\text{STLC}}(\Gamma, x : B \vdash A)$ by applying a substitution from $\Gamma, x : B$ to Γ, but this makes the story cyclic.

An elegant solution to avoid this cycle, proposed and advocated by McBride [21] and Allais et al. [4], is to separately consider renamings and substitutions. Whereas a substitution maps variables to terms, a renaming from Γ to Δ maps every variable in Δ to a variable in Γ of the same type. Weakening, in particular, is a renaming. Thus, the terms listed in a substitution can be weakened by applying a weakening *renaming*, and the variables listed in a renaming – represented as De Bruijn indices – can be weakened by incrementation. So we can break the cycle by defining first how to rename and then how to substitute a term, each time by induction on the term. Going further, code duplication between the two term traversals can be avoided with a shared generic implementation [21, 4].

1.2 Multimode Type Theory

This paper is concerned with substitution in modal type theory, more specifically in the system MTT (Multimode Type Theory[1]) by Gratzer et al. [18]. MTT is a type theory that can be instantiated with a mode theory that specifies, among others, a collection of modes and modalities. Modes m index typing judgements and qualify their meaning: judgements in one mode may represent, for example, regular values, while judgements in other modes may represent time-indexed values or pairs of values satisfying a certain relation [11]. Modalities $\mu : m_1 \to m_2$ represent ways to transport terms and types from mode m_1 to mode m_2. We postpone a more extensive introduction to MTT to Section 2, but we will already explain why substitution in modal type theory is significantly more complicated.

First, modes and modalities complicate the context structure in MTT. For every modality μ, MTT has a new primitive context operation $\_\,.\,\text{🔒}_\mu$ which also extends to substitutions: if $\sigma : \mathsf{Sub}^{\text{MTT}}(\Gamma \to \Delta)$, then we get a new substitution $\sigma\,.\,\text{🔒}_\mu : \mathsf{Sub}^{\text{MTT}}(\Gamma\,.\,\text{🔒}_\mu \to \Delta\,.\,\text{🔒}_\mu)$.[2] Furthermore, all variables in a context are annotated with a modality. This also impacts how substitutions are defined: to produce a substitution from Γ to $\Delta, \mu \mid x : T$ (i.e. Δ extended with a variable x of type T annotated with modality μ), we need to provide a $\sigma : \mathsf{Sub}^{\text{MTT}}(\Gamma \to \Delta)$ and a term $t : \mathsf{Tm}^{\text{MTT}}(\Gamma\,.\,\text{🔒}_\mu \vdash T)$ in a locked context. In other words, MTT substitutions are not mere lists of terms and applying substitutions to variables is not just a lookup operation.

Complicating things further, mode theories can define two-cells $\alpha \in \mu \Rightarrow \rho$ between modalities μ and ρ. For every two-cell $\alpha \in \mu \Rightarrow \rho$ from μ to ρ and every context Γ we get a

---

[1] The names Multimode and Multimodal Type Theory are used interchangeably for the same system MTT which supports both multiple modes and multiple modalities.

[2] The operation $\_\,.\,\text{🔒}_\mu$ can be seen as some sort of left adjoint to μ. See Section 2.1 for more details.

new primitive *key substitution* $\text{🔑}^{\alpha}_{\Gamma}$ from $\Gamma \,.\, \text{🔒}_{\rho}$ to $\Gamma \,.\, \text{🔒}_{\mu}$ and we have to specify how these act on variables and terms.

Finally, STLC substitutions $\sigma : \mathsf{Sub}^{\text{STLC}}(\Gamma \to \Delta)$ and $\tau : \mathsf{Sub}^{\text{STLC}}(\Delta \to \Xi)$ can be composed to a $\mathsf{Sub}^{\text{STLC}}(\Gamma \to \Xi)$ by applying σ to every term in τ. Applying this composed substitution is equivalent to applying τ and σ consecutively. However, with the additional primitive substitutions in MTT, we cannot compute such a composed substitution anymore (we refer to Example 2 for more details). For that reason, MTT includes a primitive constructor $\tau \circ \sigma$ for substitution composition, and we want to define $t\,[\,\tau \circ \sigma\,]$ as $(t\,[\,\tau\,])[\,\sigma\,]$. However, substitution of a term is defined by traversing the term and applying the substitution to every variable. But for a variable x, the substitution $x\,[\,\tau\,]$ is again an arbitrary term so that $(x\,[\,\tau\,])[\,\sigma\,]$ may trigger another arbitrary term traversal. Thus, this naïve definition of $t\,[\,\tau \circ \sigma\,]$ is not structurally recursive,[3] and restructuring the substitution algorithm to restore structural recursion is one of the main contributions of the current paper (Section 3).

1.3 Contributions and Overview

In this paper, we define substitution for MTT, resolving the above problems by identifying the equivalent of renamings and substitutions in MTT and building a structurally recursive substitution algorithm in terms of them. Specifically, we contribute the following.

- We define WSMTT: an intrinsically and modally scoped untyped syntax for MTT. Moreover, we define σ-equivalence for WSMTT: the congruence relation generated by substitution-related equality rules, but not β- and η-rules.
- We define SFMTT: a variant of WSMTT without explicit substitutions in terms or types. Moreover, we define a notion of SFMTT renamings and substitutions and implement a structurally recursive algorithm to apply those to types and terms.
- We provide a translation from WSMTT to SFMTT, which translates every WSMTT term and type to an expression without substitutions.
- We prove the soundness and completeness of our algorithm. Soundness means that WSMTT terms map to substitution-free terms that are σ-equivalent to the original. Completeness states that σ-equivalent WSMTT terms map to equal translations. Both results combined show that SFMTT terms are the σ-normal forms of WSMTT terms.

Section 2 will provide the necessary background and details about the multimode type theory MTT. We continue in Section 3 to describe the SFMTT syntax and the algorithm for applying renamings and substitutions in that setting. The translation from MTT to SFMTT is also discussed there. Sections 4 and 5 then cover the soundness and completeness proofs, respectively. We conclude in Section 6 with related and future work. A technical report accompanying this paper contains all details of the soundness and completeness proofs.[4]

2 Multimode Type Theory (MTT)

In this section we introduce the type system MTT as developed by Gratzer et al. [18]. We start in Section 2.1 with the necessary background and continue in Section 2.2 with our own presentation of MTT that we call WSMTT, including a discussion of the differences with

[3] One can argue that in both recursive applications the substitution gets structurally smaller and that therefore we do have structural recursion. However, substitutions do get bigger in other recursive calls, for example by lifting when they are pushed under a binder.

[4] Available at `https://people.cs.kuleuven.be/~joris.ceulemans/mtt-sub-tech-report.pdf`.

$$\textsc{ctx-empty}\ \frac{}{\cdot\ \mathsf{ctx} \mathbin{@} m} \qquad \textsc{ctx-lock}\ \frac{\Gamma\ \mathsf{ctx} \mathbin{@} n \qquad \mu : m \to n}{\Gamma\,.\,\text{🔒}_\mu\ \mathsf{sctx} \mathbin{@} m} \qquad \textsc{ctx-extend}\ \frac{\Gamma\ \mathsf{ctx} \mathbin{@} m \qquad \mu : n \to m \qquad \Gamma\,.\,\text{🔒}_\mu \vdash T\ \mathsf{ty} \mathbin{@} n}{\Gamma\,.\,(\mu \mid x : T)\ \mathsf{ctx} \mathbin{@} m}$$

$$\mathsf{locks}\,(\cdot) = \mathbb{1} \qquad \mathsf{locks}\,(\Gamma\,.\,\text{🔒}_\mu) = \mathsf{locks}\,(\Gamma) \circ \mu \qquad \mathsf{locks}\,(\Gamma\,.\,(\mu \mid x : T)) = \mathsf{locks}\,(\Gamma)$$

$$\textsc{tm-var}\ \frac{\alpha \in \mu \Rightarrow \mathsf{locks}\,(\Delta)}{\Gamma\,.\,(\mu \mid x : T)\,.\,\Delta \vdash x^\alpha : T^\alpha \mathbin{@} m} \qquad \textsc{ty-mod}\ \frac{\Gamma\,.\,\text{🔒}_\mu \vdash T\ \mathsf{ty} \mathbin{@} n}{\Gamma \vdash \langle \mu \mid T \rangle\ \mathsf{ty} \mathbin{@} m} \qquad \textsc{tm-mod}\ \frac{\Gamma\,.\,\text{🔒}_\mu \vdash t : T \mathbin{@} n}{\Gamma \vdash \mathsf{mod}_\mu\,(t) : \langle \mu \mid T \rangle \mathbin{@} m}$$

$$\textsc{ty-arrow}\ \frac{\Gamma\,.\,\text{🔒}_\mu \vdash T\ \mathsf{ty} \mathbin{@} n \qquad \Gamma\,.\,(\mu \mid x : T) \vdash S\ \mathsf{ty} \mathbin{@} m}{\Gamma \vdash (\mu \mid T) \to S\ \mathsf{ty} \mathbin{@} m} \qquad \textsc{tm-lam}\ \frac{\Gamma\,.\,(\mu \mid x : T) \vdash s : S \mathbin{@} m}{\Gamma \vdash \lambda(\mu \mid x).s : (\mu \mid T) \to S \mathbin{@} m}$$

$$\textsc{tm-app}\ \frac{\Gamma \vdash f : (\mu \mid T) \to S \mathbin{@} m \qquad \Gamma\,.\,\text{🔒}_\mu \vdash t : T \mathbin{@} n}{\Gamma \vdash \mathsf{app}_\mu\,(f; t) : S\ [\ \mathsf{id}.t\ ] \mathbin{@} m}$$

Figure 1 Selection of MTT inference rules

the original formulation. In this section we also discuss (WS)MTT's substitution calculus. Section 2.3 concludes with a discussion on an equivalence relation on terms and substitutions called σ-equivalence.

2.1 Background on the MTT Type System

MTT can be seen as a *framework* for modal type theory: it is parametrised by a mode theory which specifies the modalities and how they interact. More concretely, a mode theory in MTT is a strict 2-category of which the 0-cells (objects) are called modes and the 1-cells (morphisms) are called modalities. This already makes it clear that we have a unit modality $\mathbb{1}$ for every mode and that compatible modalities can be composed. Moreover, we also have a notion of 2-cells between modalities, which will be denoted as $\alpha \in \mu \Rightarrow \nu$ for a 2-cell α from μ to ν. Such 2-cells can be composed vertically (which we write as $\beta \circ \alpha$) and horizontally (written as $\beta \star \alpha$). For every modality $\mu : m \to n$ there is a unit 2-cell $1_\mu \in \mu \Rightarrow \mu$.

In MTT, every judgment (so every context, type and term) lives at a particular mode of the mode theory. This is made clear by adding $@\, m$ to a judgment at mode m. We can think of every mode as containing a copy of Martin-Löf Type Theory (MLTT [20]) with natural numbers, products, etc. As they are confined to a single mode and do not really interact with modalities, we will not discuss these rules in the paper (as an illustration we do include a type of Booleans in the technical report though). The connection between the different modes is made via the modalities, as explained in the following paragraphs.

A selection of the rules for constructing contexts, types and terms in MTT can be found in Figure 1. Contexts consist of variables (ctx-extend), each annotated with a modality, and locks (ctx-lock), which play an important role in determining when a variable can be used to construct a term. Note that a lock goes in the opposite direction of its modality: the lock operation for a modality $\mu : m \to n$ takes a context from mode n to mode m.

A variable can be used as a term whenever there is a two-cell from its annotation to the composition of all locks to the right of that variable (tm-var). Every modality μ gives rise to a modal type former $\langle \mu \mid \_ \rangle$ which can be seen as a (weak) dependent right adjoint [10] to $\_\,.\,\text{🔒}_\mu$ (ty-mod). One direction of transposition for this dependent adjunction is given by tm-mod. We do not discuss the MTT elimination principle for modal types here.

$$\textsc{sctx-empty}\ \frac{}{\cdot\ \mathsf{sctx} \mathbin{@} m} \qquad \textsc{sctx-lock}\ \frac{\hat{\Gamma}\ \mathsf{sctx} \mathbin{@} n \qquad \mu : m \to n}{\hat{\Gamma}\,.\,\text{🔒}_{\mu}\ \mathsf{sctx} \mathbin{@} m} \qquad \textsc{sctx-extend}\ \frac{\hat{\Gamma}\ \mathsf{sctx} \mathbin{@} m \qquad \mu : n \to m}{\hat{\Gamma}\,.\,\mu\ \mathsf{sctx} \mathbin{@} m}$$

$$\textsc{locktele-empty}\ \frac{}{\cdot : \mathsf{LockTele}(m \to m)} \qquad \textsc{locktele-lock}\ \frac{\Lambda : \mathsf{LockTele}(o \to n) \qquad \mu : m \to n}{\Lambda\,.\,\text{🔒}_{\mu} : \mathsf{LockTele}(o \to m)}$$

$$\mathsf{locks}\,(\cdot) = \mathbb{1} \qquad \mathsf{locks}\,(\Lambda\,.\,\text{🔒}_{\mu}) = \mathsf{locks}\,(\Lambda) \circ \mu$$

Figure 2 Definition of scoping contexts and lock telescopes

Finally, we can also consider modal function types (TY-ARROW). Their values can be constructed via lambda abstraction (TM-LAM), which adds an annotated variable to the context. Eliminating functions is done via application (TM-APP) where the argument should type check in a locked context. Note that we are using a substitution in this rule to accommodate for dependent types, but we postpone the discussion about substitution in MTT to Section 2.2.1.

▶ **Example 1.** To illustrate MTT, we will look at an example program. Suppose that we have a mode theory with modalities $\mu : m \to n$ and $\kappa : n \to m$, and a 2-cell $\alpha \in \mathbb{1} \Rightarrow \mu \circ \kappa$. Then we can construct a function of type $(\mathbb{1} \mid A) \to \langle \mu \mid (\mathbb{1} \mid B) \to \langle \kappa \mid A^{\alpha} \rangle \rangle$ as follows: $\lambda(\mathbb{1} \mid x).\mathsf{mod}_{\mu}\,(\lambda(\mathbb{1} \mid y).\mathsf{mod}_{\kappa}\,(x^{\alpha}))$. We leave it to the reader to verify that this is indeed a well-typed program according to the rules in Figure 1.

2.2 Alternative Presentation: Extrinsically Typed, Intrinsically Scoped

The way the MTT syntax is presented in the previous section, which is also how it is originally presented in [18], could be called *intrinsically typed*. This means that we see the typing rules from Figure 1 as the way types and terms are introduced. In other words, we cannot even talk about ill-typed terms or ill-formed types.

For the purposes of this paper, it will be more useful to work with extrinsically typed (one could say raw) syntax. In that way, our substitution algorithm can work on pure syntax without having to take typing derivations into account. Moreover, substitution is necessary to formulate some typing rules (such as TM-APP). In MTT, this does not lead to circularity thanks to the use of explicit substitutions (see further) but it would make a substitution algorithm problematically cyclic if it works with intrinsically typed syntax.

However, in order to conveniently develop our substitution algorithm, we will use *intrinsically scoped* syntax, defined in this section. In order to distinguish between our system and the original presentation of MTT, we call the intrinsically scoped syntax WSMTT (for well-scoped MTT). Apart from the change from an intrinsically-typed to an extrinsically-typed presentation, this reformulation does not modify the MTT type theory. Specifically, it does not modify MTT's treatment of substitution; that will only happen in Section 3, in a different system called SFMTT.

For defining the intrinsically scoped syntax, we introduce scoping contexts in Figure 2. They are essentially MTT contexts from Figure 1 where all type information has been removed. We note that in the rule SCTX-EXTEND only the modality annotation of a variable is added to a scoping context. Indeed, in the rest of the paper we will not use named variables but a form of De Bruijn indices. This allows us to ignore α-equivalence and variable capture when implementing substitution.

The WSMTT syntax is now introduced via a judgment $\hat{\Gamma} \vdash_{\mathsf{ws}} t\ \mathsf{expr} \mathbin{@} m$, meaning that t is a WSMTT expression in scoping context $\hat{\Gamma}$ at mode m. Note that since we are not

$$\frac{\hat{\Gamma} \mathbin{.} \text{🔒}_\mu \vdash_{\mathsf{ws}} T \ \mathsf{expr} \ @\, n \qquad \hat{\Gamma} \mathbin{.} \mu \vdash_{\mathsf{ws}} S \ \mathsf{expr} \ @\, m}{\hat{\Gamma} \vdash_{\mathsf{ws}} (\mu \mid T) \to S \ \mathsf{expr} \ @\, m}\ \textsc{wsmtt-expr-arrow} \qquad \frac{\hat{\Gamma} \mathbin{.} \mu \vdash_{\mathsf{ws}} t \ \mathsf{expr} \ @\, m}{\hat{\Gamma} \vdash_{\mathsf{ws}} \lambda^{\mu}\,(t) \ \mathsf{expr} \ @\, m}\ \textsc{wsmtt-expr-lam}$$

$$\frac{\hat{\Gamma} \ \mathsf{sctx} \ @\, n \qquad \mu : m \to n}{\hat{\Gamma} \mathbin{.} \mu \mathbin{.} \text{🔒}_\mu \vdash_{\mathsf{ws}} \mathbf{v}_0 \ \mathsf{expr} \ @\, m}\ \textsc{wsmtt-expr-var} \qquad \frac{\hat{\Delta} \vdash_{\mathsf{ws}} t \ \mathsf{expr} \ @\, m \qquad \vdash_{\mathsf{ws}} \sigma \ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \ @\, m}{\hat{\Gamma} \vdash_{\mathsf{ws}} t \ [\, \sigma \,]_{\mathsf{ws}} \ \mathsf{expr} \ @\, m}\ \textsc{wsmtt-expr-sub}$$

$$\frac{}{\vdash_{\mathsf{ws}} \ ! \ \mathsf{sub}(\hat{\Gamma} \to \cdot) \ @\, m}\ \textsc{wsmtt-sub-empty} \qquad \frac{\hat{\Gamma} \ \mathsf{sctx} \ @\, m}{\vdash_{\mathsf{ws}} \mathsf{id} \ \mathsf{sub}(\hat{\Gamma} \to \hat{\Gamma}) \ @\, m}\ \textsc{wsmtt-sub-id} \qquad \frac{\hat{\Gamma} \ \mathsf{sctx} \ @\, m}{\vdash_{\mathsf{ws}} \pi \ \mathsf{sub}(\hat{\Gamma} \mathbin{.} \mu \to \hat{\Gamma}) \ @\, m}\ \textsc{wsmtt-sub-weaken}$$

$$\frac{\vdash_{\mathsf{ws}} \sigma \ \mathsf{sub}(\hat{\Delta} \to \hat{\Xi}) \ @\, m \qquad \vdash_{\mathsf{ws}} \tau \ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \ @\, m}{\vdash_{\mathsf{ws}} \sigma \circ \tau \ \mathsf{sub}(\hat{\Gamma} \to \hat{\Xi}) \ @\, m}\ \textsc{wsmtt-sub-compose} \qquad \frac{\vdash_{\mathsf{ws}} \sigma \ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \ @\, n \qquad \mu : m \to n}{\vdash_{\mathsf{ws}} \sigma \mathbin{.} \text{🔒}_\mu \ \mathsf{sub}(\hat{\Gamma} \mathbin{.} \text{🔒}_\mu \to \hat{\Delta} \mathbin{.} \text{🔒}_\mu) \ @\, m}\ \textsc{wsmtt-sub-lock}$$

$$\frac{\Theta, \Psi : \mathsf{LockTele}(n \to m) \qquad \alpha \in \mathsf{locks}(\Theta) \Rightarrow \mathsf{locks}(\Psi)}{\vdash_{\mathsf{ws}} \text{🔑}^{\alpha \in \Theta \Rightarrow \Psi}_{\hat{\Gamma}} \ \mathsf{sub}(\hat{\Gamma} \mathbin{.} \Psi \to \hat{\Gamma} \mathbin{.} \Theta) \ @\, m}\ \textsc{wsmtt-sub-key}$$

$$\frac{\vdash_{\mathsf{ws}} \sigma \ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \ @\, m \qquad \hat{\Gamma} \mathbin{.} \text{🔒}_\mu \vdash_{\mathsf{ws}} t \ \mathsf{expr} \ @\, n}{\vdash_{\mathsf{ws}} \sigma.t \ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta} \mathbin{.} \mu) \ @\, m}\ \textsc{wsmtt-sub-extend}$$

Figure 3 Definition of raw WSMTT expressions and substitutions

specifying typing rules, the distinction between types and terms has disappeared and we talk about WSMTT *expressions*. An example of two rules that introduce WSMTT syntax can be found in the top row of Figure 3. In order to construct a modal function type in scoping context $\hat{\Gamma}$, we need a domain type in the locked scoping context $\hat{\Gamma} \mathbin{.} \text{🔒}_\mu$ and a codomain type where we extend the scoping context with a variable annotated with μ (WSMTT-EXPR-ARROW). The rule for introducing lambda abstraction is similar (WSMTT-EXPR-LAM). Note that we can obtain all these rules by removing the typing information from the typing rules in Figure 1.

The WSMTT variable rule WSMTT-EXPR-VAR has changed somewhat with respect to Figure 1: it only allows us to access the last variable added to a scoping context and only if it is locked behind the same modality as its annotation. It is standard, in formulations of type theory with explicit substitutions [1], to only allow access to the last variable which has De Bruijn index zero, since the De Bruijn index can then be incremented using a weakening substitution (WSMTT-SUB-WEAKEN) which applies not only to variables but to any objects-in-context. In the technical report on MTT [17], this standard practice is adapted to MTT with a variable rule that is a typed version of WSMTT-EXPR-VAR. The general variable rule TM-VAR (or its intrinsically scoped counterpart) however remains derivable by substituting $\mathbf{v}_0$ with substitutions constructed via π, 🔑^{α} (WSMTT-SUB-KEY) and $\_ \mathbin{.} \text{🔒}_\mu$ (WSMTT-SUB-LOCK).

2.2.1 Substitution Calculus

In both [18, 17] and our presentation, MTT is a system with explicit substitution: applying a substitution to an expression is viewed as a syntax constructor (WSMTT-EXPR-SUB). This also means that expressions are defined mutually inductively with substitutions. For the latter, we introduce a judgment form $\vdash_{\mathsf{ws}} \sigma \ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \ @\, m$ expressing that σ is a substitution from scoping context $\hat{\Gamma}$ to $\hat{\Delta}$ at mode m.

Figure 3 shows all WSMTT substitution constructors. There is a unique substitution to the empty context (WSMTT-SUB-EMPTY) and identity (WSMTT-SUB-ID) and weakening (WSMTT-SUB-WEAKEN) substitutions. We can compose substitutions (WSMTT-SUB-COMPOSE, note that this is a constructor), lock them (WSMTT-SUB-LOCK) and extend them with a term to extend the codomain with a new variable (WSMTT-SUB-EXTEND). Note that this term has to live in a locked scoping context. Finally, every 2-cell in the mode theory gives rise to a key substitution (WSMTT-SUB-KEY). This last rule introduces the concept of lock telescopes: sequences of zero or more locks that have the right domain and codomain modes to be composed. A lock telescope $\Theta : \mathsf{LockTele}(n \to m)$ can be applied to a scoping context at mode n to obtain a scoping context at mode m. We can also compose all modalities in Θ to obtain a modality $\mathsf{locks}\,(\Theta) : m \to n$. Precise definitions are given in Figure 2.

▶ **Example 2** (Non-admissibility of composition). Figure 3 contains a *constructor* for the composition of substitutions, which breaks structural recursion in the usual argument of admissibility of substitution (Section 1.2). Here we argue that this is necessary: composition of substitutions is not admissible. Suppose that we have a mode theory with a 2-cell $\alpha \in \mu \circ \nu \Rightarrow \rho$. Then we can consider the key substitution $\vdash_{\mathsf{ws}} \text{🔑}^{\alpha \in \mu \circ \nu \Rightarrow \rho}_{\hat{\Gamma}}\ \mathsf{sub}(\hat{\Gamma} \,.\, \text{🔒}_\rho \to \hat{\Gamma} \,.\, \text{🔒}_\mu \,.\, \text{🔒}_\nu) \mathbin{@} m$. Furthermore, given an expression t in $\hat{\Gamma} \,.\, \text{🔒}_\mu \,.\, \text{🔒}_{\mathbb{1}}$ we can construct $\vdash_{\mathsf{ws}} (\mathsf{id}.t) \,.\, \text{🔒}_\nu\ \mathsf{sub}(\hat{\Gamma} \,.\, \text{🔒}_\mu \,.\, \text{🔒}_\nu \to \hat{\Gamma} \,.\, \text{🔒}_\mu \,.\, \mathbb{1} \,.\, \text{🔒}_\nu) \mathbin{@} m$. The composite of these two is a substitution from $\hat{\Gamma} \,.\, \text{🔒}_\rho$ to $\hat{\Gamma} \,.\, \text{🔒}_\mu \,.\, \mathbb{1} \,.\, \text{🔒}_\nu$, which both splits ρ into $\mu \circ \nu$ and extends the codomain with a variable. In other words, if we would like composition to be admissible, the rule WSMTT-SUB-EXTEND would have to take 2-cells into account. A somewhat dual counterexample can be constructed in a mode theory with a 2-cell $\alpha \in \rho \Rightarrow \mu \circ \nu$. Now we can consider the substitutions $\vdash_{\mathsf{ws}} \pi \,.\, \text{🔒}_\nu\ \mathsf{sub}(\hat{\Gamma} \,.\, \text{🔒}_\mu \,.\, \mathbb{1} \,.\, \text{🔒}_\nu \to \hat{\Gamma} \,.\, \text{🔒}_\mu \,.\, \text{🔒}_\nu) \mathbin{@} m$ and $\vdash_{\mathsf{ws}} \text{🔑}^{\alpha \in \text{🔒}_\rho \Rightarrow \text{🔒}_\mu \,.\, \text{🔒}_\nu}_{\hat{\Gamma}}\ \mathsf{sub}(\hat{\Gamma} \,.\, \text{🔒}_\mu \,.\, \text{🔒}_\nu \to \hat{\Gamma} \,.\, \text{🔒}_\rho) \mathbin{@} m$. Their composite cannot be constructed from the other constructors unless we make the rule WSMTT-SUB-WEAKEN take 2-cells into account. This could quickly get out of hand when the involved 2-cells have a composite of more than 2 modalities in both their domain and codomain. Moreover, it would severely clutter the treatment of σ-equivalence of substitutions as discussed in Section 2.3.

2.2.2 Lock Telescopes vs. Strict Functoriality of Locks

SCTX-LOCK-ID
$$\frac{\hat{\Gamma}\ \mathsf{sctx} \mathbin{@} m}{\hat{\Gamma} \,.\, \text{🔒}_{\mathbb{1}} = \hat{\Gamma}\ \mathsf{sctx} \mathbin{@} m}$$

SCTX-LOCK-COMP
$$\frac{\hat{\Gamma}\ \mathsf{sctx} \mathbin{@} o \qquad \mu : m \to n \qquad \nu : n \to o}{\hat{\Gamma} \,.\, \text{🔒}_{\nu \circ \mu} = \hat{\Gamma} \,.\, \text{🔒}_\nu \,.\, \text{🔒}_\mu\ \mathsf{sctx} \mathbin{@} m}$$

Figure 4 Strict functoriality of the lock operation on scoping contexts (optional)

The original presentation of MTT [18, 17] makes no mention of lock telescopes. Instead, it features strict functoriality rules for the lock operation on contexts, of which we give counterparts for scoping contexts in Figure 4. A consequence of these rules is that any lock telescope can be fused into a single lock.

It is however quite unusual to have a non-trivial equational theory on contexts and early explorations of a *lock calculus* for MTT [23] suggest that it may be advantageous to drop the functoriality rules; by WSMTT-SUB-KEY for the identity 2-cell, they automatically hold up to isomorphism. During the development of the current paper, we had a formulation of MTT in mind *without* these functoriality rules. However, nowhere in our proofs do we case distinguish on the number of locks in a given part of the context, or read off the modality annotation of a specific lock, so our results remain valid when we extend raw WSMTT with the rules in Figure 4.

$$\frac{\hat{\Xi} \vdash_{\mathsf{ws}} t\ \mathsf{expr} \,@\, m \qquad \vdash_{\mathsf{ws}} \sigma\ \mathsf{sub}(\hat{\Delta} \to \hat{\Xi}) \,@\, m \qquad \vdash_{\mathsf{ws}} \tau\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \,@\, m}{\hat{\Gamma} \vdash_{\mathsf{ws}} t\ [\,\sigma \circ \tau\,]_{\mathsf{ws}} =^{\sigma} t\ [\,\sigma\,]_{\mathsf{ws}}\ [\,\tau\,]_{\mathsf{ws}}\ \ \mathsf{expr} \,@\, m}$$

$$\frac{\hat{\Delta} \vdash_{\mathsf{ws}} t =^{\sigma} s\ \mathsf{expr} \,@\, m \qquad \vdash_{\mathsf{ws}} \tau =^{\sigma} \sigma\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \,@\, m}{\hat{\Gamma} \vdash_{\mathsf{ws}} t\ [\,\tau\,]_{\mathsf{ws}} =^{\sigma} s\ [\,\sigma\,]_{\mathsf{ws}}\ \ \mathsf{expr} \,@\, m}$$

$$\frac{\hat{\Delta} \,.\, \mu \vdash_{\mathsf{ws}} t\ \mathsf{expr} \,@\, m \qquad \vdash_{\mathsf{ws}} \sigma\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \,@\, m}{\hat{\Gamma} \vdash_{\mathsf{ws}} (\lambda^{\mu}\,(t))\ [\,\sigma\,]_{\mathsf{ws}} =^{\sigma} \lambda^{\mu}\left(t\ \left[\,\sigma^{+}\,\right]_{\mathsf{ws}}\right)\ \mathsf{expr} \,@\, m} \qquad \text{with } \sigma^{+} = (\sigma \circ \pi).\mathbf{v}_0$$

$$\frac{\vdash_{\mathsf{ws}} \sigma\ \mathsf{sub}(\hat{\Delta} \to \hat{\Xi}) \,@\, m \qquad \vdash_{\mathsf{ws}} \tau\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \,@\, m}{\vdash_{\mathsf{ws}} (\sigma \circ \tau) \,.\, \text{🔒}_{\mu} =^{\sigma} (\sigma \,.\, \text{🔒}_{\mu}) \circ (\tau \,.\, \text{🔒}_{\mu})\ \mathsf{sub}(\hat{\Gamma} \,.\, \text{🔒}_{\mu} \to \hat{\Xi} \,.\, \text{🔒}_{\mu}) \,@\, n}$$

$$\frac{\hat{\Gamma}\ \mathsf{sctx} \,@\, n \qquad \Lambda : \mathsf{LockTele}(n \to m)}{\vdash_{\mathsf{ws}} \text{🔑}_{\hat{\Gamma}}^{1_{\mathsf{locks}(\Lambda)} \in \Lambda \Rightarrow \Lambda} =^{\sigma} \mathsf{id}\ \mathsf{sub}(\hat{\Gamma} \,.\, \Lambda \to \hat{\Gamma} \,.\, \Lambda) \,@\, m}$$

$$\frac{\alpha \in \mathsf{locks}\,(\Lambda) \Rightarrow \mathsf{locks}\,(\Theta) \qquad \beta \in \mathsf{locks}\,(\Theta) \Rightarrow \mathsf{locks}\,(\Psi)}{\vdash_{\mathsf{ws}} \text{🔑}_{\hat{\Gamma}}^{\beta \circ \alpha \in \Lambda \Rightarrow \Psi} =^{\sigma} \text{🔑}_{\hat{\Gamma}}^{\alpha \in \Lambda \Rightarrow \Theta} \circ \text{🔑}_{\hat{\Gamma}}^{\beta \in \Theta \Rightarrow \Psi}\ \mathsf{sub}(\hat{\Gamma} \,.\, \Psi \to \hat{\Gamma} \,.\, \Lambda) \,@\, m}$$

$$\frac{\alpha \in \mathsf{locks}\,(\Lambda) \Rightarrow \mathsf{locks}\,(\Theta) \qquad \vdash_{\mathsf{ws}} \sigma\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \,@\, n}{\vdash_{\mathsf{ws}} \text{🔑}_{\hat{\Delta}}^{\alpha \in \Lambda \Rightarrow \Theta} \circ (\sigma \,.\, \Theta) =^{\sigma} (\sigma \,.\, \Lambda) \circ \text{🔑}_{\hat{\Gamma}}^{\alpha \in \Lambda \Rightarrow \Theta}\ \mathsf{sub}(\hat{\Gamma} \,.\, \Theta \to \hat{\Delta} \,.\, \Lambda) \,@\, m}$$

$$\frac{\alpha \in \mathsf{locks}\,(\Lambda_1) \Rightarrow \mathsf{locks}\,(\Lambda_2) \qquad \beta \in \mathsf{locks}\,(\Theta_1) \Rightarrow \mathsf{locks}\,(\Theta_2)}{\vdash_{\mathsf{ws}} \text{🔑}_{\hat{\Gamma}}^{\alpha \star \beta \in \Lambda_1 \,.\, \Theta_1 \Rightarrow \Lambda_2 \,.\, \Theta_2} =^{\sigma} (\text{🔑}_{\hat{\Gamma}}^{\alpha \in \Lambda_1 \Rightarrow \Lambda_2} \,.\, \Theta_1) \circ \text{🔑}_{\hat{\Gamma} \,.\, \Lambda_2}^{\beta \in \Theta_1 \Rightarrow \Theta_2}\ \mathsf{sub}(\hat{\Gamma} \,.\, \Lambda_2 \,.\, \Theta_2 \to \hat{\Gamma} \,.\, \Lambda_1 \,.\, \Theta_1) \,@\, m}$$

Figure 5 Selected rules for σ-equivalence in WSMTT

2.3 σ-equivalence

Since substitution in WSMTT expressions is an explicit constructor, it does not compute (as will be the case in SFMTT in Section 3). This means that there are a lot of distinct WSMTT expressions that are actually equivalent. For example, from the perspective of the rules in Figure 3 the expressions $t\ [\,\sigma\,]_{\mathsf{ws}}\ [\,\tau\,]_{\mathsf{ws}}$ and $t\ [\,\sigma \circ \tau\,]_{\mathsf{ws}}$ have nothing to do with each other. For this reason, we add an axiomatic system to the intrinsically scoped WSMTT syntax that specifies when two expressions or substitutions are σ-equivalent (note that we do not add β- or η-equivalence to this system yet, those are covered in the type system that is defined on top of the syntax described here).

Some of the rules for σ-equivalence can be found in Figure 5. We make use of a judgment $\hat{\Gamma} \vdash_{\mathsf{ws}} t =^{\sigma} s\ \mathsf{expr} \,@\, m$ for expressions and $\vdash_{\mathsf{ws}} \sigma =^{\sigma} \tau\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \,@\, m$ for substitutions. We find rules expressing the connection between applying a composed substitution and consecutively applying both substitutions, expressing how to push a substitution through expression constructors such as λ^{μ} (here σ^{+} is the lifting of σ defined as $\sigma^{+} = (\sigma \circ \pi).\mathbf{v}_0$) and expressing functoriality of locks on substitutions. There are also quite some rules that express properties of key substitutions: their naturality and their behaviour with respect to the unit 2-cell and vertical and horizontal composition of 2-cells. The full definition of σ-equivalence for WSMTT can be found in the technical report.

3 Substitution Algorithm

In this section we describe our substitution algorithm for MTT. For this purpose we introduce a new language called SFMTT (for substitution-free MTT), which has no expression constructor for substitutions like WSMTT-EXPR-SUB in Figure 3. We also introduce renamings and substitutions for SFMTT. All of this is included in Section 3.1. We then proceed in

$$\frac{\mu : m \to n \quad \Theta : \mathsf{LockTele}(n \to m) \quad \hat{\Gamma}\ \mathsf{sctx} @ n \qquad \alpha \in \mu \Rightarrow \mathsf{locks}(\Theta)}{\hat{\Gamma} . \mu . \Theta \vdash_{\mathsf{sf}} \mathbf{v}_0^{\alpha}\ \mathsf{var} @ m}\ \text{SF-VAR-ZERO}$$

$$\frac{\mu : o \to n \quad \Theta : \mathsf{LockTele}(n \to m) \quad \hat{\Gamma} . \Theta \vdash_{\mathsf{sf}} v\ \mathsf{var} @ m}{\hat{\Gamma} . \mu . \Theta \vdash_{\mathsf{sf}} \mathsf{suc}\,(v)\ \ \mathsf{var} @ m}\ \text{SF-VAR-SUC}$$

Figure 6 Definition of well-scoped SFMTT variables

Section 3.2 to the core part of the substitution algorithm: applying SFMTT renamings and substitutions to SFMTT expressions. Finally, using this functionality we can translate WSMTT expressions to SFMTT expressions.

3.1 Substitution-free Multimode Type Theory (SFMTT)

3.1.1 SFMTT Expressions

Exactly like our presentation of WSMTT, the expressions in SFMTT will be extrinsically typed but intrinsically scoped. We can reuse the same notion of scoping context and lock telescope from Figure 2. However, to introduce SFMTT expressions we cannot just take all expression constructors from Figure 3 and drop the one handling substitution (WSMTT-EXPR-SUB). This would prevent us from accessing any other variable than the last one added to a scoping context and moreover we would no longer be able to take 2-cells into account.

For this reason, we introduce a new variable judgment $\hat{\Gamma} \vdash_{\mathsf{sf}} v\ \mathsf{var} @ m$ expressing that v is an accessible variable in scoping context $\hat{\Gamma}$ at mode m. The inference rules for this judgment can be found in Figure 6. Either we want to access the last variable in the scoping context, in which case we have to provide an appropriate 2-cell (SF-VAR-ZERO), or we skip the last variable in the scoping context, which may be located under a lock telescope (SF-VAR-SUC). As a conclusion, an SFMTT variable is just a De Bruijn index where the number zero is annotated with a 2-cell.

SFMTT expressions can now be introduced via a judgment $\hat{\Gamma} \vdash_{\mathsf{sf}} t\ \mathsf{expr} @ m$ stating that t is an SFMTT expression in scoping context $\hat{\Gamma}$ at mode m. The constructors are now more or less the same as those for intrinsically scoped WSMTT in Figure 3, where the constructors for variables and substituted expressions are not included. Furthermore, every variable $\hat{\Gamma} \vdash_{\mathsf{sf}} v\ \mathsf{var} @ m$ gives rise to an SFMTT expression in $\hat{\Gamma}$. We emphasize that SFMTT expressions cannot contain substitutions.

3.1.2 SFMTT Renamings and Substitutions

We can also define substitutions for the SFMTT syntax, which will be required in the next section. As in our intrinsically scoped presentation of WSMTT, every SFMTT renaming and substitution has a domain and a codomain scoping context. This ensures that applying a renaming or substitution to an SFMTT expression is a total (always defined) operation.

Similar to McBride [21] and Allais et al. [4], we define an action of renaming on expressions before we discuss the action of substitutions. Such a renaming does not only allow us to lift a substitution when pushing it under a binder, but also to perform some modal operations. Of course, we have to take into account that we want a structurally recursive substitution algorithm, which is impossible when substitution composition is added as a constructor. We solve this problem by first defining atomic renamings and substitutions, which are not closed under composition but which can be applied to SFMTT expressions in a structurally recursive

$$\textsc{sf-arensub-empty}\quad \frac{}{\vdash_{\mathsf{sf}}\ !\ \mathsf{aren/asub}(\hat{\Gamma} \to \cdot) \mathrel{@} m}$$

$$\textsc{sf-arensub-id}\quad \frac{\hat{\Gamma}\ \mathsf{sctx} \mathrel{@} m}{\vdash_{\mathsf{sf}} \mathsf{id}^{\mathsf{a}}\ \mathsf{aren/asub}(\hat{\Gamma} \to \hat{\Gamma}) \mathrel{@} m}$$

$$\textsc{sf-arensub-weaken}\quad \frac{\vdash_{\mathsf{sf}} \sigma\ \mathsf{aren/asub}(\hat{\Gamma} \to \hat{\Delta}) \mathrel{@} m}{\vdash_{\mathsf{sf}} \mathsf{weaken}(\sigma)\ \mathsf{aren/asub}(\hat{\Gamma} \mathbin{.} \mu \to \hat{\Delta}) \mathrel{@} m}$$

$$\textsc{sf-arensub-lock}\quad \frac{\vdash_{\mathsf{sf}} \sigma\ \mathsf{aren/asub}(\hat{\Gamma} \to \hat{\Delta}) \mathrel{@} n \qquad \mu : m \to n}{\vdash_{\mathsf{sf}} \sigma \mathbin{.} \text{🔒}_{\mu}\ \mathsf{aren/asub}(\hat{\Gamma} \mathbin{.} \text{🔒}_{\mu} \to \hat{\Delta} \mathbin{.} \text{🔒}_{\mu}) \mathrel{@} m}$$

$$\textsc{sf-arensub-key}\quad \frac{\Theta, \Psi : \mathsf{LockTele}(n \to m) \qquad \alpha \in \mathsf{locks}(\Theta) \Rightarrow \mathsf{locks}(\Psi)}{\vdash_{\mathsf{sf}} \text{🔑}^{\alpha \in \Theta \Rightarrow \Psi}_{\hat{\Gamma}}\ \mathsf{aren/asub}(\hat{\Gamma} \mathbin{.} \Psi \to \hat{\Gamma} \mathbin{.} \Theta) \mathrel{@} m}$$

$$\textsc{sf-aren-extend}\quad \frac{\vdash_{\mathsf{sf}} \sigma\ \mathsf{aren}(\hat{\Gamma} \to \hat{\Delta}) \mathrel{@} m \qquad \hat{\Gamma} \mathbin{.} \text{🔒}_{\mu} \vdash_{\mathsf{sf}} v\ \mathsf{var} \mathrel{@} n}{\vdash_{\mathsf{sf}} \sigma.v\ \mathsf{aren}(\hat{\Gamma} \to \hat{\Delta} \mathbin{.} \mu) \mathrel{@} m}$$

$$\textsc{sf-asub-extend}\quad \frac{\vdash_{\mathsf{sf}} \sigma\ \mathsf{asub}(\hat{\Gamma} \to \hat{\Delta}) \mathrel{@} m \qquad \hat{\Gamma} \mathbin{.} \text{🔒}_{\mu} \vdash_{\mathsf{sf}} t\ \mathsf{expr} \mathrel{@} n}{\vdash_{\mathsf{sf}} \sigma.t\ \mathsf{asub}(\hat{\Gamma} \to \hat{\Delta} \mathbin{.} \mu) \mathrel{@} m}$$

Figure 7 Definition of atomic SFMTT renamings and substitutions

$$\textsc{sf-rensub-id}\quad \frac{\hat{\Gamma}\ \mathsf{sctx} \mathrel{@} m}{\vdash_{\mathsf{sf}} \mathsf{id}\ \mathsf{ren/sub}(\hat{\Gamma} \to \hat{\Gamma}) \mathrel{@} m}$$

$$\textsc{sf-rensub-snoc}\quad \frac{\vdash_{\mathsf{sf}} \sigma\ \mathsf{ren/sub}(\hat{\Delta} \to \hat{\Xi}) \mathrel{@} m \qquad \vdash_{\mathsf{sf}} \tau\ \mathsf{aren/asub}(\hat{\Gamma} \to \hat{\Delta}) \mathrel{@} m}{\vdash_{\mathsf{sf}} \sigma \circledast \tau\ \mathsf{ren/sub}(\hat{\Gamma} \to \hat{\Xi}) \mathrel{@} m}$$

Figure 8 Definition of regular SFMTT renamings and substitutions

way. Regular renamings and substitutions (from now on also referred to as rensubs) will be defined in terms of these atomic rensubs. We add a judgment $\vdash_{\mathsf{sf}} \sigma\ \mathsf{aren/asub}(\hat{\Gamma} \to \Delta) \mathrel{@} m$ to denote that σ is an atomic renaming or substitution (much of the structure between renamings and substitutions is shared) from $\hat{\Gamma}$ to $\hat{\Delta}$ at mode m. There is a similar judgment $\vdash_{\mathsf{sf}} \sigma\ \mathsf{ren/sub}(\hat{\Gamma} \to \hat{\Delta}) \mathrel{@} m$ for regular rensubs.

The actual definition of atomic rensubs can be found in Figure 7. Many of the constructors are similar to the ones for WSMTT substitutions, such as the empty atomic rensub (sf-arensub-empty), locking (sf-arensub-lock) and keys (sf-arensub-key). As explained, we purposely omit a constructor for composition of atomic rensubs. As a consequence, we need a constructor for weakening rensubs (sf-arensub-weaken) which in WSMTT would have been accomplished by precomposing with π. Also note that we have an atomic identity rensub id^{a} (sf-arensub-id). We could have alternatively implemented id^{a} in terms of the other constructors but taking it as a constructor will make the rest of the paper easier because we can define its action on expressions to be trivial, whereas that would require a non-trivial proof in case of a defined identity atomic rensub. The only difference between atomic renamings and substitutions is the way they can be extended: a renaming is extended by a variable (sf-aren-extend) whereas a substitution can be extended with an arbitrary SFMTT expression (sf-asub-extend).

The full definition of regular rensubs is shown in Figure 8. In essence, they are well-scoped snoc-lists of atomic substitutions. They can be empty, in which case the rensub is called the identity (sf-rensub-id), or they consist of an atomic rensub postcomposed with a regular rensub (sf-rensub-snoc).

One operation that we will need in the next section, is the lifting of atomic rensubs. Given an atomic rensub σ from $\hat{\Gamma}$ to $\hat{\Delta}$, we can construct a new, lifted atomic rensub

$$\sigma^{+} := \mathsf{weaken}(\sigma).\mathbf{v}_0^{1_\mu} \qquad \text{for an atomic rensub } \sigma$$

from $\hat{\Gamma} \,.\, \mu$ to $\hat{\Delta} \,.\, \mu$ (here $\mathbf{v}_0^{1_\mu}$ is interpreted as a variable in the case of renamings and as an expression in the case of substitutions).[5] Moreover, for any scoping context $\hat{\Gamma}$ and modality μ, we have a weakening atomic rensub

$$\pi := \mathsf{weaken}(\mathsf{id}^{\mathsf{a}})$$

from $\hat{\Gamma} \,.\, \mu$ to $\hat{\Gamma}$. The lift and lock operations can be extended to regular rensubs by applying those operations to all constituent atomic rensubs. In other words, we have

$$\mathsf{id}^+ = \mathsf{id} \qquad\qquad \mathsf{id} \,.\, 🔒_\mu = \mathsf{id}$$
$$(\sigma \circledcirc \tau)^+ = \sigma^+ \circledcirc \tau^+ \qquad\qquad (\sigma \circledcirc \tau) \,.\, 🔒_\mu = (\sigma \,.\, 🔒_\mu) \circledcirc (\tau \,.\, 🔒_\mu).$$

3.2 Renaming and Substitution Algorithm for SFMTT

We are now ready to describe one of the core parts of the paper: the algorithm for applying an SFMTT substitution to an SFMTT expression. The definition is built up in 4 steps, each defining the action of another class of syntactic objects on SFMTT expressions:

1. Atomic renamings.
2. Regular renamings.
3. Atomic substitutions.
4. Regular substitutions.

However, there is considerable overlap between some of these steps. For this reason, we will treat steps 2 and 4 together as well as large parts of steps 1 and 3.[6] All operations take an (atomic) rensub from $\hat{\Gamma}$ to $\hat{\Delta}$ and an SFMTT expression in scoping context $\hat{\Delta}$ to produce an SFMTT expression in scoping context $\hat{\Gamma}$.

3.2.1 Atomic rensubs acting on non-variable expressions

We first discuss the application of atomic rensubs on SFMTT expressions other than variables. Note that we present the operation here as if it were acting on raw syntax, but strictly speaking it works on derivations of judgments of the form $\hat{\Gamma} \vdash_{\mathsf{sf}} t \;\mathsf{expr} \,@\, m$ (which are however fully determined by the expression itself).

$$\langle \mu \mid A \rangle \, [\, \sigma \,]_{\mathsf{aren/asub}} = \langle \mu \mid A \, [\, \sigma \,.\, 🔒_\mu \,]_{\mathsf{aren/asub}} \rangle$$
$$\mathsf{mod}_\mu \,(t) \, [\, \sigma \,]_{\mathsf{aren/asub}} = \mathsf{mod}_\mu \left(t \, [\, \sigma \,.\, 🔒_\mu \,]_{\mathsf{aren/asub}} \right)$$
$$((\mu \mid A) \to B) \, [\, \sigma \,]_{\mathsf{aren/asub}} = \left(\mu \mid A \, [\, \sigma \,.\, 🔒_\mu \,]_{\mathsf{aren/asub}} \right) \to B \, [\, \sigma^+ \,]_{\mathsf{aren/asub}}$$
$$(\lambda^\mu \,(t)) \, [\, \sigma \,]_{\mathsf{aren/asub}} = \lambda^\mu \left(t \, [\, \sigma^+ \,]_{\mathsf{aren/asub}} \right)$$
$$\mathsf{app}_\mu \,(f; t) \, [\, \sigma \,]_{\mathsf{aren/asub}} = \mathsf{app}_\mu \left(f \, [\, \sigma \,]_{\mathsf{aren/asub}} \,; t \, [\, \sigma \,.\, 🔒_\mu \,]_{\mathsf{aren/asub}} \right)$$

---

[5] It might be surprising that this works for substitutions too, since we explained in the introduction for STLC that defining weakening for substitutions requires recursively applying a subtitution (or renaming) to terms in the context. However, substituting a variable with $\mathsf{weaken}(\sigma)$ will involve the application of a renaming, as we will see in the next section.

[6] In fact, the action of regular renamings is not really used anywhere. Only atomic renamings will be important. However, as already mentioned the treatment of regular renamings and regular substitutions is entirely the same.

3.2.2 Atomic renamings acting on variables

We now turn to the case for variables. This is where we distinguish between atomic renamings and atomic substitutions. We first discuss the action of an atomic renaming on a variable, producing another variable. The intuitive "type signature" of this operation is too weak to make recursion work. In particular, it does not allow us to go under locks in renamings. Therefore, we have a result that generalizes over a lock telescope Λ, but we can recover the desired result by taking the empty lock telescope for Λ.

▶ **Lemma 3.** *If we have an SFMTT atomic renaming* $\vdash_{\mathsf{sf}} \sigma\ \mathsf{aren}(\hat{\Gamma} \to \hat{\Delta}) \,@\, n$, *a lock telescope* $\Lambda : \mathsf{LockTele}(n \to m)$ *and an SFMTT variable* $\hat{\Delta} \,.\, \Lambda \vdash_{\mathsf{sf}} v\ \mathsf{var} \,@\, m$, *then we can deduce* $\hat{\Gamma} \,.\, \Lambda \vdash_{\mathsf{sf}} v\ [\,\sigma\,]^{\Lambda}_{\mathsf{aren,var}}\ \mathsf{var} \,@\, m$

Lemma 3 is a core lemma for this paper. Our substitution algorithm crucially relies on identifying a notion of renamings that can be recursively applied to MTT terms. It is this lemma that establishes that our choices achieve this and we include the proof below because it clarifies well why atomic renamings should be defined as they are.

In the proof of Lemma 3 we will make use of the following result.

▶ **Lemma 4.** *Given two lock telescopes* $\Theta, \Psi : \mathsf{LockTele}(n \to m)$ *and a 2-cell* $\alpha \in \mathsf{locks}\,(\Theta) \Rightarrow \mathsf{locks}\,(\Psi)$, *we can transform a variable* $\hat{\Gamma} \,.\, \Theta \vdash_{\mathsf{sf}} v\ \mathsf{var} \,@\, m$ *to a variable* $\hat{\Gamma} \,.\, \Psi \vdash_{\mathsf{sf}} v\ [\,\alpha\,]^{\Theta \Rightarrow \Psi}_{\mathsf{2\text{-}cell}}\ \mathsf{var} \,@\, m$.

Proof. We proceed by induction on the variable v (i.e. the annotated De Bruijn index).

- CASE $\hat{\Gamma} \,.\, \Theta \vdash_{\mathsf{sf}} \mathbf{v}^{\beta}_{0}\ \mathsf{var} \,@\, m$ with $\hat{\Gamma} = \hat{\Delta} \,.\, \mu \,.\, \Lambda$ (SF-VAR-ZERO, Λ is a lock telescope so it only contains locks)
  We know that $\hat{\Delta} \,.\, \mu \,.\, \Lambda \,.\, \Theta \vdash_{\mathsf{sf}} \mathbf{v}^{\beta}_{0}\ \mathsf{var} \,@\, m$, so $\beta \in \mu \Rightarrow \mathsf{locks}\,(\Lambda \,.\, \Theta) = \mathsf{locks}\,(\Lambda) \circ \mathsf{locks}\,(\Theta)$. Using the horizontal composition $\star$, we can construct a 2-cell $1_{\mathsf{locks}(\Lambda)} \star \alpha \in \mathsf{locks}\,(\Lambda) \circ \mathsf{locks}\,(\Theta) \Rightarrow \mathsf{locks}\,(\Lambda) \circ \mathsf{locks}\,(\Psi)$. Hence we use the rule SF-VAR-ZERO again to obtain $\mathbf{v}^{\beta}_{0}\ [\,\alpha\,]^{\Theta \Rightarrow \Psi}_{\mathsf{2\text{-}cell}} = \mathbf{v}_{0}^{(1_{\mathsf{locks}(\Lambda)} \star \alpha) \circ \beta}$.[7]
- CASE $\hat{\Gamma} \,.\, \Theta \vdash_{\mathsf{sf}} \mathsf{suc}\,(v)\ \mathsf{var} \,@\, m$ with $\hat{\Gamma} = \hat{\Delta} \,.\, \mu \,.\, \Lambda$ (SF-VAR-SUC, Λ is a lock telescope)
  In this case we have that $\hat{\Delta} \,.\, \Lambda \,.\, \Theta \vdash_{\mathsf{sf}} v\ \mathsf{var} \,@\, m$. The induction hypothesis then gives us $\hat{\Delta} \,.\, \Lambda \,.\, \Psi \vdash_{\mathsf{sf}} v\ [\,\alpha\,]^{\Theta \Rightarrow \Psi}_{\mathsf{2\text{-}cell}}\ \mathsf{var} \,@\, m$. Applying the rule SF-VAR-SUC again to this result gives us the desired variable, so $\mathsf{suc}\,(v)\ [\,\alpha\,]^{\Theta \Rightarrow \Psi}_{\mathsf{2\text{-}cell}} = \mathsf{suc}\left(v\ [\,\alpha\,]^{\Theta \Rightarrow \Psi}_{\mathsf{2\text{-}cell}}\right)$. ◀

Proof of Lemma 3. We proceed by induction on σ.

- CASE $\vdash_{\mathsf{sf}}\ !\ \mathsf{aren}(\hat{\Gamma} \to \cdot) \,@\, n$
  In this case, $\hat{\Delta}$ is the empty scoping context. We can see from Figure 6 that there can be no variables in the empty scoping context (the scoping contexts in conclusions of both inference rules both contain at least a variable annotation). Hence we do not have to deal with this case further.[8]
- CASE $\vdash_{\mathsf{sf}} \mathsf{id}^{\mathsf{a}}\ \mathsf{aren}(\hat{\Gamma} \to \hat{\Gamma}) \,@\, n$
  Now $\hat{\Gamma} \,.\, \Lambda \vdash_{\mathsf{sf}} v\ \mathsf{var} \,@\, m$, so we can just say $v\ [\,\mathsf{id}^{\mathsf{a}}\,]^{\Lambda}_{\mathsf{aren,var}} = v$.
- CASE $\vdash_{\mathsf{sf}} \mathsf{weaken}(\sigma)\ \mathsf{aren}(\hat{\Gamma} \,.\, \mu \to \hat{\Delta}) \,@\, n$
  We know that $\hat{\Delta} \,.\, \Lambda \vdash_{\mathsf{sf}} v\ \mathsf{var} \,@\, m$, so we can use the induction hypothesis for σ and obtain a variable $\hat{\Gamma} \,.\, \Lambda \vdash_{\mathsf{sf}} v\ [\,\sigma\,]^{\Lambda}_{\mathsf{aren,var}}\ \mathsf{var} \,@\, m$. Since Λ is a lock telescope not containing variable

---

7 This definition seems to imply a dependency of $\mathbf{v}^{\beta}_{0}\ [\,\alpha\,]^{\Theta \Rightarrow \Psi}_{\mathsf{2\text{-}cell}}$ on Λ, but note that Λ is completely determined by the scoping context and the variable.

8 This case illustrates why it is advantageous to use intrinsically scoped syntax. It makes sure that the codomain of the renaming and the scoping context of the expression match, so we do not have to cover insensible cases.

annotations, we can then apply the rule SF-VAR-SUC from Figure 6 with $\Theta = \Lambda$ to obtain a variable in $\hat{\Gamma} \,.\, \mu \,.\, \Lambda$ as required. In other words, $v\ [\mathsf{weaken}(\sigma)\,]^{\Lambda}_{\mathsf{aren,var}} = \mathsf{suc}\left(v\ [\,\sigma\,]^{\Lambda}_{\mathsf{aren,var}}\right)$.

- CASE $\vdash_{\mathsf{sf}} \sigma \,.\, \text{🔒}_\mu\ \mathsf{aren}(\hat{\Gamma} \,.\, \text{🔒}_\mu \to \hat{\Delta} \,.\, \text{🔒}_\mu) \mathbin{@} n$
  Adding the 🔒_μ to the left of the lock telescope Λ, we get $v\ [\,\sigma \,.\, \text{🔒}_\mu\,]^{\Lambda}_{\mathsf{aren,var}} = v\ [\,\sigma\,]^{\text{🔒}_\mu \,.\, \Lambda}_{\mathsf{aren,var}}$.
- CASE $\vdash_{\mathsf{sf}} \text{🔑}^{\beta \in \Theta \Rightarrow \Psi}_{\hat{\Gamma}}\ \mathsf{aren}(\hat{\Gamma} \,.\, \Psi \to \hat{\Gamma} \,.\, \Theta) \mathbin{@} n$
  We have that $\hat{\Gamma} \,.\, \Theta \,.\, \Lambda \vdash_{\mathsf{sf}} v\ \mathsf{var} \mathbin{@} m$ and that $\beta \in \mathsf{locks}\,(\Theta) \Rightarrow \mathsf{locks}\,(\Psi)$. This means that $\beta \star 1_{\mathsf{locks}(\Lambda)} \in \mathsf{locks}\,(\Theta \,.\, \Lambda) \Rightarrow \mathsf{locks}\,(\Psi \,.\, \Lambda)$. Using Lemma 4, we can use this 2-cell to obtain a variable in $\hat{\Gamma} \,.\, \Psi \,.\, \Lambda$, so $v\ \left[\text{🔑}^{\beta \in \Theta \Rightarrow \Psi}_{\hat{\Gamma}}\right]^{\Lambda}_{\mathsf{aren,var}} = v\ [\,\beta \star 1_{\mathsf{locks}(\Lambda)}\,]^{\Theta \,.\, \Lambda \Rightarrow \Psi \,.\, \Lambda}_{\mathsf{2\text{-}cell}}$.
- CASE $\vdash_{\mathsf{sf}} \sigma.w\ \mathsf{aren}(\hat{\Gamma} \to \hat{\Delta} \,.\, \mu) \mathbin{@} n$
  We know that $\hat{\Delta} \,.\, \mu \,.\, \Lambda \vdash_{\mathsf{sf}} v\ \mathsf{var} \mathbin{@} m$ (where Λ contains only locks) and perform a case split on v.
  - CASE $\hat{\Delta} \,.\, \mu \,.\, \Lambda \vdash_{\mathsf{sf}} \mathbf{v}^{\alpha}_{0}\ \mathsf{var} \mathbin{@} m$
    In this case we have a 2-cell $\alpha \in \mu \Rightarrow \mathsf{locks}\,(\Lambda)$. Moreover, from the way the substitution is constructed we know that $\hat{\Gamma} \,.\, \text{🔒}_\mu \vdash_{\mathsf{sf}} w\ \mathsf{var} \mathbin{@} m$. We can then use Lemma 4 with lock telescopes $\Theta = \text{🔒}_\mu$ and $\Psi = \Lambda$ to transform w to a variable in $\hat{\Gamma} \,.\, \Lambda$. In other words, $\mathbf{v}^{\alpha}_{0}\ [\,\sigma.w\,]^{\Lambda}_{\mathsf{aren,var}} = w\ [\,\alpha\,]^{\text{🔒}_\mu \Rightarrow \Lambda}_{\mathsf{2\text{-}cell}}$.
  - CASE $\hat{\Delta} \,.\, \mu \,.\, \Lambda \vdash_{\mathsf{sf}} \mathsf{suc}\,(v)\ \ \mathsf{var} \mathbin{@} m$
    Now we know that $\hat{\Delta} \,.\, \Lambda \vdash_{\mathsf{sf}} v\ \mathsf{var} \mathbin{@} m$ and that $\vdash_{\mathsf{sf}} \sigma\ \mathsf{aren}(\hat{\Gamma} \to \hat{\Delta}) \mathbin{@} n$. Consequently, we can use the induction hypothesis to obtain a variable in $\hat{\Gamma} \,.\, \Lambda$. So $\mathsf{suc}\,(v)\ [\,\sigma.w\,]^{\Lambda}_{\mathsf{aren,var}} = v\ [\,\sigma\,]^{\Lambda}_{\mathsf{aren,var}}$. ◀

Note that the algorithm presented in the proof of Lemma 3 is indeed structurally recursive: in every recursive call the substitution gets structurally smaller (and moreover the algorithm in the proof of Lemma 4 does not depend on that of Lemma 3).

Together with the equations from Section 3.2.1, we have now proved the following.

▶ **Lemma 5** (Admissibility of atomic renaming). *If* $\vdash_{\mathsf{sf}} \sigma\ \mathsf{aren}(\hat{\Gamma} \to \hat{\Delta}) \mathbin{@} m$ *and* $\hat{\Delta} \vdash_{\mathsf{sf}} t\ \mathsf{expr} \mathbin{@} m$*, then we can deduce* $\hat{\Gamma} \vdash_{\mathsf{sf}} t\ [\,\sigma\,]_{\mathsf{aren}}\ \ \mathsf{expr} \mathbin{@} m$*.*

3.2.3 Atomic substitutions acting on variables

We now describe the action of atomic substitutions on variables. This will produce an SFMTT expression, that is not necessarily a variable anymore (as was the case for atomic renamings). We have a result very similar to Lemma 3.

▶ **Lemma 6.** *If we have an SFMTT atomic substitution* $\vdash_{\mathsf{sf}} \sigma\ \mathsf{asub}(\hat{\Gamma} \to \hat{\Delta}) \mathbin{@} n$*, a lock telescope* $\Lambda : \mathsf{LockTele}(n \to m)$ *and an SFMTT variable* $\hat{\Delta} \,.\, \Lambda \vdash_{\mathsf{sf}} v\ \mathsf{var} \mathbin{@} m$*, then we can deduce* $\hat{\Gamma} \,.\, \Lambda \vdash_{\mathsf{sf}} v\ [\,\sigma\,]^{\Lambda}_{\mathsf{asub,var}}\ \ \mathsf{expr} \mathbin{@} m$*.*

Proof. Again we proceed by case distinction and induction on σ. The cases for $!$, id^{a}, $\sigma \,.\, \text{🔒}_\mu$ and $\text{🔑}^{\beta \in \Theta \Rightarrow \Psi}_{\hat{\Gamma}}$ are similar to the proof of Lemma 3 so we omit them.

- CASE $\vdash_{\mathsf{sf}} \mathsf{weaken}(\sigma)\ \mathsf{asub}(\hat{\Gamma} \,.\, \mu \to \hat{\Delta}) \mathbin{@} n$
  We have that $\hat{\Delta} \,.\, \Lambda \vdash_{\mathsf{sf}} v\ \mathsf{var} \mathbin{@} m$, so we can use the induction hypothesis to obtain an expression in $\hat{\Gamma} \,.\, \Lambda$. Then we can apply Lemma 5 with the atomic renaming $\pi \,.\, \Lambda$ (i.e. applying all the locks from Λ to π) to obtain an expression in $\hat{\Gamma} \,.\, \mu \,.\, \Lambda$ as required. Consequently, we have $v\ [\mathsf{weaken}(\sigma)\,]^{\Lambda}_{\mathsf{asub,var}} = \left(v\ [\,\sigma\,]^{\Lambda}_{\mathsf{asub,var}}\right)\ [\,\pi \,.\, \Lambda\,]_{\mathsf{aren}}$.
- CASE $\vdash_{\mathsf{sf}} \sigma.t\ \mathsf{asub}(\hat{\Gamma} \to \hat{\Delta} \,.\, \mu) \mathbin{@} n$
  We know that $\hat{\Delta} \,.\, \mu \,.\, \Lambda \vdash_{\mathsf{sf}} v\ \mathsf{var} \mathbin{@} m$ and perform a case split and induction on v.

- CASE $\hat{\Delta} \mathbin{.} \mu \mathbin{.} \Lambda \vdash_{\mathsf{sf}} \mathbf{v}_0^{\alpha}\ \mathsf{var} \mathbin{@} m$
  In this case $\alpha \in \mu \Rightarrow \mathsf{locks}\,(\Lambda)$ and $\hat{\Gamma} \mathbin{.} \text{🔒}_{\mu} \vdash_{\mathsf{sf}} t\ \mathsf{expr} \mathbin{@} m$. Therefore, we can apply Lemma 5 with the renaming $\text{🔑}_{\hat{\Gamma}}^{\alpha \in \text{🔒}_{\mu} \Rightarrow \Lambda}$ and the expression t to obtain an expression in $\hat{\Gamma} \mathbin{.} \Lambda$. In other words $\mathbf{v}_0^{\alpha}\ [\,\sigma.t\,]^{\Lambda}_{\mathsf{asub,var}} = t\ \left[\,\text{🔑}_{\hat{\Gamma}}^{\alpha \in \text{🔒}_{\mu} \Rightarrow \Lambda}\,\right]_{\mathsf{aren}}$.
- CASE $\hat{\Delta} \mathbin{.} \mu \mathbin{.} \Lambda \vdash_{\mathsf{sf}} \mathsf{suc}\,(v)\ \ \mathsf{var} \mathbin{@} m$
  Now $\hat{\Delta} \mathbin{.} \Lambda \vdash_{\mathsf{sf}} v\ \mathsf{var} \mathbin{@} m$, so we can apply the induction hypothesis to v and σ. Hence $\mathsf{suc}\,(v)\ [\,\sigma.t\,]^{\Lambda}_{\mathsf{asub,var}} = v\ [\,\sigma\,]^{\Lambda}_{\mathsf{asub,var}}$. ◀

Note that all of the cases in the previous proof are similar to the corresponding cases in the proof of Lemma 3. The most important difference is that the result of applying a substitution is an expression and not a variable. In order to transform the results from recursive calls, we therefore make use of the fact that atomic renamings act on expressions as proved in Lemma 5 (as opposed to directly manipulating variables as in the proof of Lemma 3). This is reminiscent of how renaming gets used in the definition of substitution in [21, 4].

As a corollary, we get the following.

▶ **Lemma 7** (Admissibility of atomic substitution). *If* $\vdash_{\mathsf{sf}} \sigma\ \mathsf{asub}(\hat{\Gamma} \to \hat{\Delta}) \mathbin{@} m$ *and* $\hat{\Delta} \vdash_{\mathsf{sf}} t\ \mathsf{expr} \mathbin{@} m$*, then we can deduce* $\hat{\Gamma} \vdash_{\mathsf{sf}} t\ [\,\sigma\,]_{\mathsf{asub}}\ \mathsf{expr} \mathbin{@} m$*.*

3.2.4 Regular renamings/substitutions

We now turn to regular renamings and substitutions. There is no need to distinguish between these two as the procedure for renamings and substitutions will be exactly the same. Since a regular rensub is a sequence of atomic rensubs, we can just sequentially apply the results from the previous sections. We therefore get the following.

$$t\ [\,\mathsf{id}\,]_{\mathsf{ren/sub}} = t \qquad\qquad t\ [\,\sigma \circledcirc \tau\,]_{\mathsf{ren/sub}} = \left(t\ [\,\sigma\,]_{\mathsf{ren/sub}}\right)\ [\,\tau\,]_{\mathsf{aren/asub}}$$

As a conclusion, we have proved the following theorem.

▶ **Theorem 8** (Admissibility of renaming and substitution). *Given a renaming or substitution* $\vdash_{\mathsf{sf}} \sigma\ \mathsf{ren/sub}(\hat{\Gamma} \to \hat{\Delta}) \mathbin{@} m$ *and an SFMTT expression* $\hat{\Delta} \vdash_{\mathsf{sf}} t\ \mathsf{expr} \mathbin{@} m$*, we can deduce* $\hat{\Gamma} \vdash_{\mathsf{sf}} t\ [\,\sigma\,]_{\mathsf{ren/sub}}\ \mathsf{expr} \mathbin{@} m$*.*

Note that we do not actually need the action of full renamings on SFMTT expressions in order to define the action of atomic substitutions, atomic renamings suffice for that purpose.

Although we are not really concerned with performance in this paper, we note that optimisations are certainly possible. For example, as it is currently described, the algorithm will, when applying a regular substitution consisting of n atomic ones to an expression t, perform n traversals of t, one for every atomic substitution. This could be reduced by traversing the expression just once and applying lifting ($^{+}$) or locks to all atomic substitutions simultaneously when required.

3.3 Interpretation of WSMTT Expressions in SFMTT

We now turn to the relation between WSMTT and SFMTT. Using the substitution algorithm just defined, we will show that WSMTT expressions can be translated to SFMTT expressions, essentially proving that explicit substitutions can be computed away. The reverse direction is easier: apart from variables, every SFMTT expression constructor also appears in WSMTT so we can almost trivially embed the former system into the latter. We define the two translations here and consider their meta-theoretical properties (particularly soundness and completeness) in the next sections.

3.3.1 Translation from WSMTT to SFMTT

The translation from WSMTT to SFMTT is defined mutually recursively for both expressions and substitutions. In other words, for any WSMTT expression $\hat{\Gamma} \vdash_{\mathsf{ws}} t \ \mathsf{expr} \mathbin{@} m$ we get an SFMTT expression $\hat{\Gamma} \vdash_{\mathsf{sf}} [\![t]\!] \ \mathsf{expr} \mathbin{@} m$ and for any WSMTT substitution $\vdash_{\mathsf{ws}} \sigma \ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \mathbin{@} m$ we get an SFMTT (regular) substitution $\vdash_{\mathsf{sf}} [\![\sigma]\!] \ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \mathbin{@} m$. We only show some of the cases for the different expression constructors.

$$
\begin{aligned}
[\![\mathbf{v}_0]\!] &= \mathbf{v}_0^{1_\mu} & [\![\pi]\!] &= \mathsf{id} ⓐ \mathsf{weaken}(\mathsf{id}^{\mathsf{a}}) \\
[\![(\mu \mid A) \to B]\!] &= (\mu \mid [\![A]\!]) \to [\![B]\!] & [\![\sigma \circ \tau]\!] &= [\![\sigma]\!] \mathbin{+\!\!+} [\![\tau]\!] \\
[\![t \ [\ \sigma\ ]_{\mathsf{ws}}]\!] &= [\![t]\!] \ [\ [\![\sigma]\!]\ ]_{\mathsf{sub}} & [\![\sigma \,.\, 🔒_\mu]\!] &= [\![\sigma]\!] \,.\, 🔒_\mu \\
[\![!]\!] &= \mathsf{id} ⓐ ! & \left[\!\!\left[🔑_{\hat{\Gamma}}^{\alpha \in \Theta \Rightarrow \Psi} \right]\!\!\right] &= \mathsf{id} ⓐ 🔑_{\hat{\Gamma}}^{\alpha \in \Theta \Rightarrow \Psi} \\
[\![\mathsf{id}]\!] &= \mathsf{id} & [\![\sigma . t]\!] &= [\![\sigma]\!]^{+} ⓐ (\mathsf{id}^{\mathsf{a}} . [\![t]\!])
\end{aligned}
$$

When translating an (explicitly) substituted WSMTT expression $t\ [\ \sigma\ ]_{\mathsf{ws}}$, we translate both the expression t and the substitution σ and then apply Theorem 8 (i.e. the algorithm from the previous section). Translation of a composite substitution involves the concatenation of the two translated substitutions, which are regular SFMTT substitutions so sequences of atomic SFMTT substitutions. Recall that the operations $\_ \,.\, 🔒_\mu$ and $^{+}$ for regular SFMTT substitutions are defined at the end of Section 3.1. Finally, one could wonder why in the translation of $\sigma.t$ we first add $[\![t]\!]$ to the identity atomic substitution and then apply the lifted version of $[\![\sigma]\!]$ where it would seem easier to first apply (the non-lifted) $[\![\sigma]\!]$ and then extend id^{a} with $[\![t]\!]$. The answer is that in that case we would $[\![t]\!]$ would live in the wrong scoping context: if $[\![\sigma]\!]$ goes from $\hat{\Gamma}$ to $\hat{\Delta}$, then $[\![t]\!]$ lives in $\hat{\Gamma} \,.\, 🔒_\mu$ but if we want the translation of $\sigma.t$ to be of the form $(\mathsf{id}^{\mathsf{a}}.?) ⓐ [\![\sigma]\!]$, then we need some term in scoping context $\hat{\Delta} \,.\, 🔒_\mu$ at the place of the question mark.

3.3.2 Embedding of SFMTT into WSMTT

We only provide an embedding of SFMTT expressions to WSMTT expressions (so not for substitutions). Apart from the constructor for variable expressions, all SFMTT expression constructors also occur in WSMTT. We therefore only specify how to embed variables.

$$
\begin{aligned}
\mathsf{embed}(\mathbf{v}_0^{\alpha}) &= \mathbf{v}_0 \left[🔑_{\hat{\Gamma}}^{\alpha \in 🔒_\mu \Rightarrow \Theta} \right]_{\mathsf{ws}} \\
\mathsf{embed}(\mathsf{suc}\,(v)) &= \mathsf{embed}(v) \ [\ \pi \,.\, \Theta\ ]_{\mathsf{ws}}
\end{aligned}
$$

The lock telescopes Θ in both cases are inferred from the scoping context (recall that we consider SFMTT expressions to be intrinsically scoped).

As a result, for every SFMTT expression $\hat{\Gamma} \vdash_{\mathsf{sf}} t \ \mathsf{expr} \mathbin{@} m$ we get a corresponding WSMTT expression $\hat{\Gamma} \vdash_{\mathsf{ws}} \mathsf{embed}(t) \ \ \mathsf{expr} \mathbin{@} m$.

4 Soundness

In the previous section, we introduced a translation from WSMTT to SFMTT that uses our substitution algorithm to translate away WSMTT's explicit substitution. In this section and the next, we establish the translation's key properties: soundness and completeness of the translation with respect to σ-equivalence in WSMTT. First, in this section, we establish soundness: translating a WSMTT expression to SFMTT and embedding the result back into WSMTT should produce a WSMTT expression that is σ-equivalent to the original. Next, Section 5 will establish completeness: any two σ-equivalent WSMTT expressions are mapped to equal SFMTT terms. Soundness and completeness combined give us the result that SFMTT expressions can be regarded as the σ-normal forms of WSMTT expressions.

Specifically, in this section we prove the following result.

▶ **Theorem 9** (Soundness)**.** *Given a WSMTT expression* $\hat{\Gamma} \vdash_{\mathsf{ws}} t\ \mathsf{expr} \,@\, m$*, we have that* $\hat{\Gamma} \vdash_{\mathsf{ws}} \mathsf{embed}(\llbracket t \rrbracket) =^{\sigma} t\ \mathsf{expr} \,@\, m$*.*

In other words, if we start with a WSMTT expression, apply the translation where all explicit substitutions are computed away, and then embed the result back into WSMTT, we get a result that is σ-equivalent to the original expression.

Although we did not provide an embedding of substitutions in Section 3.3.2, the proof of Theorem 9 is easiest to formulate when we have such an embedding. The reason for this is that we will perform an induction on the WSMTT expression t, but WSMTT expressions are defined mutually recursively with WSMTT substitutions as can be seen in Figure 3. We therefore define the following for both atomic and regular SFMTT substitutions.

$$\mathsf{embed}(!) = ! \qquad \mathsf{embed}\left(\text{🔑}_{\hat{\Gamma}}^{\alpha \in \Lambda \Rightarrow \Theta}\right) = \text{🔑}_{\hat{\Gamma}}^{\alpha \in \Lambda \Rightarrow \Theta}$$

$$\mathsf{embed}(\mathsf{id}^{\mathsf{a}}) = \mathsf{id} \qquad \mathsf{embed}(\sigma . t) = \mathsf{embed}(\sigma)\, .\mathsf{embed}(t)$$

$$\mathsf{embed}(\mathsf{weaken}(\sigma)) = \mathsf{embed}(\sigma) \circ \pi \qquad \mathsf{embed}(\mathsf{id}) = \mathsf{id}$$

$$\mathsf{embed}(\sigma \,.\, \text{🔒}_{\mu}) = \mathsf{embed}(\sigma)\ .\text{🔒}_{\mu} \qquad \mathsf{embed}(\sigma \circledast \tau) = \mathsf{embed}(\sigma) \circ \mathsf{embed}(\tau)$$

The proof of Theorem 9 proceeds by induction and case analysis on the WSMTT expression t. The crucial case is when t is of the form $s\ [\,\sigma\,]_{\mathsf{ws}}$. In that case the induction hypothesis would give us $\hat{\Delta} \vdash_{\mathsf{ws}} \mathsf{embed}(\llbracket s \rrbracket) =^{\sigma} s\ \mathsf{expr} \,@\, m$ and $\vdash_{\mathsf{ws}} \mathsf{embed}(\llbracket \sigma \rrbracket) =^{\sigma} \sigma\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \,@\, m$. In order to derive the desired result from this, we need the following lemma.

▶ **Lemma 10.** *Given an SFMTT expression* $\hat{\Delta} \vdash_{\mathsf{sf}} t\ \mathsf{expr} \,@\, m$ *and substitution* $\vdash_{\mathsf{sf}} \sigma\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \,@\, m$*, we have that* $\hat{\Gamma} \vdash_{\mathsf{ws}} \mathsf{embed}(t\ [\,\sigma\,]_{\mathsf{sub}}) =^{\sigma} \mathsf{embed}(t)\ [\,\mathsf{embed}(\sigma)\,]_{\mathsf{ws}}\ \mathsf{expr} \,@\, m$*.*

This lemma tells us that computing away a substitution in SFMTT and embedding the result in WSMTT should give an expression that is σ-equivalent to the result of applying the WSMTT substitution constructor to the embedded substitution. The proof of Lemma 10 is technically quite involved (it proceeds by induction on t and σ, the most difficult cases being weakening and key substitutions) and can therefore be found in the technical report.

Using Lemma 10, we can sketch the proof of Theorem 9. In fact we will prove the following stronger result.

▶ **Theorem 11.** *For every WSMTT expression* $\hat{\Gamma} \vdash_{\mathsf{ws}} t\ \mathsf{expr} \,@\, m$ *we have* $\hat{\Gamma} \vdash_{\mathsf{ws}} \mathsf{embed}(\llbracket t \rrbracket) =^{\sigma} t\ \mathsf{expr} \,@\, m$ *and for every WSMTT substitution* $\vdash_{\mathsf{ws}} \sigma\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \,@\, m$ *we have* $\vdash_{\mathsf{ws}} \mathsf{embed}(\llbracket \sigma \rrbracket) =^{\sigma} \sigma\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) \,@\, m$*.*

Sketch of proof. This proof proceeds by induction on the expression t and the substitution σ. We only show 3 cases for t, the other cases can be found in the technical report.

- CASE $\hat{\Gamma} \,.\, \mu \,.\, \text{🔒}_{\mu} \vdash_{\mathsf{ws}} \mathbf{v}_0\ \mathsf{expr} \,@\, m$
  Now we have that $\mathsf{embed}(\llbracket \mathbf{v}_0 \rrbracket) = \mathsf{embed}\left(\mathbf{v}_0^{1_\mu}\right) = \mathbf{v}_0\ \left[\text{🔑}_{\hat{\Gamma}\,.\,\mu}^{1_\mu \in \text{🔒}_\mu \Rightarrow \text{🔒}_\mu}\right]_{\mathsf{ws}}$. This last expression is indeed σ-equivalent to $\mathbf{v}_0$ because of the functoriality of key substitutions.
- CASE $\hat{\Gamma} \vdash_{\mathsf{ws}} t\ [\,\sigma\,]_{\mathsf{ws}}\ \mathsf{expr} \,@\, m$
  In this case $\mathsf{embed}(\llbracket t\ [\,\sigma\,]_{\mathsf{ws}} \rrbracket) = \mathsf{embed}(\llbracket t \rrbracket\ [\,\llbracket \sigma \rrbracket\,]_{\mathsf{sub}}) =^{\sigma} \mathsf{embed}(\llbracket t \rrbracket)\ [\,\mathsf{embed}(\llbracket \sigma \rrbracket)\,]_{\mathsf{ws}}$ where

the last σ-equivalence holds because of Lemma 10. We can now apply the induction hypothesis to t to obtain $\mathsf{embed}(\llbracket t \rrbracket) =^{\sigma} t$ and to σ to get $\mathsf{embed}(\llbracket \sigma \rrbracket) =^{\sigma} \sigma$, which proves the desired result.

- CASE $\hat{\Gamma} \vdash_{\mathsf{ws}} \lambda^{\mu}(t)\ \mathsf{expr} @ m$
  By definition of the translation and embedding between WSMTT and SFMTT, we have that $\mathsf{embed}(\llbracket \lambda^{\mu}(t) \rrbracket) = \lambda^{\mu}(\mathsf{embed}(\llbracket t \rrbracket))$. Hence the result follows from the induction hypothesis applied to the subterm t. ◀

5 Completeness

Completeness of our algorithm with respect to σ-equivalence states that whenever two WSMTT expressions are σ-equivalent, the results when computing away all substitutions in these expressions should be the same. Hence we want to prove the following theorem.

▶ **Theorem 12** (Completeness). *If we can deduce $\hat{\Gamma} \vdash_{\mathsf{ws}} t =^{\sigma} s\ \mathsf{expr} @ m$, then $\llbracket t \rrbracket = \llbracket s \rrbracket$.*

Recall that σ-equivalence for WSMTT expressions is defined mutually recursively with σ-equivalence for WSMTT substitutions (see Figure 5). Therefore, in order to prove Theorem 12, we need to first extend it so as to also make a claim about σ-equivalent WSMTT substitutions. However, in SFMTT, syntactic equality of substitutions is not a good notion of equivalence. Instead, we will use the following.

▶ **Definition 13** (Observational equivalence). *We say that two SFMTT substitutions $\vdash_{\mathsf{sf}} \sigma, \tau\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) @ m$ are observationally equivalent when $t\,[\,\sigma\,]_{\mathsf{sub}} = t\,[\,\tau\,]_{\mathsf{sub}}$ for every expression $\hat{\Delta} \vdash_{\mathsf{sf}} t\ \mathsf{expr} @ m$. We will write this as $\sigma \approx^{\mathsf{obs}} \tau$.*

This notion of observational equivalence is actually quite strong because it quantifies over all possible SFMTT expressions. That means that both substitutions might get pushed under a lot of expression constructors, with locks or lifts added along the way. The technical report proves the following lemma, which makes it easier to prove observational equivalence.

▶ **Lemma 14.** *Let $\vdash_{\mathsf{sf}} \sigma, \tau\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) @ n$ be two SFMTT substitutions and suppose that $v\,[\,\sigma \,.\, \Lambda\,]_{\mathsf{sub}} = v\,[\,\tau \,.\, \Lambda\,]_{\mathsf{sub}}$ for every lock telescope $\Lambda : \mathsf{sTele}(n \to m)$ and every variable $\hat{\Delta} \,.\, \Lambda \vdash_{\mathsf{sf}} v\ \mathsf{var} @ m$. Then $\sigma \approx^{\mathsf{obs}} \tau$.*

▶ Remark 15. If we instantiate SFMTT on the trivial mode theory (by which we mean the terminal 2-category) then variables are non-modal De Bruijn indices and lock telescopes can be essentially ignored. In this setting, what Lemma 14 really says is that a substitution is uniquely determined, up to observational equivalence, by its action on De Bruijn indices. Since there exists exactly one De Bruijn index for every variable in the context, this means that we have an injection from substitutions, up to observational equivalence, to lists of terms. In plain dependent type theory, substitutions are often *defined* as lists of terms, or at least it is clear that they can be uniquely represented in this way. In other words, the aforementioned injection is actually a bijection. In general SFMTT, this is no longer the case: we have a non-bijective injection, proving that the structure of substitutions in modal type theory is fundamentally more complex than that of plain dependent type theory. An example that proves the failure of surjectivity is given in the technical report.

We can now prove an extension of Theorem 12.

▶ **Theorem 16.** *Given two σ-equivalent WSMTT expressions $\hat{\Gamma} \vdash_{\mathsf{ws}} t =^{\sigma} s\ \mathsf{expr} @ m$, we have that $\llbracket t \rrbracket = \llbracket s \rrbracket$. Furthermore, given two σ-equivalent WSMTT substitutions $\vdash_{\mathsf{ws}} \sigma =^{\sigma} \tau\ \mathsf{sub}(\hat{\Gamma} \to \hat{\Delta}) @ m$, we have that $\llbracket \sigma \rrbracket \approx^{\mathsf{obs}} \llbracket \tau \rrbracket$.*

Sketch of proof. We proceed by induction on a derivation of the σ-equivalence judgment, going over all inference rules from Figure 5. Only some cases are covered, the other can be found in the technical report.

- CASE $\hat{\Gamma} \vdash_{\mathsf{ws}} t\,[\,\sigma \circ \tau\,]_{\mathsf{ws}} =^{\sigma} t\,[\,\sigma\,]_{\mathsf{ws}}\,[\,\tau\,]_{\mathsf{ws}}\ \mathsf{expr} @ m$
  For the left-hand side we get that $[\![t\,[\,\sigma \circ \tau\,]_{\mathsf{ws}}]\!] = [\![t]\!]\,[\,[\![\sigma]\!] +\!\!+ [\![\tau]\!]\,]_{\mathsf{sub}}$, whereas for the right-hand side we have $[\![t\,[\,\sigma\,]_{\mathsf{ws}}\,[\,\tau\,]_{\mathsf{ws}}]\!] = [\![t]\!]\,[\,[\![\sigma]\!]\,]_{\mathsf{sub}}\,[\,[\![\tau]\!]\,]_{\mathsf{sub}}$. Since applying a regular substitution to an SFMTT expression amounts to applying all constituent atomic substitutions, both expressions are equal.
- CASE $\hat{\Gamma} \vdash_{\mathsf{ws}} t\,[\,\tau\,]_{\mathsf{ws}} =^{\sigma} s\,[\,\sigma\,]_{\mathsf{ws}}\ \mathsf{expr} @ m$
  The premises of this inference rule tell us that $\hat{\Gamma} \vdash_{\mathsf{ws}} t =^{\sigma} s\ \mathsf{expr} @ m$ and $\vdash_{\mathsf{ws}} \tau =^{\sigma} \sigma\ \mathsf{sub}(\hat{\Delta} \to \hat{\Gamma}) @ m$. From the induction hypothesis it then follows that $[\![t]\!] = [\![s]\!]$ and $[\![\tau]\!] \approx^{\mathsf{obs}} [\![\sigma]\!]$. By the definition of $\approx^{\mathsf{obs}}$ we therefore have that $[\![t\,[\,\tau\,]_{\mathsf{ws}}]\!] = [\![t]\!]\,[\,[\![\tau]\!]\,]_{\mathsf{sub}} = [\![s]\!]\,[\,[\![\sigma]\!]\,]_{\mathsf{sub}} = [\![s\,[\,\sigma\,]_{\mathsf{ws}}]\!]$.
- CASE $\hat{\Gamma} \vdash_{\mathsf{ws}} (\lambda^{\mu}\,(t))\,[\,\sigma\,]_{\mathsf{ws}} =^{\sigma} \lambda^{\mu}\,(t\,[\,\sigma^{+}\,]_{\mathsf{ws}})\ \mathsf{expr} @ m$
  Since all atomic SFMTT substitutions can be pushed through $\lambda^{\mu}\,(\_)$ and the lifting of a regular substitution consists of the lifted atomic substitutions, we have $[\![(\lambda^{\mu}\,(t))\,[\,\sigma\,]_{\mathsf{ws}}]\!] = [\![\lambda^{\mu}\,(t)]\!]\,[\,[\![\sigma]\!]\,]_{\mathsf{sub}} = \lambda^{\mu}\,([\![t]\!])\,[\,[\![\sigma]\!]\,]_{\mathsf{sub}} = \lambda^{\mu}\left([\![t]\!]\left[\,[\![\sigma]\!]^{+}\,\right]_{\mathsf{sub}}\right)$. On the other hand we know that $[\![\lambda^{\mu}\,(t\,[\,\sigma^{+}\,]_{\mathsf{ws}})]\!] = \lambda^{\mu}\,([\![t]\!]\,[\,[\![\sigma^{+}]\!]\,]_{\mathsf{sub}})$. We conclude that both expressions are indeed equal because $[\![\sigma^{+}]\!] \approx^{\mathsf{obs}} [\![\sigma]\!]^{+}$ by a lemma proved in the technical report.
- CASE $\vdash_{\mathsf{ws}} (\sigma \circ \tau)\,.\,🔒_{\mu} =^{\sigma} (\sigma\,.\,🔒_{\mu}) \circ (\tau\,.\,🔒_{\mu})\ \mathsf{sub}(\hat{\Gamma}\,.\,🔒_{\mu} \to \hat{\Xi}\,.\,🔒_{\mu}) @ n$
  This case is trivial since a lock is applied to every atomic substitution in a sequence and hence it distributes over sequence concatenation. ◀

As a consequence of the soundness and completeness of our algorithm, we have the following result.

▶ **Theorem 17.** *Given two WSMTT expressions $\hat{\Gamma} \vdash_{\mathsf{ws}} t, s\ \mathsf{expr} @ m$, then $\hat{\Gamma} \vdash_{\mathsf{ws}} t =^{\sigma} s\ \mathsf{expr} @ m$ if and only if $[\![t]\!] = [\![s]\!]$. From this it follows that SFMTT expressions are the σ-normal forms of WSMTT expressions, and $[\![-]\!]$ is the normalization function.*

Proof. The direction from left to right is exactly Theorem 12. Conversely, suppose that $[\![t]\!] = [\![s]\!]$. Then we know that $t =^{\sigma} \mathsf{embed}([\![t]\!]) = \mathsf{embed}([\![s]\!]) =^{\sigma} s$. To show that SFMTT expressions are the σ-normal forms of WSMTT expressions, we only need to prove that every SFMTT expression is in the image of the $[\![-]\!]$ function. This is easily seen to be the case since $[\![\mathsf{embed}(t)]\!] = t$ for all $\hat{\Gamma} \vdash_{\mathsf{sf}} t\ \mathsf{expr} @ m$ (which is provable via a trivial induction on t). ◀

6 Related and Future Work

6.1 Normalization by Evaluation for MTT

Normalization of MTT w.r.t. $\sigma\beta\eta$-equality had already been proven by Gratzer [15][16, ch. 8]. He uses a normalization by evaluation (NbE) argument [5, 3], structured using the more recent technique of Synthetic Tait Computability (STC) [29][16, ch. 4]. We compare Gratzer's work with ours both in terms of approach and of implications.

Implications An NbE algorithm will take as input a term $\Gamma \vdash t : T$ (considered up to $\sigma\beta\eta$-equality) and a *value environment* $\rho : \mathsf{env}(\Delta \to \Gamma)$ and return a $\sigma\beta\eta$-normal form $\Delta \vdash \mathsf{nbe}(t, \rho) : T[\rho]$. When we instantiate ρ with the identity environment, which substitutes every variable with itself or its η-expansion, then we are really just normalizing t. When instead we are only interested in syntax up to $\sigma\beta\eta$-equality, and thus not in $\sigma\beta\eta$-normalization which is inobservable up to $\sigma\beta\eta$-equality, then the algorithm really just applies the substitution ρ to the term t. So in this sense an NbE algorithm already allows for substitution and indeed this is sufficient for a proof-of-concept implementation of MTT [28].

However, for conceptual, didactical and practical reasons, we see a role for a substitution and σ-normalization algorithm unreliant on $\beta\eta$-equality as presented in the current paper. Conceptually, there is the fact that substitution originates as a find-replace operation that replaces every occurrence of a given variable with a term of the same type. While the admissibility of such an operation becomes more difficult to prove with the introduction of variable binding, dependent types, ..., it is still a reasonable expectation and indeed a sanity check to ask that this operation be admissible, without referring to computation or $\beta\eta$-equality. It ensures that, even before considering computation, variables can be thought of as placeholders, and that programs are not permanently tied to the context in which they are defined, but merely use the context as an interface. Didactically, since computation relies on substitution, it is desirable to be able to explain substitution *first*, and especially without having to introduce NbE. Practically, when working in a dependently typed proof-assistant, we want to get type goals that are *not* in $\sigma\beta\eta$-normal form. For example, an η-normal form of an advanced algebraic structure will typically be a big nested record type listing all carriers and implementing all available operations, which may not be quite as readable as the more intensional way in which the algebra was constructed. A proof-assistant that relies on NbE for substitution, will not be able to type a function application $f\,a$ of a dependently typed function f without normalizing the codomain of f. Our algorithm, on the other hand, will cleanly push substitutions through all non-substitution-related syntax constructors and merely find and replace variables.

Approach A first stark difference between NbE and the current work is that NbE considers a type system's syntax up to $\sigma\beta\eta$-equality, i.e. it considers the type system's initial model in which important type formers can be characterized by their universal properties. In order to speak about σ-equality, we need to distinguish $\beta\eta$-equal terms and lose some of the categorical tooling. In particular, the category of models of a type system is of little use and most type formers do not satisfy their universal properties up to σ- or syntactic equality.

Similarly, because typing relies on $\beta\eta$-equality and we want to get the complications of substitution out of the way *before* considering $\beta\eta$-equality (e.g. because of the conceptual and didactical reasons above), we work with intrinsically scoped untyped syntax, whereas NbE generally works with intrinsically typed syntax.

NbE arguments generally feature at least five 'collections of program representations': variables, neutrals, normal forms, values, and $\sigma\beta\eta$-equivalence classes[9] of terms. An NbE proof involves several operations on and between these collections, and each of them is stable under renaming, which is necessary to deal with λ-abstraction and application. In the current work, we do not ever need to construct or eliminate functions, so while we do need to apply scoping telescopes to renamings and substitutions , it turns out there is no need to prove that every operation featured in the proof, is stable under renaming. Furthermore, while MTT and SFMTT can be regarded as the collections of terms and normal forms respectively, and we also have a definition of SFMTT variables, we do not need to distinguish between values and normal forms (which in NbE has mostly to do with η-equality) and we do not

[9] When formalizing type theory in type theory, one would not use set-theory-style quotients based on equivalence classes, but instead use quotient-inductive-inductive types [6].

need a separate collection of neutrals (as σ-reduction, unlike β-reduction, is never stuck on a variable).

6.2 Second-order Algebraic Theories

Allais, Atkey, Chapman, McBride and McKinna [4] implement renaming and substitution (among many other things) at once for a large class of languages, which Fiore and Szamozvancev [14] identify to be second-order multisorted algebraic theories (SOMATs). Here, multisorted means simply-typed, and second-order means that they accommodate variable-binding, but no other context features, i.e. it is assumed that contexts, renamings and substitutions are lists of types, variables and terms respectively. More recently and in a more categorical perspective, Uemura has defined the corresponding class of dependently typed languages, which in the larger naming scheme would be called second-order generalized algebraic theories (SOGATs). A similar general substitution result should be possible for SOGATs, and in any case it is very well understood (but considered tedious) how to prove admissibility of substitution for specific SOGATs, which is why there is nowadays little attention for this problem in the metatheory of specific non-modal languages.

The necessity of the current work arises from the fact that, due to the presence of locks, MTT is not a SOGAT. A generalization of second-order algebraic theories that would subsume MTT or at least Multimode Simple Type Theory (MSTT) [11] is work in progress [22] and will be informed by our current findings.

6.3 Other Approaches to Modal Contexts and/or Substitution

Lock calculi Bahr, Grathwohl and Møgelberg [7] introduce Clocked Type Theory (CloTT), a system for guarded type theory which features a *later* modality $\triangleright$ for every clock listed in the clock context. If we keep the clock context fixed, then to a large extent CloTT can be regarded as an instance of MTT,[10] but the 'lock' operation for each later modality is *named*. To clarify, we put the introduction rules for the later types for a clock κ in MTT and CloTT side by side:

$$\frac{\Gamma\,.\,🔒_{\triangleright^\kappa} \vdash t : T}{\Gamma \vdash \mathsf{mod}_{\triangleright^\kappa}(t) : \langle \triangleright^\kappa \mid T \rangle} \qquad \frac{\Gamma, \alpha : \kappa \vdash t : T}{\Gamma \vdash \lambda(\alpha : \kappa).t : \triangleright(\alpha : \kappa).T}$$

The variable α is called a tick of the clock κ, but we can more generally call it a lock variable. The specific mode theory for CloTT is enforced by requiring that α be used substructurally. This slightly complicates the type system but on the bright side, substitutions in CloTT are simply variable and tick replacement operations and do not have the complex categorical structure they have in MTT, facilitating implementation in Agda [31]. Dependent quantification over an affine or cartesian interval variable in cubical homotopy or parametric type theory [9, 13, 8] can also be regarded as an instance of this approach, with the interval variable being analogous to the tick.

We could similarly try to assign a lock variable to every lock in MTT and extend MTT with a substructural lock calculus [23]. This is challenging however, as we need to deal with arbitrarily complex mode theories and the lock calculus admits in general neither weakening, exchange nor contraction.

[10] Alternatively, we could regard the clock context as the mode, in which case we have an instance of MTT where clock substitution and quantification are also modalities. However, our discussion about lock calculi does not apply if we take that perspective.

2-posetal MTT If MTT is instantiated on a mode theory that is a 2-poset, meaning that the 2-cell of a given domain and codomain is unique if it exists, and if moreover this existence is decidable, then rather than listing 2-cell information on variables and in substitutions, the unique existence of the necessary 2-cells can be checked. Then all the remaining *information* in a substitution is a list of terms, and the substitution operation is again merely a find-replace operation. In the implementation of the proof-assistant Mitten [28], this fact is used to optimize the NbE algorithm (Section 6.1) for implementation.

Left division MTT is based on a line of work on type systems using a *left division* operation [2, 26, 25, 24], which in turn can be regarded as a generalization of a dual-context approach [27]. Rather than having a context *constructor* 🔒$_\mu$ which is semantically left adjoint to the modality, it is assumed that there exists a left division operation $\mu \setminus \_$ left adjoint to $\mu \circ \_$ on modalities, and this operation extends to contexts by applying it to the modal annotation of every variable. In systems based on left division, contexts are lists of modality-annotated types, and substitutions are lists of terms. The difficult question there is not whether substitution is admissible, but whether left division of contexts is functorial. This question has to our knowledge never been properly studied for general mode theories. Moreover, left division of contexts is itself an admissible operation on syntax and, unlike substitution, typically does not have clean denotational semantics.

Fitch-style calculi Logics and type systems that feature typically a single modality $\Box$ and a left adjoint context constructor 🔒, but no modal annotations on variables, are referred to as Fitch-style calculi [12]. Given the presence of only a single modality, Example 2 applies only if there is a non-trivial and non-horizontally-decomposable two-cell between powers of $\Box$, e.g. the duplication $\delta \in \Box \Rightarrow \Box\Box$ of a comonad. Gratzer, Sterling and Birkedal [19] implement type theory with an S4-style $\Box$-modality (i.e. an applicative comonad) and indeed our counterexample applies. They do not prove admissibility of substitution and instead use NbE. Valliappan, Ruch and Cortiñas [30] prove NbE for four modal systems, where $\Box$ is an applicative functor with optionally a co-unit $\epsilon \in \Box \Rightarrow \mathbb{1}$ and/or a duplication δ. Each time, they define a *modal accessibility relation* $\Delta \lhd \Gamma$ on contexts which entails the existence of a substitution $\Gamma \to \Delta$. 🔒 involving only weakening and 2-cells. As such, unlike MTT, their system has a composition-free substitution Γ . 🔒.A . 🔒 $\to \Gamma$. 🔒. Still, they do not claim admissibility of composition of substitution (only identity), nor do they prove admissibility of substitution, instead using NbE. For pointed modalities and monads, on the other hand, we refer back to the lock calculi discussed above, with the later modality and interval quantification as examples.

Acknowledgements

This research is partially funded by the Research Fund KU Leuven. This work was partially supported by a research project of the Research Foundation - Flanders (FWO).

References

1 M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Levy. Explicit substitutions. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, page 31–46, New York, NY, USA, 1989. Association for Computing Machinery. doi:10.1145/96709.96712.

2 Andreas Abel. Polarised subtyping for sized types. *Mathematical Structures in Computer Science*, 18(5):797–822, 2008. doi:10.1017/S0960129508006853.

3 Andreas Abel. *Normalization by evaluation: Dependent types and impredicativity.* Habilitation thesis, Ludwig-Maximilians-Universität München, Germany, 2013.

4 Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. A type- and scope-safe universe of syntaxes with binding: their semantics and proofs. *Journal of Functional Programming*, 31:e22, 2021. doi:10.1017/S0956796820000076.

5 Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Categorical reconstruction of a reduction free normalization proof. In David H. Pitt, David E. Rydeheard, and Peter T. Johnstone, editors, *Category Theory and Computer Science, 6th International Conference, CTCS '95, Cambridge, UK, August 7-11, 1995, Proceedings*, volume 953 of *Lecture Notes in Computer Science*, pages 182–199. Springer, 1995. doi:10.1007/3-540-60164-3_27.

6 Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29. ACM, 2016. doi:10.1145/2837614.2837638.

7 Patrick Bahr, Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. The clocks are ticking: No more delays! In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005097.

8 Jean-Philippe Bernardy, Thierry Coquand, and Guilhem Moulin. A presheaf model of parametric type theory. *Electron. Notes in Theor. Comput. Sci.*, 319:67 – 82, 2015. doi:http://dx.doi.org/10.1016/j.entcs.2015.12.006.

9 Marc Bezem, Thierry Coquand, and Simon Huber. A Model of Type Theory in Cubical Sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26, pages 107–128, Dagstuhl, Germany, 2014. URL: http://drops.dagstuhl.de/opus/volltexte/2014/4628, doi:10.4230/LIPIcs.TYPES.2013.107.

10 Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. Modal dependent type theory and dependent right adjoints. *Mathematical Structures in Computer Science*, 30(2):118–138, 2020. doi:10.1017/S0960129519000197.

11 Joris Ceulemans, Andreas Nuyts, and Dominique Devriese. Sikkel: Multimode simple type theory as an agda library. In Jeremy Gibbons and Max S. New, editors, *Proceedings Ninth Workshop on Mathematically Structured Functional Programming, MSFP@ETAPS 2022, Munich, Germany, 2nd April 2022*, volume 360 of *EPTCS*, pages 93–112, 2022. doi:10.4204/EPTCS.360.5.

12 Ranald Clouston. Fitch-style modal lambda calculi. In Christel Baier and Ugo Dal Lago, editors, *FOSSACS 2018, Held as Part of ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2018. doi:10.1007/978-3-319-89366-2_14.

13 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: A constructive interpretation of the univalence axiom. In Tarmo Uustalu, editor, *21st International Conference on Types for Proofs and Programs, TYPES 2015, May 18-21, 2015, Tallinn, Estonia*, volume 69 of *LIPIcs*, pages 5:1–5:34. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.TYPES.2015.5.

14 Marcelo Fiore and Dmitrij Szamozvancev. Formal metatheory of second-order abstract syntax. *Proc. ACM Program. Lang.*, 6(POPL):1–29, 2022. doi:10.1145/3498715.

15 Daniel Gratzer. Normalization for multimodal type theory. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, 2022. doi:10.1145/3531130.3532398.

16 Daniel Gratzer. *Syntax and semantics of modal type theory.* PhD thesis, Aarhus University, Denmark, 2023. URL: Syntaxandsemanticsofmodaltypetheory.

17 Daniel Gratzer, Alex Kavvos, Andreas Nuyts, and Lars Birkedal. Type theory à la mode. Pre-print, 2020. URL: `https://lirias.kuleuven.be/retrieve/635295`.

18 Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal Dependent Type Theory. *Logical Methods in Computer Science*, Volume 17, Issue 3, July 2021. URL: `https://lmcs.episciences.org/7713`, `doi:10.46298/lmcs-17(3:11)2021`.

19 Daniel Gratzer, Jonathan Sterling, and Lars Birkedal. Implementing a modal dependent type theory. *Proc. ACM Program. Lang.*, pages 107:1–107:29, 2019. `doi:10.1145/3341711`.

20 Per Martin-Löf. Intuitionistic type theory. In *Studies in proof theory*. Bibliopolis, 1984.

21 Conor McBride. Type-preserving renaming and substitution. Unpublished note, 2005. URL: `http://strictlypositive.org/ren-sub.pdf`.

22 Andreas Nuyts. Contextual algebraic theories: Generic boilerplate beyond abstraction (extended abstract). In *Workshop on Type-Driven Development (TyDe)*, 9 2022. URL: `https://anuyts.github.io/files/cmat-tyde22-abstract.pdf`.

23 Andreas Nuyts. A lock calculus for multimode type theory. In *29th International Conference on Types for Proofs and Programs (TYPES)*, 6 2023. URL: `https://lirias.kuleuven.be/retrieve/720873`.

24 Andreas Nuyts and Dominique Devriese. Degrees of relatedness: A unified framework for parametricity, irrelevance, ad hoc polymorphism, intersections, unions and algebra in dependent type theory. In *Logic in Computer Science (LICS) 2018, Oxford, UK, July 09-12, 2018*, pages 779–788, 2018. `doi:10.1145/3209108.3209119`.

25 Andreas Nuyts, Andrea Vezzosi, and Dominique Devriese. Parametric quantifiers for dependent type theory. *PACMPL*, 1(ICFP):32:1–32:29, 2017. URL: `http://doi.acm.org/10.1145/3110276`, `doi:10.1145/3110276`.

26 Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *LICS '01*, pages 221–230, 2001. `doi:10.1109/LICS.2001.932499`.

27 Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001. `doi:10.1017/S0960129501003322`.

28 Philipp Stassen, Daniel Gratzer, and Lars Birkedal. {mitten}: A Flexible Multimodal Proof Assistant. In Delia Kesner and Pierre-Marie Pédrot, editors, *28th International Conference on Types for Proofs and Programs (TYPES 2022)*, volume 269 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:23, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TYPES.2022.6`, `doi:10.4230/LIPIcs.TYPES.2022.6`.

29 Jonathan Sterling. *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory*. PhD thesis, Carnegie Mellon University, USA, 2022. URL: `https://doi.org/10.1184/r1/19632681.v1`, `doi:10.1184/R1/19632681.V1`.

30 Nachiappan Valliappan, Fabian Ruch, and Carlos Tomé Cortiñas. Normalization for fitch-style modal calculi. 6(ICFP), aug 2022. `doi:10.1145/3547649`.

31 Niccolò Veltri and Andrea Vezzosi. Formalizing π-calculus in guarded cubical agda. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 270–283. ACM, 2020. `doi:10.1145/3372885.3373814`.