Higher-order term indexing using substitution trees

BRIGITTE PIENTKA
McGill University

We present a higher-order term indexing strategy based on substitution trees for simply typed lambda-terms. The strategy is based in linear higher-order patterns where computationally expensive parts are delayed. While insertion of terms into the index is based on computing the most specific linear generalization of two linear higher-order patterns, retrieval is based on matching two linear higher-order patterns. We give a theoretical framework for higher-order term indexing, describe insertion and retrieval algorithms and prove their correctness. This indexing structure is implemented as part of the Twelf system to speed-up the execution of the tabled higher-logic programming interpreter.

Categories and Subject Descriptors: F.4.1 [**Theory of Computation**]: Mathematical Logic and Formal Languages; D.3.3 [**Software**]: Language Constructs and Features—*Frameworks*

General Terms: Design, Theory

Additional Key Words and Phrases: Indexing, type theory, logical frameworks

1. INTRODUCTION

First-order logic programming and theorem proving systems have developed into highly sophisticated automated reasoning systems with remarkable performance over the last decade. This success is to a large extent due to term indexing techniques which allow these systems to manage and use redundancy elimination techniques. In general, term indexing is concerned with compactly storing a large collection of terms and rapidly retrieving a set of candidate terms satisfying some property (e.g. unifiability, instance, variant etc.) from a large collection of terms.

There are many examples where term indexing is used. In logic programming, for example, we need to select all clauses from the program whose head unifies with the current goal. In tabled logic programming we memoize intermediate goals in a table and reuse their results later in order to eliminate redundant and infinite computation. Here we need to find all entries in the table such that the current goal is a variant or an instance of a table entry and re-use the associated answers. Similarly in theorem proving, we keep track of previously derived formulas to eliminate redundancy and detect loops. Since rapid retrieval and efficient storage of large collection of terms plays a central role in logic programming and in proof search in general, a variety of indexing techniques have been proposed for first-order terms (see [Ramakrishnan et al. 2001] for a survey). However, indexing techniques for

This material is based upon work supported by ....

higher-order terms, i.e. terms which may contain lambda-abstraction, are largely missing thereby severely hampering the performance of higher-order systems and limiting their potential applications. There are mainly two problems in adapting first-order indexing techniques. First many operations used in building an efficient term index and retrieving a set of candidate terms from a large collection are undecidable in general for higher-order terms. Second, the scoping of variables and binders in the higher-order case presents challenges.

In this paper, we present a higher-order term indexing technique based on substitution trees. Substitution tree indexing [Graf 1995] is a highly successful first-order term indexing strategy which allows the sharing of common sub-expressions via substitutions. We extend this idea to the higher-order setting and present an indexing technique for higher-order terms, i.e. terms which may contain lambda-abstractions. The challenge in the higher-order setting is that many common operations on higher-order terms which are necessary to build and maintain substitution trees or retrieve elements from the index are undecidable in general. For example, to build a substitution tree, we compute the most specific common generalization between two terms. However, in general the most specific generalization of two terms does not exist in the higher-order setting. Similarly, retrieving all terms, which unify or match, needs to be efficient – but higher-order unification is undecidable in general. Fortunately, there exists a fragment called higher-order patterns for which checking unifiability of two terms and computing the most specific generalization between two terms is decidable [Miller 1991b; Pfenning 1991]. However, even for this fragment algorithms may not be efficient in practice [Pientka and Pfennning 2003] and are sufficiently complex that it is not obvious that they are a suitable basis for higher-order term indexing techniques. In this paper we consider an even stricter class of lambda-terms, called linear higher-order patterns which refines the notion of higher-order patterns further and factor out any computationally expensive parts. As we have shown in [Pientka and Pfennning 2003] many terms encountered fall into this fragment and linear higher-order pattern unification performs well in practice. In this paper, we demonstrate that linear higher-order patterns are well suited to elegantly describe term indexing operations such as computing the most specific linear generalization or checking unifiability of two terms. Moreover, we give algorithms for inserting linear higher-order patterns into an index and for retrieving a set of terms from the index such that the query is an instance of the term in the index and prove the correctness of these operations. Although we concentrate on the simply typed terms in this paper, the presented techniques can be generalized to the dependently typed setting (see [Pientka 2003b]) and are in fact implemented as part of the logical framework Twelf system [Pfenning and Schürmann 1999]. We have used higher-order substitution trees to speed-up the execution of the tabled logic programming interpreter [Pientka 2002] and to facilitate the generation of small proof witnesses [Sarkar et al. 2005]. Preliminary results have been published in [Pientka 2003a], and this paper expands the theoretical results.

The paper is organized as follows: In Section 2, we present the general idea of higher-order substitution trees. In Section 3 we give the theoretical background. In Section 5, we give algorithms for computing the most specific linear generalization of two terms and inserting terms into the index. Retrieval is discussed in Section

6. We conclude with summarizing the results and related work.

2. HIGHER-ORDER SUBSTITUTION TREES

We illustrate the general idea of substitution tree indexing using a first-order example and then focus on indexing of higher-order terms. In particular, we highlight some of the subtle issues concerning the interplay of bound and meta-variables

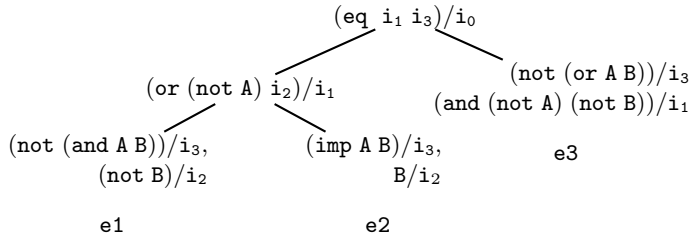
Example 1. To illustrate the basic idea consider the example of equality transformations for propositional logic, described by $A \Leftrightarrow B$. In the logical framework Twelf [Pfenning and Schürmann 1999], we first declare constructors for propositions such as conjunction, implication, disjunction and negation as follows.

```
prop:  type.
and:   prop -> prop -> prop.        or:   prop -> prop -> prop.
imp:   prop -> prop -> prop.        neg:  prop -> prop.
```

Next, we present some standard equivalence preserving transformations on propositions together with their encoding in the logical framework Twelf.

$A \Leftrightarrow B$	eq : prop → prop → type.
$e_1 : \neg(A \wedge B) \Leftrightarrow (\neg A) \vee (\neg B)$.	e1 : eq (not (and A B)) (or (not A) (not B)).
$e_2 : (A \supset B) \Leftrightarrow (\neg A) \vee B$.	e2 : eq (imp A B) (or (not A) B).
$e_3 : (\neg(A \vee B) \Leftrightarrow ((\neg A) \wedge (\neg B))$.	e3 : eq (not (or A B) (and (not A) (not B)).

First, we define a type family `eq` which represents the judgment for equivalence preserving transformation between two propositions. Next, we represent each equivalence transformation. As we see, the three transformations share quite a lot of structure. For example, `e1` and `e2` share the same structure in the second argument, namely (`or` (`not` `A`) □) where □ denotes a whole in the term. Our intention is to share common structure of terms in order to share common operations. For example when checking whether a term U is already in the index, we only want to compare once against the skeleton (`or` (`not` `A`) □). To achieve this, we compute the most specific generalization between the given terms. For example. the most specific generalization of the first and second clause stated is, `eq` $\mathtt{i}_1$ (`or` (`not` `A`) $\mathtt{i}_2$) where we can obtain the clause `e1` by instantiating $\mathtt{i}_1$ with (`not` (`and` `A` `B`)) and $\mathtt{i}_2$ with (`not` `B`). Similarly, we can obtain the clause `e2` by instantiating $\mathtt{i}_1$ with (`imp` `A` `B`) and $\mathtt{i}_2$ with `B`. $\mathtt{i}_0$,$\mathtt{i}_1$, $\mathtt{i}_2, \ldots$ denote meta-variables which represent wholes in terms. A term can be represented as a sequence of substitutions. For example, the clause `e1` can be described as $[\![(\mathtt{not}\ (\mathtt{and}\ \mathtt{A}\ \mathtt{B}))/\mathtt{i}_1, (\mathtt{not}\ \mathtt{B})/\mathtt{i}_2]\!](\mathtt{eq}\ \ \mathtt{i}_1\ (\mathtt{or}\ (\mathtt{not}\ \mathtt{A})\ \mathtt{i}_2))$. A substitution tree is a tree where each node contains a set of substitutions. One possible substitution tree which allows sharing of this sub-structure for the three clauses is given below. The original clause can be obtained by composing all the substitutions along one branch. To easily identify, which branch corresponds to which clause, we labeled the leafs with the name of the clause.



Example 2. There are also well-know equivalence preserving transformation in first-order logic and we give some involving universal quantification below.

$$
\begin{array}{l}
e_5 : (\forall x.A(x)) \wedge\ B \ \Leftrightarrow\ \forall x.A(x) \wedge\ B \\
e_6 : A \wedge\ (\forall x.B(x)) \ \Leftrightarrow\ \forall x.A \wedge\ B(x) \\
e_7 : A \supset (\forall x.B(x)) \ \Leftrightarrow\ \forall x.A \supset B(x)
\end{array}
$$

Of course these stated equality preserving transformations are only valid, if certain bound variable conditions are satisfied. For example, the first translation requires that the bound variable x does not occur in B. These conditions are naturally enforced using higher-order abstract syntax where bound variables in the object language are represented by bound variables in the meta-language. We first define the constructor `forall` with type $(\mathtt{i} \to \mathtt{prop}) \to \mathtt{prop}$. The universal quantifier $\forall x.A(x)$ is then encoded as `forall` λ`x.A x` and bound variables are represented by the λ-binder. The representation of the equivalence preserving transformations is given below.

```
e5 : eq (and  (forall λx.A x)        B        ) (forall λx.(and  (A x)   B)).
e6 : eq (and        A        (forall λx.B x) ) (forall λx.(and   A   (B x))).
e7 : eq (imp        A        (forall λx.B x) ) (forall λx.(imp   A   (B x))).
```

In the higher-order setting, meta-variables denote a *closed* instance of terms. `A` in the first translation denotes a function which is applied to `x`, `B` however denotes an atomic proposition, and hence cannot depend on the bound variable `x`. As this example illustrates, bound variable dependencies are naturally and elegantly encoded in this higher-order setting. To highlight the common structure between the three transformation, we have spaced each line.

Inspecting the four given clauses closely, we observe that they share a lot of structure. For example clause `e5` and `e6` "almost" agree on the second argument. Our goal is to share common structure of terms in order to share common operations even below a binder. This means for example that we would like to share the term `forall` λ`x.`$\Box$ where $\Box$ is once instantiated with `and (A x) B` to obtain the terms `e5` and `e6` and secondly instantiated with `(imp A (B x))` to obtain `e7`. Note that $\Box$ is instantiated with open terms which are allowed to refer to the bound variable `x`. Our indexing structure supports such sharing of expressions even in the presence of binders and allows instantiations with open terms, i.e. terms which may contain bound variables. This is unlike the first-order case where holes were always instantiated with closed terms. To achieve this, we characterize holes as a closure of a meta-variable `i` together with a delayed substitution. This substitution precisely characterizes the dependencies we allow when instantiating the meta-variable with

an open term. For example, forall λx.□ is denoted by forall λx.i[x/y] where i is a meta-variable and [x/y] is a post-poned substitution. When we instantiate the meta-variable i with and (A y) B we will apply the substitution [x/y] which essentially renames the variable y to x, and yields as a final result forall λx.and (A x) B.

Associating meta-variables with a postponed substitution is a known technique from explicit substitution calculus. Instead of using the explicit substitution calculus based in de Bruijn indices [Abadi et al. 1990; Dowek et al. 1995], we use the contextual modal type theory [Nanevski et al. 2006] as a foundation which provides a high-level explanation of meta-variables. Characterizing holes in terms as a closure of meta-variable and a post-poned substitution will allow us to instantiate holes using first-order replacement.

To insert these three clauses into a substitution tree, we need to compute the most specific common generalization. To do this in a simple manner, we first translate them into linear higher-order patterns[Pientka and Pfennning 2003]. Linear higher-order patterns refine the notion of higher-order patterns [Miller 1991b; Pfenning 1991] where every meta-variable must be applied to *some* distinct bound variables in two ways: First, linear higher-order patterns require that every meta-variable occurs only once and in addition every meta-variable is applied to *all* distinct bound variables in its context. This observation to restrict higher-order patterns even further to patterns where meta-variables must be applied to all bound variables has also been made by Hanus and Prehofer [Hanus and Prehofer 1999] in the context of higher-order functional logic programming. While Hanus and Prehofer syntactically disallow terms which are not fully applied, we translate any term into a linear higher-order pattern together with some variable definitions. Maintaining these two conditions yield a simple algorithm and allows us to delay the occurs check and any other complicated conditions involving bound variable occurrences.

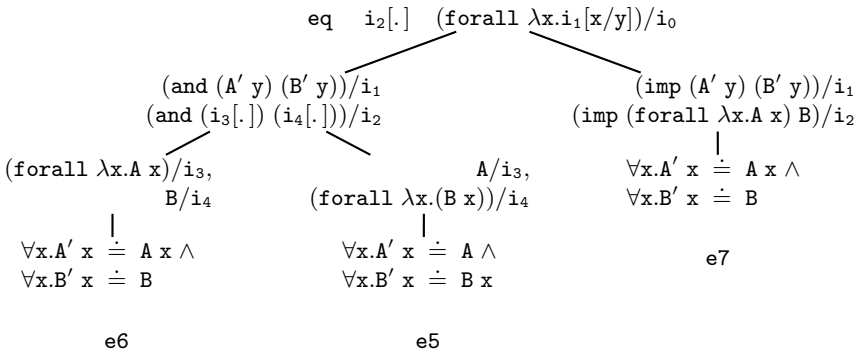
As we observe, none of the above clauses fulfills this stringent condition. For example, in eq (and (forall λx.A x) B) (forall λx.(and (A x) B) the meta-variable B does not depend on the bound variable x in (forall λx.(imp (A x) B)) although it occurs within the scope of the binder x. Hence, B is not a linear higher-order pattern, since it is not applied to all bound variables in whose scope it occurs. In addition, the meta-variable A occurs twice. Before inserting the clauses into a substitution tree, we therefore first linearize terms by eliminating any duplicate occurrences of meta-variables, and replacing any meta-variable which is not fully applied with one which is. The program after linearization is shown next:

$$\texttt{e5} : \texttt{eq}\ (\texttt{and}\ (\texttt{forall}\ \lambda\texttt{x.A x})\ \texttt{B})\quad (\texttt{forall}\ \lambda\texttt{x.and}\ (\texttt{A}'\ \texttt{x})\ (\texttt{B}'\ \texttt{x})).$$
$$\forall\texttt{x}.(\texttt{A}'\ \texttt{x}) \doteq (\texttt{A x}) \wedge \texttt{B}'\ \texttt{x} \doteq \texttt{B}$$
$$\texttt{e6} : \texttt{eq}\ (\texttt{and A}\ (\texttt{forall}\ \lambda\texttt{x.B x}))\quad (\texttt{forall}\ \lambda\texttt{x.and}\ (\texttt{A}'\ \texttt{x})\ (\texttt{B}'\texttt{x})).$$
$$\forall\texttt{x}.(\texttt{A}'\ \texttt{x}) \doteq \texttt{A} \wedge \texttt{B}'\ \texttt{x} \doteq (\texttt{B x})$$
$$\texttt{e7} : \texttt{eq}\ (\texttt{imp A}\ (\texttt{forall}\ \lambda\texttt{x.B x}))\quad (\texttt{forall}\ \lambda\texttt{x.}(\texttt{imp}\ (\texttt{A}'\ \texttt{x})\ (\texttt{B}'\ \texttt{x}))).$$
$$\forall\texttt{x}.(\texttt{A}'\ \texttt{x}) \doteq \texttt{A} \wedge \texttt{B}'\ \texttt{x} \doteq (\texttt{B x})$$

We view linearization as a standardization step, which is also in a simpler form present in first-order indexing techniques. In the first-order setting, terms are linearized and duplicate occurrences of meta-variables are factored out in order to postpone the occurs check. Our notion of liner higher-order patterns establishes a

criteria with the same intentions of factoring out expensive operations in the higher-order setting. Together with the linear term, we simply store variable definitions, which establish the equality between these two meta-variables.

Now even more sharing becomes apparent. For example, the clauses `e5` and `e6` agree upon the last argument. We now compute the most specific generalization between these clauses, and can build up a substitution tree. Each node in the substitution tree contains a set of substitutions, and variable definitions which resulted from linearizing terms are found at the leafs.



3. BACKGROUND

3.1 Contextual modal type theory

In this section we briefly introduce a typed lambda-calculus with first-class meta-variables which allows the instantiation of meta-variables with open terms. Previously, we have used the contextual modal type theory as a foundation for describing linear higher-order pattern unification for example [Pientka and Pfennning 2003; Pientka 2003b].

Normal Kinds	K	::=	$\mathsf{type} \mid A \to K$
Atomic Types	P, Q	::=	$a \cdot S$
Normal Types	A, B, C	::=	$P \mid A \to B$
Atomic Objects	R	::=	$H \cdot S \mid u[\sigma]$
Normal Objects	M, N	::=	$\lambda x.M \mid R$
Head	H	::=	$x \mid c$
Spines	S	::=	$\mathsf{nil} \mid M; S$
Contexts	Γ, Ψ	::=	$\cdot \mid \Gamma, x{:}A$
Substitutions	σ	::=	$. \mid \sigma, N/x \mid \sigma, R/\!\!/x$
Modal Contexts	Δ	::=	$\cdot \mid \Delta, u{::}P[\Psi]$
Signatures	Σ	::=	$\cdot \mid \Sigma, a{:}K \mid \Sigma, c{:}A$

We base our description of the simply typed lambda-calculus on a variant of the contextual modal type theory as presented in [Nanevski et al. 2006]. Our presentation uses a spine notation [Iliano Cervesato 2003] which makes it more natural and elegant to present algorithms and we enforce terms to be in normal form by exploiting a presentation technique due to Watkins et al. [Watkins et al.

2002]. We follow a bi-directional type checking approach. In order to achieve this we divide the term calculus into *atomic objects* R and *normal objects* M. We start by defining a simply typed version of logical frameworks.

Contexts Γ and Ψ contain only declarations $x{:}A$ where A is normal, all terms occurring in substitutions σ are either normal (in N/x) or atomic (in $R/\!\!/x$), and so on. Finally, while the syntax only guarantees that terms N are normal (that is, contain no β-redexes), the typing rules will in addition guarantee that all well-typed terms are fully η-expanded. The modal context Δ contains declaration of meta-variables $u{::}P[\Psi]$. We enforce that all meta-variables occurring in well-typed terms must be of atomic type, i.e. they are lowered. This can always be achieved.

Signatures declare global constants and never change in the course of a typing derivation. We therefore suppress the signatures throughout. Typing at the level of objects is divided into three judgments:

$\Delta;\Gamma \vdash M \Leftarrow A$	Check normal object M against canonical A
$\Delta;\Gamma \vdash R \Rightarrow P$	Synthesize atomic P for atomic object R
$\Delta;\Gamma \vdash S > A \Rightarrow P$	Synthesize atomic P for spine S and canonical A
$\Delta;\Gamma \vdash \sigma \Leftarrow \Psi$	Check σ against Ψ

We always assume that Δ and Γ and the subject (M, R, or σ) are given, and that the contexts Δ and Γ contain only canonical types. For synthesis $R \Rightarrow P$ we assume R is given and we generate an atomic type P. Similar judgments hold for kinds which we omit here.

Normal objects

$$\frac{\Delta;\Gamma, x{:}A \vdash M \Leftarrow B}{\Delta;\Gamma \vdash \lambda x.M \Leftarrow A \to B}\ \mathsf{Lam} \qquad \frac{\Delta;\Gamma \vdash R \Rightarrow P' \quad P' = P}{\Delta;\Gamma \vdash R \Leftarrow P}\ \Rightarrow\!\Leftarrow$$

Atomic objects

$$\frac{\Delta;\Gamma, x{:}A, \Gamma' \vdash S > A \Rightarrow P}{\Delta;\Gamma, x{:}A, \Gamma' \vdash x \cdot S \Rightarrow P}\ \mathsf{var} \qquad \frac{c{:}A \in \Sigma \quad \Delta;\Gamma \vdash S > A \Rightarrow P}{\Delta;\Gamma \vdash c \cdot S \Rightarrow P}\ \mathsf{con}$$

$$\frac{\Delta, u{::}P[\Psi], \Delta';\Gamma \vdash \sigma \Leftarrow \Psi}{\Delta, u{::}P[\Psi], \Delta';\Gamma \vdash u[\sigma] \Rightarrow P}\ \mathsf{mvar}$$

Spines

$$\frac{\Delta;\Gamma \vdash S > B \Rightarrow P \quad \Delta;\Gamma \vdash M \Leftarrow A}{\Delta;\Gamma \vdash M\ ;\ S > A \to B \Rightarrow P}\ \mathsf{Scons} \qquad \frac{}{\Delta;\Gamma \vdash \mathsf{nil} > P \Rightarrow P}\ \mathsf{Snil}$$

In general, introduction forms for a type constructor break down a type when read from the conclusion to the premise. This means if the type in the conclusion is given, we can extract the type for the premise, and therefore introduction forms should be checked against a type. Conversely, elimination forms break down the type when read from premise to conclusion. This means if the type in the premise can be synthesized, we can extract the component type for the conclusion, and therefore elimination forms should synthesize their type.

When checking a normal object that happens to be atomic (that is, has the form R) against a type P we have to synthesize the type for R and compare it with P. Since all synthesized types are canonical, this comparison is simple syntactic equality for simple types.

The rules for ordinary variables and constants are as usual. For meta-variables we need to be careful about directions and dependencies. We will enforce that meta-variables are lowered, i.e. they must be of atomic type P. While $u[\sigma]$ synthesizes a type, we need the type of u, namely $P[\Psi]$ so we can check σ against Ψ. Some renaming is left implicit here, as the variables in the domain of σ should match the variables declared in Ψ. Moreover, we need to apply σ to transport A from Ψ (upon which it may depend) to Γ.

THEOREM 3.1 DECIDABILITY OF TYPE CHECKING. *All judgments in the simply typed contextual modal type theory are decidable.*

PROOF. The typing judgments are syntax-directed and therefore clearly decidable. □

Since our description of substitution trees relies on a concise notion of substitution, we carefully define ordinary substitution for ordinary variables and contextual substitutions for meta-variables.

3.2 Substitution on Terms

In this section we start with defining the operations of substitution on terms. The substitution function we need must construct canonical terms, since those are the only ones that are well-formed and the only ones of interest. Hence, in places where the ordinary substitution operation would create a redex, in particular when applying the substitution $[M/x]$ to a term $x \cdot S$, we must apply the substitution $[M/x]$ to the spine S, but we also must reduce the redex $(M \cdot [M/x]S)$ which would be created. Since when applying $[M/x]$ to the spine S, we again may encounter situations which require us to contract a redex, the substitution $[M/x]$ must be hereditary. We therefore call this operation *hereditary substitution*.

This technique is due to Watkins *et. al.* where it has been used to describe canonical version of logical framework. Here we demonstrate that this technique is in fact very general and even useful in the simply typed setting. The main difficulty in defining hereditary substitutions is that this operation could easily fail to terminate. Consider for example the term which arises when computing the normal form of $(\lambda y.y\ y)\ (\lambda x.x\ x)$. Clearly, on well-typed terms this should not occur.

We define hereditary substitutions as a primitive recursive functional where we pass in the type of the variable we substitute for. This will be crucial in determining termination of the overall substitution operation. If we herditarily substitute $[\lambda y.M/x](x \cdot S)$, then if everything is well-typed, $x : A_1 \to A_2$ for some A_1 and A_2 and we will write $[\lambda y.M/x]_{A_1 \to A_2}(x \cdot S)$ indexing the substitution with the type for x. These will all be total operations since any side condition can be satisfied by α-conversion. First, we present the ordinary capture-avoiding substitution for a single variable, $[M/x]_A N$, $[M/x]_A S$, and$[M/x]_A \sigma$.

$$
\begin{array}{lcll}
[M/x]_A(\lambda y.N) & = & \lambda y.N' & \text{where } N' = [M/x]_A N \\
 & & & \text{choosing } y \notin \mathsf{FV}(M) \text{ and } y \neq x \\
 & & & \\
[M/x]_A(u[\sigma]) & = & u[\sigma'] & \text{where } \sigma' = [M/x]_A \sigma \\
[M/x]_A(c \cdot S) & = & c \cdot S' & \text{where } S' = [M/x]_A S \\
[M/x]_A(x \cdot S) & = & \mathsf{reduce}(M : A, S') & \text{where } S' = [M/x]_A S \\
[M/x]_A(y \cdot S) & = & y \cdot S' & \text{if } y \neq x \text{ and } S' = [M/x]_A S \\
 & & & \\
[M/x]_A(\mathsf{nil}) & = & \mathsf{nil} & \\
[M/x]_A(N; S) & = & N'; S' & \text{where } N' = [M/x]_A N \text{ and } S' = [M/x]_A S \\
 & & & \\
[M/x]_A(\cdot) & = & \cdot & \\
[M/x]_A(\sigma, N/y) & = & (\sigma', N'/y) & \text{where } \sigma' = [M/x]_A \sigma \text{ and } N' = [M/x]_A N \\
[M/x]_A(\sigma, R/\!\!/y) & = & (\sigma', R'/\!\!/y) & \text{where } \sigma' = [M/x]_A \sigma \text{ and } R' = [M/x]_A R \\
[M/x]_A(\sigma, R/\!\!/y) & = & (\sigma', N'/y) & \text{where } \sigma' = [M/x]_A \sigma \text{ and } N' = [M/x]_A R
\end{array}
$$

Inductive definition for substituting an atomic term R for a variable x is straightforward.

$$
\begin{array}{lcll}
\mathsf{reduce}(\lambda y.M : A_1 \to A_2, (N; S)) & = & M'' & \text{where } [N/y]_{A_1} M = M' \\
 & & & \text{and } \mathsf{reduce}(M' : A_2, S) = M'' \\
\mathsf{reduce}(R : P, \mathsf{nil}) & = & R & \\
\mathsf{reduce}(M : A, S) & & \text{fails} & \text{otherwise}
\end{array}
$$

Substitution may fail to be defined only if substitutions into the subterms are undefined. The side conditions $y \notin \mathsf{FV}(M)$ and $y \neq x$ do not cause failure, because they can always be satisfied by appropriately renaming y. However, substitution may be undefined if we try for example to substitute an atomic term R for x in the term $x \cdot S$ where the spine S is non-empty. Similarly, the reduce operation is undefined. The substitution operation is well-founded since recursive appeals to the substitution operation take place on smaller terms with equal type A, or the substitution operates on smaller types (see the case for $\mathsf{reduce}(\lambda y.M : A_1 \to A_2, (N; S))$).

The first property states that the hereditary substitution operations terminate, independently of whether the terms involved are well-typed or not. The operation may fail, in particular if we have ill-typed terms, or yield a canonical term as a result.

THEOREM 3.2 TERMINATION. *$[M/x]_A(N)$, $[M/x]_A R$, $[M/x]_A \sigma$, $\mathsf{reduce}(M : A, S)$ terminates, either by returning a result or failing after a finite number of steps.*

PROOF. This can be verified by a nested induction, first on the structure of A, and second on the structure of the term we apply hereditary substitution to or the term S we apply to $M : A$ in the case for reduce. □

THEOREM 3.3 SUBSTITUTION ON TERMS.

(1) *If $\Delta; \Gamma \vdash M \Leftarrow A$ and $\Delta; \Gamma, x{:}A, \Gamma' \vdash N \Leftarrow C$ and $[M/x]_A N = N'$ then $\Delta; \Gamma, \Gamma' \vdash N' \Leftarrow C$.*

(2) *If* $\Delta;\Gamma \vdash M \Leftarrow A$ *and* $\Delta;\Gamma, x{:}A, \Gamma' \vdash R \Rightarrow P$ *and* $R' = [M/x]_A R$
then $\Delta;\Gamma,\Gamma' \vdash R' \Rightarrow P$.

(3) *If* $\Delta;\Gamma \vdash M \Leftarrow A$ *and* $\Delta;\Gamma, x{:}A, \Gamma' \vdash S > B \Rightarrow P$ *and* $S' = [M/x]_A S$
then $\Delta;\Gamma,\Gamma' \vdash S' > B \Rightarrow P$.

(4) *If* $\Delta;\Gamma \vdash M \Leftarrow A$ *and* $\Delta;\Gamma \vdash S > A \Rightarrow P$
then $\mathsf{reduce}(M : A, S) = R$ *and* $\Delta;\Gamma \vdash R \Rightarrow P'$ *and* $P' = P$.

(5) *If* $\Delta;\Gamma \vdash M \Leftarrow A$ *and* $\Delta;\Gamma, x{:}A, \Gamma' \vdash \sigma \Leftarrow \Psi$ *and* $\sigma' = [M/x]_A \sigma$
then $\Delta;\Gamma,\Gamma' \vdash \sigma' \Leftarrow \Psi$.

Proof. By simultaneous induction on the definition of substitution, structure of the type A occurring in the type annotation of the substitution $[M/x]_A$ or $\mathsf{reduce}(M : A, S)$ and the second derivation. Either we apply the substitution to a smaller term, or the type A is decreasing or the second derivation is decreasing. □

3.3 Simultaneous Substitutions

The ideas underlying the definition of substitutions in the previous section can be extended to capture simultaneous substitutions. The substitution is again hereditary.

$$\frac{}{\Delta;\Gamma \vdash (\cdot) \Leftarrow (\cdot)} \qquad \frac{\Delta;\Gamma \vdash \sigma \Leftarrow \Psi \quad \Delta;\Gamma \vdash M \Leftarrow A}{\Delta;\Gamma \vdash (\sigma, M/x) \Leftarrow (\Psi, x{:}A)}$$

$$\frac{\Delta;\Gamma \vdash \sigma \Leftarrow \Psi \quad \Delta;\Gamma \vdash R \Rightarrow A' \quad A' = A}{\Delta;\Gamma \vdash (\sigma, R/\!\!/x) \Leftarrow (\Psi, x{:}A)}$$

Besides M/x for canonical M, there is a second way to construct a substitution to replace a variable by an atomic term R, written $R/\!\!/x$. This is justified from the nature of hypothetical judgments, since an assumption $x{:}A$ represents $x \Rightarrow A$ so we can substitute R for x if $R \Rightarrow A$[1].

Substitutions $R/\!\!/x$ are necessary so that we can extend a given substitution with $x/\!\!/x$ when traversing a binding operator in a type-free way. We could not extend substitutions with x/x, since x is not a canonical term unless it is of atomic type. Identity substitutions can now have the form $x_1/\!\!/x_1, \ldots, x_n/\!\!/x_n$.

Next we define simultaneous substitution $[\sigma]M$ and $[\sigma]\tau$. It is only total when the substitution σ is defined on all free variables in M and τ, respectively. This will be satisfied, because simultaneous substitution is only applied when the assumptions of the theorem following this definition are satisfied. Simultaneous substitutions commute with the term constructors, as one would expect. Just as we annotated the substitution $[M/x]_A$ with the type of the variable x, we will annotate the simultaneous substitution σ with an approximation ψ of its domain Ψ where in fact the type for variables which will be replaced by atomic terms can be omitted. The intuition is that simultaneous substitution may contain substitutions such as $x/\!\!/x$ and the type of x may not always be available to extend the context annotation (see the case for $[\sigma]_\psi(\lambda y.M)$).

[1] In fact, R must be of atomic type.

$$
\begin{array}{lcll}
[\sigma]_\psi(\lambda y.N) & = & \lambda y.N' & \text{where } N' = [\sigma, y/\!\!/y]_{\psi,y:\_}(N) \\
 & & & \text{choosing } y \notin \mathsf{FV}(\sigma), \mathsf{dom}(\sigma) \\
[\sigma]_\psi(c \cdot S) & = & c \cdot S' & \text{where } [\sigma]_\psi(S) = S' \\
[\sigma]_\psi(x \cdot S) & = & R' & \text{where } [\sigma]_\psi(S) = S',\ M/x \in \sigma \text{ and } x{:}A \in \Psi, \\
 & & & \text{and } R' = \mathsf{reduce}(M : A, S') \\
[\sigma]_\psi(x \cdot \mathsf{nil}) & = & R & \text{where } R/\!\!/x \in \sigma \text{ and } x{:}\_ \in \Psi \text{ or } x{:}A \in \Psi \\
[\sigma]_\psi(v[\tau]) & = & v[\tau'] & \text{where } \tau' = [\sigma]_\psi(\tau) \\
[\sigma]_\psi(R) & & \text{fails} & \text{otherwise} \\
[\sigma]_\psi(\cdot) & = & \cdot & \\
[\sigma]_\psi(\tau, N/y) & = & (\tau', N'/y) & \text{where } \tau' = [\sigma]_\psi(\tau) \text{ and } N' = [\sigma]_\psi(N) \\
[\sigma]_\psi(\tau, R/\!\!/y) & = & (\tau', R'/\!\!/y) & \text{where } \tau' = [\sigma]_\psi(\tau) \text{ and } [\sigma]_\psi(R) = R' \\
[\sigma]_\psi(\tau, R/\!\!/y) & = & \tau', M'/y & \text{if } [\sigma]_\psi(R) = M' : \alpha' \text{ with } \tau' = [\sigma]_\psi(\tau) \\
[\sigma]_\psi(\tau) & & \text{fails} & \text{otherwise}
\end{array}
$$

The definition of simultaneous substitutions is a straightforward extension of the ordinary substitution described earlier. The only difficulty is that we sometimes need to rename the domain of a substitution to match a given context. When $\sigma = (M_1/x_1, \ldots, M_n/x_n)$ and $\Psi = (y_1{:}A_1, \ldots, y_n{:}A_n)$ then we will rename the domain of the substitution σ by writing $\sigma/\Psi = (M_1/y_1, \ldots, M_n/y_n)$.

Simultaneous substitutions satisfy the simultaneous substitution principle, annotated with proof terms. The second property amounts to composition of the substitutions τ and σ.

Theorem 3.4 Simultaneous Substitution on Terms.

(1) *If* $\Delta; \Gamma \vdash \sigma \Leftarrow \Psi$ *and* $\Delta; \Psi \vdash N \Rightarrow C$ *and* $[\sigma]_\psi N = N'$ *then* $\Delta; \Gamma \vdash N' \Rightarrow C$.

(2) *If* $\Delta; \Gamma \vdash \sigma \Leftarrow \Psi$ *and* $\Delta; \Psi \vdash R \Leftarrow P$ *and* $[\sigma]_\psi R = R'$ *then* $\Delta; \Gamma \vdash R \Leftarrow P$.

(3) *If* $\Delta; \Gamma \vdash \sigma \Leftarrow \Psi$ *and* $\Delta; \Psi \vdash S > A \Rightarrow P$ *and* $[\sigma]_\psi S = S'$ *then* $\Delta; \Gamma \vdash S' > A \Rightarrow P$.

(4) *If* $\Delta; \Gamma \vdash \sigma \Leftarrow \Psi$ *and* $\Delta; \Psi \vdash \tau \Leftarrow \Theta$ *then* $\Delta; \Gamma \vdash [\sigma]_\psi \tau \Leftarrow \Theta$.

Proof. By induction on the structure of the second given derivation. □

Finally, we remark that composition of hereditary substitution is written as $[\sigma]_\psi \tau$, and the standard composition principles hold (see [Nanevski et al. 2006]).

Finally, we emphasize that substitutions σ are defined only on ordinary variables x and not modal variables u. We write id_Γ for the identity substitution $(x_1/\!\!/x_1, \ldots, x_n/\!\!/x_n)$ for a context $\Gamma = (\cdot, x_1{:}A_1, \ldots, x_n{:}A_n)$.

3.4 Contextual substitution

Meta-variables $u[\sigma]$ give rise to new contextual substitutions, which are only slightly more difficult than ordinary substitutions. To understand contextual substitutions, we take a closer look at the closure $u[\sigma]$ which describes the meta-variable. Recall that the substitution σ which is associated with every meta-variable u stands for a postponed substitution. As a consequence, we can apply σ as soon as we know which term u should stand for. Moreover, we require that meta-variables have atomic type P and hence, we will only substitute atomic objects for meta-variables.

Finally because of α-conversion, the variables that are substituted at different occurrences of u may be different. As a result, substitution for a meta-variable must carry a context, written as $[\hat{\Psi}.R/u]N$ and $[\hat{\Psi}.R/u]\sigma$ where $\hat{\Psi}$ binds all free variables in R. This complication can be eliminated in an implementation of our calculus based on de Bruijn indexes. In general, we must again ensure that the result is a canonical term, we will define contextual substitution herditarily following the ideas for hereditary ordinary substitutions. Just as we annotated the substitution $[M/x]_A$ with the type of the variable x, we will annotate the contextual substitution $[\![\Psi.M/u]\!]_{A[\Psi]}$ with the type of the meta-variable $A[\Psi]$. We will abbreviate $A[\Psi]$ with α for better readability.

$$
\begin{array}{lcll}
[\![\hat{\Psi}.R/u]\!]_\alpha(\lambda y.N) & = & \lambda y.N' & \text{where } N' = [\![\hat{\Psi}.R/u]\!]_\alpha N \\
[\![\hat{\Psi}.R/u]\!]_\alpha(c \cdot S) & = & c \cdot S' & \text{where } S' = [\![\hat{\Psi}.R/u]\!]_\alpha S \\
[\![\hat{\Psi}.R/u]\!]_\alpha(x \cdot S) & = & x \cdot S' & \text{where } S' = [\![\hat{\Psi}.R/u]\!]_\alpha S \\
[\![\hat{\Psi}.R/u]\!]_\alpha(u[\tau]) & = & R' & \text{where } \tau' = [\![\hat{\Psi}.R/u]\!]_\alpha \tau \text{ and } R' = [\tau'/\Psi]_\psi R \\
[\![\hat{\Psi}.R/u]\!]_\alpha(v[\tau]) & = & v[\tau'] & \text{where } \tau' = [\![\hat{\Psi}.R/u]\!]_\alpha \tau \text{ and provided } v \neq u \\
[\![\hat{\Psi}.R/u]\!]_\alpha(\cdot) & = & \cdot & \\
[\![\hat{\Psi}.R/u]\!]_\alpha(\tau, N/y) & = & \tau', N'/y & \text{where } \tau' = [\![\hat{\Psi}.R/u]\!]_\alpha \tau \text{ and } N' = [\![\hat{\Psi}.R/u]\!]_\alpha N \\
[\![\hat{\Psi}.R/u]\!]_\alpha(\tau, R'/\!\!/y)_\alpha & = & \tau', R''/\!\!/y & \text{where } \tau' = [\![\hat{\Psi}.R/u]\!]_\alpha \tau \text{ and } R'' = [\![\hat{\Psi}.R/u]\!]_\alpha R'
\end{array}
$$

Applying $[\![\hat{\Psi}.R/u]\!]$ to the closure $u[\tau]$ first obtains the simultaneous substitution $\tau' = [\![\Psi.R/u]\!]\tau$, but instead of returning $R[\tau']$, it proceeds to eagerly apply τ' to R. Before τ' can be carried out, however, it's domain must be renamed to match the variables in Ψ, denoted by τ'/Ψ. We note that maintaining canonical forms is easy since we enforce that every occurrence of a meta-variable must have atomic type. While the definition of the discussed case may seem circular at first, it is actually well-founded. The computation of τ' recursively invokes $[\![\hat{\Psi}.R/u]\!]$ on τ, a constituent of $u[\tau]$. Then τ'/Ψ is applied to R, but applying simultaneous substitutions has already been defined without appeal to meta-variable substitution.

Substitution of a meta-variable satisfies the following contextual substitution property.

Theorem 3.5 Contextual Substitution on Terms.

(1) *If* $\Delta; \Psi \vdash R \Leftarrow P$ *and* $(\Delta, u{::}P[\Psi], \Delta'); \Gamma \vdash N \Leftarrow C$ *and* $[\![\hat{\Psi}.R/u]\!]N = N'$ *then* $(\Delta, \Delta'); \Gamma \vdash N' \Leftarrow C$.

(2) *If* $\Delta; \Psi \vdash R \Leftarrow P$ *and* $(\Delta, u{::}P[\Psi], \Delta'); \Gamma \vdash R' \Rightarrow P'$ *and* $[\![\hat{\Psi}.R/u]\!]R' = R''$ *then* $(\Delta, \Delta'); \Gamma \vdash R'' \Rightarrow P'$.

(3) *If* $\Delta; \Psi \vdash R \Leftarrow P$ *and* $(\Delta, u{::}P[\Psi], \Delta'); \Gamma \vdash \tau \Leftarrow \Theta$ *and* $\tau' = [\![\hat{\Psi}.R/u]\!]\tau$ *then* $(\Delta, \Delta'); \Gamma \vdash \tau' : \Theta$.

Proof. By simple inductions on the second given derivation, appealing to Theorem 3.4 in the case for meta-variables. □

3.5 Simultaneous contextual substitution

This contextual substitution can be extended to a simultaneous contextual substitution in a similar way we extended ordinary substitutions to simultaneous substi-

tutions

$$\text{Simultaneous contextual substitutions } \theta ::= \cdot \mid \theta, \hat{\Psi}.R/u$$

We write θ for a simultaneous substitution $[\![\hat{\Psi}_1.R_1/u_1, \ldots, \hat{\Psi}_n.R_n/u_n]\!]$. We first define typing rules for simultaneous contextual substitutions.

$$\frac{}{\Delta \vdash (\cdot) \Leftarrow (\cdot)} \qquad \frac{\Delta; \Psi \vdash R \Leftarrow P \quad \Delta \vdash \theta \Leftarrow \Delta'}{\Delta \vdash (\theta, \hat{\Psi}.R/u) \Leftarrow (\Delta', u{::}P[\Psi])}$$

The new operation of substitution is compositional, but two interesting situations arise: when a variable u is encountered, and when we substitute into a λ-abstraction. We again annotated the simultaneous contextual substitution $[\![\theta]\!]_\Delta$ with its domain.

Objects

$$\begin{array}{lcll}
[\![\theta]\!]_\Delta(\lambda y.N) & = & \lambda y.N' & \text{where } N' = [\![\theta]\!]_\Delta N \\
[\![\theta]\!]_\Delta(c \cdot S) & = & c \cdot S' & \text{where } S' = [\![\theta]\!]_\Delta S \\
[\![\theta]\!]_\Delta(x \cdot S) & = & x \cdot S' & \text{where } S' = [\![\theta]\!]_\Delta S \\
[\![\theta]\!]_\Delta(u[\sigma]) & = & R & \text{where } \theta = (\theta_1, \hat{\Psi}.R/u, \theta_2) \text{ and } \sigma' = [\![\theta]\!]_\Delta(\sigma) \\
 & & & \text{and } R' = [\sigma']_\psi R \text{ where } u{::}P[\Psi] \in \Delta
\end{array}$$

Ordinary Substitutions

$$\begin{array}{lcll}
[\![\theta]\!]_\Delta(\cdot) & = & \cdot & \\
[\![\theta]\!]_\Delta(\sigma, N/y) & = & (\sigma', N'/y) & \text{where } \sigma' = [\![\theta]\!]_\Delta\sigma \text{ and } N' = [\![\theta]\!]_\Delta N \\
[\![\theta]\!]_\Delta(\sigma, R/\!\!/y) & = & (\sigma', R'/\!\!/y) & \text{where } \sigma' = [\![\theta]\!]_\Delta\sigma \text{ and } R' = [\![\theta]\!]_\Delta R
\end{array}$$

We remark that the rule for substitution into a λ-abstraction does not need to extend the substitution θ nor does it need any other restrictions. This is because the object R is defined in a different context, which is accounted for by the explicit substitution stored at occurrences of u. Finally, consider the case of substituting into a closure, which is the critical case of this definition.

$$\begin{array}{lcll}
[\![\theta]\!]_\Delta(u[\sigma]) & = & R' & \text{where } \theta = (\theta_1, \hat{\Psi}.R/u, \theta_2) \text{ and } \sigma' = [\![\theta]\!]_\Delta(\sigma) \\
 & & & \text{and } R' = [\sigma']_\psi R \text{ where } u{::}P[\Psi] \in \Delta
\end{array}$$

This is clearly well-founded, because σ is a subexpression (so $[\![R/u]\!]\sigma$ will terminate) and the application of an ordinary substitution has been defined previously without reference to the new form of substitution.

Similar to composition of ordinary substitution, composition for contextual substitutions holds (see [Nanevski et al. 2006]).

3.6 Pattern substitutions

An important fragment of higher-order terms, is the pattern fragment. While in general many algorithms such as unification are undecidable in the general higher-order case, these operations become decidable with suitable restrictions to patterns

[Miller 1991a]. Higher-order patterns are terms where meta-variables must be applied to distinct bound variables. In our setting, this means that substitution σ which is associated with the meta-variable $u[\sigma]$ must be a pattern substitution of the form $[x_{\phi(1)}/\!\!/x_1, \ldots, x_{\phi(n)}/\!\!/x_n]$. In other words the pattern substitution is just a renaming of variables.

When we consider only closures of meta-variables together with pattern substitutions then applying the modal substitution θ to a term M will directly yield a canonical term and it is unnecessary to annotate $[\![\theta]\!]M$ with the domain of θ. In the subsequent development, we therefore omit this annotation.

Finally, we note that applying a modal substitution θ to a pattern substitution σ does not change σ itself, since the range of σ refers only to bound variables, while θ refers to modal variables.

LEMMA 3.6.
If $\Delta' \vdash \theta : \Delta$ *and* σ *is a pattern substitution, s.t.* $\Delta; \Gamma \vdash \sigma : \Psi$ *then* $[\![\theta]\!]_\Delta(\sigma) = \sigma$

PROOF. Induction on the structure of σ □

4. HIGHER-ORDER SUBSTITUTION TREES

Higher-order substitution trees are designed for linear higher-order patterns and are built with contextual substitutions. Recall the example given earlier, where we described equality preserving transformations on logical propositions. One such transformation was the following:

$$\texttt{eq}\ \ (\texttt{and}\ \ (\texttt{forall}\ \lambda\texttt{x.A x})\ \texttt{B})\ \ (\texttt{forall}\ \lambda\texttt{x.(and}\ \ (\texttt{A x})\ \texttt{B})).$$

In this example, `A` and `B` denote meta-variables which are present in the original query, while `x` denotes an ordinary bound variable. In our contextual modal type theory, this term would be represented as follows:

$$\texttt{eq}\ \ (\texttt{and}\ \ (\texttt{forall}\ \lambda\texttt{x.u[x/y]})\ \texttt{v[}\cdot\texttt{]})\ \ (\texttt{forall}\ \lambda\texttt{x.(and}\ \ (\texttt{u[x/y]})\ \texttt{v[}\cdot\texttt{]})).$$

The meta-variables `u[x/y]` and `v[·]` directly encode the dependencies which must be obeyed. As we can see, the meta-variable `v[·]` is not fully applied, since the substitution associated with the meta-variable `v` is empty although `v[·]` occurs within the scope of a λ-binder. Hence we translate this term into:

$$\texttt{eq}\ \ (\texttt{and}\ \ (\texttt{forall}\ \lambda\texttt{x.u[x/y]})\ \texttt{v[}\cdot\texttt{]})\ \ (\texttt{forall}\ \lambda\texttt{x.(and}\ \ (\texttt{u[x/y]})\ \texttt{w[x/y]}))\quad \text{where } \texttt{w[x/y]} \doteq \texttt{v[}\cdot\texttt{]}$$

When computing the most specific generalization between two terms to build the substitution tree, we will create internal meta-variables. For example, $\texttt{i}_3[\cdot]$ and $\texttt{i}_4[\cdot]$ are internal meta-variables in

$$\texttt{eq}\quad (\texttt{and}\ (\texttt{i}_3[.\,])\ (\texttt{i}_4[.\,]))\quad (\texttt{forall}\ \lambda\texttt{x.(and}\ (\texttt{u[x/y]})\ (\texttt{w[x/y]}))).$$

In the definition of higher-order substitution trees we will distinguish between a modal context Δ which denotes the original meta-variables such as `u`, `v`, and `w`, a

modal context Σ for the internal meta-variables $\mathtt{i_3}$ and $\mathtt{i_4}$, and a context Γ denoting ordinary variables. A higher-order substitution tree is an ordered n-ary tree.

(1) A node with a contextual substitution ρ such that $\Delta \vdash \rho \Leftarrow \Sigma$ and no children is called a leaf and is a tree.

(2) If $N_1, \ldots, N_n$ are trees such that for every i, N_i has a substitution ρ_i, such that $(\Delta, \Sigma_i) \vdash \rho_i \Leftarrow \Sigma$, and a list of children C_i,
then a node with a contextual substitution ρ, such that $(\Delta, \Sigma) \vdash \rho \Leftarrow \Sigma'$, and children $N_1, \ldots, N_n$ is a tree.

First we note that the domain of a contextual substitution $(\Delta, \Sigma) \vdash \rho \Leftarrow \Sigma'$, can be extended to the domain (Δ, Σ') by extending ρ with the contextual identity substitution id_Δ. Then for every path from the top node ρ_0 where $(\Delta, \Sigma_1) \vdash (\mathsf{id}_\Delta, \rho_0) : (\Delta, \Sigma_0)$ to the leaf node ρ_n, we have $\Delta \vdash [\![\mathsf{id}_\Delta, \rho_n]\!]([\![\mathsf{id}_\Delta, \rho_{n-1}]\!] \ldots (\mathsf{id}_\Delta, \rho_0)) : (\Delta, \Sigma_0)$. In other words, there are no internal meta-variables left after we compose all the substitutions ρ_n up to ρ_0. We assume that all meta-variables occurring in one path are unique, and are fully applied, i.e. every meta-variable $u{::}P[\Psi]$ where $\Psi = x_1{:}A_1, \ldots, x_n{:}A_n$ is applied to all the bound variables in Ψ.

Note that there are no typing dependencies among the variables in Σ and they can be arbitrarily re-ordered. Moreover, we point out that while it is convenient to consider the extended modal substitution $(\mathsf{id}_\Delta, \rho_i)$ in the theory, we do not need to carry around explicitly the contextual substitution id_Δ in an implementation, but can always assume that any substitution ρ_i can be extended appropriately.

The algorithms for insertion and retrieval in substitution trees are based on most specific linear generalization (mslg) and unifiability. Types themselves do not play a role when computing the mslg and unifiers. However, we assume the term is well-typed before it is inserted into the substitution tree, and we will show that the term can be decomposed into contextual substitutions such that their composition results in the original term.

5. INSERTION

Insertion of a term R into the index is viewed as insertion of the substitution $\hat{\Psi}.R/i_0$. Assuming that R has type P in a modal context Δ and a bound variable context Ψ. $\hat{\Psi}.R/i_0$ is a contextual substitution such that $\Delta \vdash \hat{\Psi}.R/i_0 \Leftarrow i_0{::}P[\Psi]$. This will simplify the following theoretical development. Again we note that we do not need to carry around explicitly the contextual substitution id_Δ, but can always assume that any substitution can be extended appropriately. The insertion process works by following down a path in the tree that is *compatible* with the contextual substitution ρ. To formally define insertion, we first describe the most specific linear generalization of two objects, and then show how to compute the most specific linear generalization (mslg) of two contextual substitutions.

5.1 Most specific generalization of two linear objects

Computing the most specific linear generalization of two contextual substitutions relies on finding the most specific linear generalization of two objects. Recall that we require that all objects are linear higher-order patterns and are in canonical forms. Moreover, we assume that all meta-variables are lowered and have atomic

type. We define the computation of the most specific linear generalization of two terms next.

$$
\begin{array}{lll}
(\Delta,\Sigma);\Gamma \vdash\ M_1 \sqcup M_2\ : A & \Longrightarrow M/(\Sigma',\ \theta_1,\theta_2) & M \text{ is the mslg of } M_1 \text{ and } M_2 \\
(\Delta,\Sigma);\Gamma \vdash\ R_1 \underline{\sqcup} R_2\ : P & \Longrightarrow R/(\Sigma',\ \theta_1,\theta_2) & R \text{ is the mslg of } R_1 \text{ and } R_2 \\
(\Delta,\Sigma);\Gamma \vdash\ S_1 \underline{\sqcup} S_2\ : A > P & \Longrightarrow S/(\Sigma',\ \theta_1,\theta_2) & S \text{ is the mslg of } S_1 \text{ and } S_2
\end{array}
$$

If the canonical terms M_1 and M_2 have type A in modal context (Δ,Σ) and bound variable context Γ, then M is the most specific linear generalization of M_1 and M_2 such that $[\![\theta_1]\!]M$ is equal to M_1 and $[\![\theta_2]\!]M$ is equal to M_2. Moreover, θ_1 and θ_2 are contextual substitutions which map meta-variables from Σ' to the modal context (Δ,Σ). Finally, $(\Delta,\Sigma');\Gamma \vdash M \Leftarrow A$. Similar invariant holds for atomic terms and spines. If S_1 and S_2 are spines from heads of type A to terms of type P, then S is the mslg of S_1 and S_2 such that $[\![\theta_1]\!]S$ is equal to S_1 and $[\![\theta_2]\!]S$ is equal to S_2. θ_1 and θ_2 are contextual substitutions which map meta-variables from Σ' to the modal context (Δ,Σ).

We think of M_1 (R_1, or S_1) as an object which is already in the index and M_2 (R_2, or S_2) is the object to be inserted. As a consequence, only M_1 (R_1, and S_1) may refer to the internal variables in Σ, while M_2 (R_2, and S_2) only depends on Δ. In defining the most specific linear generalization, we distinguish between the the internal meta-variables i and the global meta-variables u in the rules, because the play different roles. The inference rules for computing the mslg are given next.

Normal linear objects

$$\frac{(\Sigma,\Delta);\Gamma, x{:}A_1 \vdash M_1 \sqcup M_2 : A_2 \Longrightarrow M/(\Sigma',\theta_1,\theta_2)}{(\Delta,\Sigma);\Gamma \vdash \lambda x.M_1 \sqcup \lambda x.M_2 : A_1 \to A_2 \Longrightarrow \lambda x.M/(\Sigma',\ \theta_1,\theta_2)}\ \textsf{a-lam}$$

$$\frac{(\Delta,\Sigma);\Gamma \vdash R_1 \underline{\sqcup} R_2 : P \Longrightarrow R/(\Sigma',\theta_1,\theta_2)}{(\Delta,\Sigma);\Gamma \vdash R_1 \sqcup R_2 : P \Longrightarrow R/(\Sigma',\ \theta_1,\theta_2)}\ \textsf{a-coe}$$

In the rule for lambda, we do not need to worry about capture, since meta-variables and bound variables are defined in different context. Hence, we can just traverse the body of the lambda-term. Next, we consider the rules for neutral objects.

Atomic linear objects

$$\frac{u{::}P[\Psi]) \in \Delta}{(\Delta,\Sigma);\Psi \vdash u[\pi_\psi] \underline{\sqcup} u[\pi_\psi] : P \Longrightarrow u[\pi_\psi]/(\cdot,\cdot,\cdot)}\ \textsf{a-mvar-same}$$

$$\frac{u{::}P[\Psi] \in \Delta \quad i \text{ must be new} \quad R \neq u[\pi]}{(\Delta,\Sigma);\Psi \vdash u[\pi_\psi] \underline{\sqcup} R : P \Longrightarrow i[\mathsf{id}_\psi]/(i{::}P[\Psi],\ \hat{\Psi}.u[\pi_\psi]/i,\ \hat{\Psi}.R/i)}\ \textsf{a-mvar-diff-1}$$

$$\frac{u{::}P[\Psi] \in \Delta \quad R \neq i_0[\mathsf{id}_\psi] \text{ for some } i_0 \quad R \neq u[\pi] \quad i \text{ must be new}}{(\Delta,\Sigma);\Psi \vdash R \underline{\sqcup} u[\pi_\psi] : P \Longrightarrow i[\mathsf{id}_\psi]/(i{::}P[\Psi],\ \hat{\Psi}.R/i,\ \hat{\Psi}.u[\pi]/i)}\ \textsf{a-mvar-diff-2}$$

$$\frac{i{::}P[\Psi] \in \Sigma}{(\Delta,\Sigma);\Psi \vdash i[\mathsf{id}_\psi] \sqcup R : P \Longrightarrow i[\mathsf{id}_\psi]/(i{::}P[\Psi],\ \hat{\Psi}.i[\mathsf{id}_\psi]/i,\ \hat{\Psi}.R/i)}\ \textsf{a-ivar}$$

$$\frac{(\Delta,\Sigma);\Psi \vdash S_1 \sqcup S_2 : A > P \Longrightarrow S/(\Sigma',\theta_1,\theta_2) \quad x{:}A \in \Psi}{(\Delta,\Sigma);\Psi \vdash x \cdot S_1 \sqcup x \cdot S_2 : P \Longrightarrow x \cdot S/(\Sigma',\theta_1,\theta_2)}\ \textsf{a-var}$$

$$\frac{(\Delta,\Sigma);\Psi \vdash S_1 \sqcup S_2 : A > P \Longrightarrow S/(\Sigma',\theta_1,\theta_2) \quad c{:}A \in \Sigma}{(\Delta,\Sigma);\Psi \vdash c \cdot S_1 \sqcup c \cdot S_2 : P \Longrightarrow c \cdot S/(\Sigma',\theta_1,\theta_2)}\ \textsf{a-con}$$

$$\frac{H_1 \cdot S_1 = R_1 \quad R_2 = H_2 \cdot S_2 \quad H_1 \neq H_2 \quad i \text{ must be new}}{(\Delta,\Sigma);\Psi \vdash R_1 \sqcup R_2 : P \Longrightarrow i[\mathsf{id}_\psi]/((i{::}P[\Psi]),\hat{\Psi}.R_1/i,\hat{\Psi}.R_2/i))}\ \textsf{a-diff}$$

Normal linear spines

$$\frac{}{(\Delta,\Sigma);\Psi \vdash \mathsf{nil} \sqcup \mathsf{nil} : P > P \Longrightarrow \mathsf{nil}/(\cdot,\cdot,\cdot)}\ \textsf{a-nil}$$

$$\frac{\begin{array}{l}(\Delta,\Sigma);\Psi \vdash M_1 \ \sqcup\ M_2 \ : A_1 \qquad \Longrightarrow M/(\Sigma_1,\ \theta_1,\theta_2)\\ (\Delta,\Sigma);\Psi \vdash S_1 \ \sqcup\ S_2 \ : A_2 > P \ \Longrightarrow S/(\Sigma_2,\ \theta_1',\theta_2')\\ \text{and } \Sigma' = (\Sigma_1,\Sigma_2) \quad \theta = (\theta_1,\theta_1') \quad \theta' = (\theta_2,\theta_2')\end{array}}{(\Delta,\Sigma);\Psi \vdash (M_1;S_1) \sqcup (M_2;S_2) : A_1 \to A_2 > P \Longrightarrow (M;S)/(\Sigma',\theta,\theta')}\ \textsf{a-cons}$$

Rule a-mvar-same treats the case where both terms are meta-variables. Note that we require that both meta-variables must be the same and their associated substitutions must also be equal. In the rule a-mvar-diff-1 and a-mvar-diff-2, we just create the substitution $\hat{\Psi}.u[\pi_\psi]/i$. In general, we would need to create $[\mathsf{id}_\psi]^{-1}(u[\pi_\psi])$, but since we know that π is a permutation substitution, we know that $[\mathsf{id}_\psi]^{-1}(\pi)$ always exists. In addition, the inverse substitution of the identity is the identity. The different roles of meta-variables u and internal meta-variables i becomes clear in the rules above. In a-mvar-diff-1 and a-mvar-diff-2 we pick a new internal meta-variable i while we re-use the internal meta-variable i in rule a-ivar. If we encounter a meta-variable u and another object R then we generalize and generate a new internal meta-variable i. However, when we encounter an internal meta-variable i and another object R, we do not generate a new internal meta-variable, because i will be defined later on in the branch of the substitution tree, and we will need to continue to insert R into the tree. This is important for maintaining the invariant that any child of $(\Delta,\Sigma_2) \vdash \rho \Leftarrow \Sigma_1$ has the form $(\Delta,\Sigma_3) \vdash \rho' \Leftarrow \Sigma_2$ during insertion (see the insertion algorithm later on).

In rule a-var, a-con, and a-diff, we distinguish on the head symbol H and compute the most specific linear generalization of two objects $H_1 \cdot S_1$ and $H_1 \cdot S_1$. If H_1 and H_2 are not equal, then we generate a new internal meta-variable $i[\mathsf{id}_\psi]$ together with the substitutions $\hat{\Psi}.(H_1 \cdot S_1)/i$ and $\hat{\Psi}.(H_2 \cdot S_2)/i$ (see a-diff rule). Otherwise, we traverse the spines S_1 and S_2 and compute the most specific linear generalization of them (see rules a-var and a-con). Finally the rules for computing the most specific generalization of two spines are straightforward. We compute the mslg of all the sub-expressions, and just combine the substitution θ_1 and θ_1' and θ_2 and θ_2' respectively. As we require that all meta-variables occur uniquely, and hence there

are no dependencies among Σ_1 and Σ_2.

Definition 5.1 *Compatibility of neutral objects.*
If $(\Delta, \Sigma); \Psi \vdash R_1 \sqcup\!\!\underline{\sqcup} R_2 : P \Longrightarrow i[\mathsf{id}_\psi]/(i{::}P[\Psi], \hat{\Psi}.R_1/i, \hat{\Psi}.R_2/i)$, then two neutral objects R_1 and R_2 are called incompatible. Otherwise, we call R_1 and R_2 compatible.

In other words, we call two terms compatible, if they share at least the head symbol or a λ-prefix. We are now ready to prove correctness of the algorithm for computing the most specific linear generalization of higher-order linear patterns.

THEOREM 5.2 SOUNDNESS OF MSLG FOR OBJECTS.

(1) *If* $(\Delta, \Sigma); \Gamma \vdash M_1 \sqcup M_2 : A \Longrightarrow M/(\Sigma', \theta_1, \theta_2)$ *and*
$(\Delta, \Sigma); \Gamma \vdash M_1 \Leftarrow A$ *and* $(\Delta, \Sigma); \Gamma \vdash M_2 \Leftarrow A$
then $(\Delta, \Sigma) \vdash \theta_1 \Leftarrow \Sigma'$ *and* $(\Delta, \Sigma) \vdash \theta_2 \Leftarrow \Sigma'$ *and*
$M_1 = [\![\theta_1]\!]M$ *and* $M_2 = [\![\theta_2]\!]M$ *and* $(\Delta, \Sigma'); \Gamma \vdash M \Leftarrow A$.

(2) *If* $(\Delta, \Sigma); \Gamma \vdash R_1 \sqcup\!\!\underline{\sqcup} R_2 : P \Longrightarrow R/(\Sigma', \theta_1, \theta_2)$ *and*
$(\Delta, \Sigma); \Gamma \vdash R_1 \Rightarrow P_1$ *and* $(\Delta, \Sigma); \Gamma \vdash R_2 \Rightarrow P_2$ *and* $P_1 = P_2 = P$
then $(\Delta, \Sigma) \vdash \theta_1 \Leftarrow \Sigma'$ *and* $(\Delta, \Sigma) \vdash \theta_2 \Leftarrow \Sigma'$ *and*
$R_1 = [\![\theta_1]\!]R$ *and* $R_2 = [\![\theta_2]\!]R$ *and* $(\Delta, \Sigma'); \Gamma \vdash R \Rightarrow P'$ *and* $P' = P$.

(3) *If* $(\Delta, \Sigma); \Gamma \vdash S_1 \sqcup\!\!\underline{\sqcup} S_2 : A > P \Longrightarrow S/(\Sigma', \theta_1, \theta_2)$ *and*
$(\Delta, \Sigma); \Gamma \vdash S_1 > A \Rightarrow P_1$ *and* $(\Delta, \Sigma); \Gamma \vdash S_2 > A \Rightarrow P_2$ *and* $P_2 = P = P_1$
then $(\Delta, \Sigma) \vdash \theta_1 : \Sigma'$ *and* $(\Delta, \Sigma) \vdash \theta_2 : \Sigma'$ *and*
$(\Delta, \Sigma'); \Gamma \vdash S > A \Rightarrow P$ *and* $S_1 = [\![\theta_1]\!]S$ *and* $S_2 = [\![\theta_2]\!]S$.

PROOF. Simultaneous induction on the structure of the first derivation. We give here a few cases.

Case. $\mathcal{D} = (\Delta, \Sigma); \Gamma \vdash \lambda x.M_1 \sqcup \lambda x.M_2 : A_1 \to A_2 \Longrightarrow \lambda x.M/(\Sigma', \theta_1, \theta_2)$

$(\Delta, \Sigma); \Gamma, x{:}A_1 \vdash M_1 \sqcup M_2 : A_2 \Longrightarrow M/(\Sigma', \theta_1, \theta_2)$	by premise
$(\Delta, \Sigma); \Gamma \vdash \lambda x.M_1 \Leftarrow A_1 \to A_2$	by assumption
$(\Delta, \Sigma); \Gamma, x{:}A_1 \vdash M_1 \Leftarrow A_2$	by inversion
$(\Delta, \Sigma); \Gamma \vdash \lambda x.M_2 \Leftarrow A_1 \to A_2$	by assumption
$(\Delta, \Sigma); \Gamma, x{:}A_1 \vdash M_2 \Leftarrow A_2$	by inversion
$(\Delta, \Sigma) \vdash \theta_1 \Leftarrow \Sigma'$	by i.h.
$(\Delta, \Sigma) \vdash \theta_2 \Leftarrow \Sigma'$	by i.h.
$M_1 = [\![\theta_1]\!]M$	by i.h.
$\lambda x.M_1 = \lambda x.[\![\theta_1]\!]M$	by rule
$\lambda x.M_1 = [\![\theta_1]\!](\lambda x.M)$	by contextual substitution definition
$M_2 = [\![\theta_2]\!]M$	by i.h.
$\lambda x.M_2 = \lambda x.[\![\theta_2]\!]M$	by rule
$\lambda x.M_2 = [\![\theta_2]\!](\lambda x.M)$	by contextual substitution definition
$(\Delta, \Sigma'); \Gamma, x{:}A_1 \vdash M \Leftarrow A_2$	by i.h.
$(\Delta, \Sigma'); \Gamma \vdash \lambda x.M \Leftarrow A_1 \to A_2$	by rule

Case. $\mathcal{D} = (\Delta; \Sigma); \Gamma \vdash R_1 \sqcup R_2 : P \Longrightarrow R/(\Sigma', \theta_1, \theta_2)$

$(\Delta, \Sigma); \Gamma \vdash R_1 \sqcup\!\!\underline{\sqcup} R_2 : P \Longrightarrow R/(\Sigma', \theta_1, \theta_2)$	by rule
$(\Delta, \Sigma); \Gamma \vdash R_1 \Leftarrow P$	by ass

$(\Delta, \Sigma); \Gamma \vdash R_1 \Rightarrow P_1$ and $P_1 = P$ by rule
$(\Delta, \Sigma); \Gamma \vdash R_2 \Leftarrow P$ by ass
$(\Delta, \Sigma); \Gamma \vdash R_2 \Rightarrow P_2$ and $P_2 = P$ by rule
$(\Delta, \Sigma) \vdash \theta_1 \Leftarrow \Sigma'$ by i.h.
$(\Delta, \Sigma) \vdash \theta_2 \Leftarrow \Sigma'$ and
$R_1 = [\![\theta_1]\!]R$ and $R_2 = [\![\theta_2]\!]R$ and $(\Sigma', \Delta); \Gamma \vdash R \Rightarrow P'$ and $P' = P$.
$(\Delta, \Sigma'); \Gamma \vdash R \Leftarrow P$ by rule

Case. $\mathcal{D} = (\Delta, \Sigma); \Gamma \vdash u[\pi] \mathbin{\underline{\sqcup}} u[\pi] : P \Longrightarrow u[\pi]/(\cdot, \cdot, \cdot)$

$u{::}P[\Psi] \in \Delta$ and $\Delta; \Gamma \vdash \pi \Leftarrow \Psi$ by premise
$(\Delta, \Sigma); \Gamma \vdash u[\pi] \Rightarrow P_1$ and $P = P_1$ by assumption
$u[\pi] = u[\pi]$ by reflexivity
$(\Delta, \Sigma) \vdash \cdot \Leftarrow \cdot$ by rule
$\Delta; \Gamma \vdash u[\pi] \Rightarrow P_1$ by rule

Case. $\mathcal{D} = (\Delta, \Sigma); \Gamma \vdash u[\pi] \mathbin{\underline{\sqcup}} R : P \Longrightarrow i[\mathsf{id}_\gamma]/(i{::}P[\Gamma],\ \hat{\Gamma}.u[\pi]/i,\ \hat{\Gamma}.R/i)$

$u{::}P[\Psi] \in \Delta$ and $\Delta; \Gamma \vdash \pi \Leftarrow \Psi$ by premise
$(\Delta, \Sigma); \Gamma \vdash u[\pi] \Rightarrow P_1$ and $P_1 = P$ by assumption
$(\Delta, \Sigma); \Gamma \vdash R \Rightarrow P_2$ and $P_2 = P$ by assumption
$(\Delta, \Sigma); \Gamma \vdash R \Leftarrow P$ by rule
$(\Delta, \Sigma); \Gamma \vdash u[\pi] \Leftarrow P$ by rule
$u[\pi] = [\![\hat{\Gamma}.u[\pi]/i]\!]i[\mathsf{id}_\gamma]$
$u[\pi] = u[\pi]$ by reflexivity
$R = [\![\hat{\Gamma}.R/i]\!]i[\mathsf{id}_\gamma]$
$R = R$ by reflexivity
$(\Delta, \Sigma) \vdash \hat{\Gamma}.R/i \Leftarrow i{::}P[\Gamma]$ by rule using assumption
$(\Delta, \Sigma) \vdash u[\pi]/i \Leftarrow i{::}P[\Gamma]$ by rule using assumption
$(\Delta, i{::}P[\Gamma]); \Gamma \vdash \mathsf{id}_\gamma \Leftarrow \Gamma$ by definition
$(\Delta, i{::}P[\Gamma]); \Gamma \vdash i[\mathsf{id}_\gamma] \Rightarrow P$ by rule
$P = P$ by reflexivity

Case. $\mathcal{D} = (\Delta, \Sigma); \Gamma \vdash c \cdot S_1 \mathbin{\underline{\sqcup}} c \cdot S_2 : P \Longrightarrow c \cdot S/(\Sigma',\ \theta_1,\ \theta_2)$

$(\Delta, \Sigma); \Gamma \vdash S_1 \mathbin{\underline{\sqcup}} S_2 : A > P \Longrightarrow S/(\Sigma', \theta_1, \theta_2)$ by premise
$(\Delta, \Sigma); \Gamma \vdash c \cdot S_1 \Rightarrow P_1$ and $P_1 = P$ by assumption
$(\Delta, \Sigma); \Gamma \vdash S_1 > A \Rightarrow P_1$ by inversion
$(\Delta, \Sigma); \Gamma \vdash c \cdot S_2 \Rightarrow P_2$ and $P_2 = P$ by assumption
$(\Delta, \Sigma); \Gamma \vdash S_2 > A \Rightarrow P_2$ by inversion
$S_1 = [\![\theta_1]\!]S$ by i.h.
$S_2 = [\![\theta_2]\!]S$ by i.h.
$(\Delta, \Sigma) \vdash \theta_1 \Leftarrow \Sigma'$ by i.h.
$(\Delta, \Sigma) \vdash \theta_2 \Leftarrow \Sigma'$ by i.h.
$c \cdot S_1 = c \cdot\ [\![\theta_1]\!]S$ by rule
$c \cdot S_1 = [\![\theta_1]\!](c \cdot S)$ by contextual substitution definition
$c \cdot S_2 = c \cdot\ [\![\theta_2]\!]S$ by rule
$c \cdot S_2 = [\![\theta_2]\!](c \cdot\ S)$ by contextual substitution definition
$(\Delta, \Sigma'); \Gamma \vdash S > A \Rightarrow P$ by i.h.

$(\Delta, \Sigma'); \Gamma \vdash c \cdot S \Rightarrow P$ by rule

Case. $\mathcal{D} = (\Delta, \Sigma); \Gamma \vdash R_1 \sqcup\!\!\underline{\sqcup} R_2 : P \Longrightarrow i[\mathsf{id}_\gamma]/(i{::}P[\Gamma],\ \hat{\Gamma}.R_1/i,\ \hat{\Gamma}.R_2/i)$

$R_1 = H_1 \cdot S_1$ and $R_2 = H_2 \cdot S_2$ and $H_1 \neq H_2$ by inversion
$(\Delta, \Sigma); \Gamma \vdash H_1 \cdot S_1 \Rightarrow P_1$ and $P_1 = P$ by assumption
$(\Delta, \Sigma); \Gamma \vdash H_1 \cdot S_1 \Leftarrow P$ by rule
$(\Delta, \Sigma); \Gamma \vdash H_2 \cdot S_2 \Rightarrow P_2$ and $P_2 = P$ by assumption
$(\Delta, \Sigma); \Gamma \vdash H_2 \cdot S_2 \Leftarrow P$ by rule
$H_1 \cdot S_1 = [\![\hat{\Gamma}.(H_1 \cdot S_1)/i]\!](i[\mathsf{id}_\gamma])$ by contextual substitution definition
$H_1 \cdot S_1 = H_1 \cdot S_1$ by reflexivity
$H_2 \cdot S_2 = [\![\hat{\Gamma}.(H_2 \cdot S_2)/i]\!](i[\mathsf{id}_\gamma])$ by contextual substitution definition
$H_2 \cdot S_2 = H_2 \cdot S_2$ by reflexivity
$(\Delta, i{::}P[\Gamma]); \Gamma \vdash \mathsf{id}_\gamma \Leftarrow \Gamma$ by definition
$(\Delta, i{::}P[\Gamma]); \Gamma \vdash i[\mathsf{id}_\gamma] \Rightarrow P$ by rule

Case. $\mathcal{D} = (\Delta, \Sigma); \Gamma \vdash (M_1; S_1) \sqcup\!\!\underline{\sqcup} (M_2; S_2) : (A_1 \to A_2) > P \Longrightarrow (M; S)/(\Sigma', \theta, \theta')$

$(\Delta, \Sigma); \Gamma \vdash M_1 \sqcup M_2 : A_1 \Longrightarrow M/(\Sigma_1,\ \theta_1,\ \theta_2)$ by premise
$(\Delta, \Sigma); \Gamma \vdash S_1 \sqcup\!\!\underline{\sqcup} S_2 : A_2 > P \Longrightarrow S/(\Sigma_2,\ \theta_1',\ \theta_2')$
$\Sigma' = (\Sigma_1, \Sigma_2)$, $\theta = (\theta_1, \theta_1')$, $\theta' = (\theta_2, \theta_2')$
$(\Delta, \Sigma); \Gamma \vdash (M_1; S_1) > A_1 \to A_2 \Rightarrow P_1$ and $P_1 = P$ by assumption
$(\Delta, \Sigma); \Gamma \vdash M_1 \Leftarrow A_1$ by inversion
$(\Delta, \Sigma); \Gamma \vdash S_1 > A_2 \Rightarrow P_1$
$(\Delta, \Sigma); \Gamma \vdash (M_2; S_2) > A_1 \to A_2 \Rightarrow P_2$ and $P_2 = P$ by assumption
$(\Delta, \Sigma); \Gamma \vdash M_2 \Leftarrow A_1$ by inversion
$(\Delta, \Sigma); \Gamma \vdash S_2 > A_2 \Rightarrow P_2$
$M_1 = [\![\theta_1]\!]M$ by i.h.
$M_2 = [\![\theta_2]\!]M$ by i.h.
$(\Delta, \Sigma_1); \Gamma \vdash M \Leftarrow A_1$ by i.h.
$(\Delta, \Sigma) \vdash \theta_1 \Leftarrow \Sigma_1$ by i.h.
$(\Delta, \Sigma) \vdash \theta_2 \Leftarrow \Sigma_1$ by i.h.
$(\Delta, \Sigma'); \Gamma \vdash M \Leftarrow A_1$ by weakening
$S_1 = [\![\theta_1']\!]S$ by i.h.
$S_2 = [\![\theta_2']\!]S$ by i.h.
$(\Delta, \Sigma_2); \Gamma \vdash S > A_2 \Rightarrow P$ by i.h.
$(\Delta, \Sigma) \vdash \theta_1' \Leftarrow \Sigma_2$ by i.h.
$(\Delta, \Sigma) \vdash \theta_2' \Leftarrow \Sigma_2$ by i.h.
$(\Delta, \Sigma',); \Gamma \vdash S > A_2 \Rightarrow P$ by weakening
$(\Delta, \Sigma) \vdash (\theta_1, \theta_1') \Leftarrow (\Sigma')$ θ_1 and θ_1' refer to distinct meta-variables by typing rules for contextual substitutions
$(\Delta, \Sigma) \vdash (\theta_2, \theta_2') \Leftarrow (\Sigma')$ θ_2 and θ_2' refer to distinct meta-variables by typing rules for contextual substitutions
$M_1 = [\![\theta_1, \theta_1']\!]M$ by lemma weakening
$M_2 = [\![\theta_2, \theta_2']\!]M$ by lemma weakening
$S_1 = [\![\theta_1, \theta_1']\!]S$ by lemma weakening
$S_2 = [\![\theta_2, \theta_2']\!]S$ by lemma weakening
$(M_1; S_1) = ([\![\theta_1, \theta_1']\!]M; [\![\theta_1, \theta_1']\!]S)$ by rule

$(M_1; S_1) = [\![\theta_1, \theta_1']\!](M; S)$ by contextual substitution definition
$(M_2; S_2) = ([\![\theta_2, \theta_2']\!]M; [\![\theta_2, \theta_2']\!]S)$ by rule
$(M_2; S_2) = [\![\theta_2, \theta_2']\!](M; S)$ by contextual substitution definition
$(\Delta, \Sigma'); \Gamma \vdash (M\ ;\ S) > A_1 \to A_2 \Rightarrow P$ by rule □

Next, we prove completeness.

THEOREM 5.3 COMPLETENESS OF MSLG OF TERMS.

(1) *If* $\Delta, \Sigma \vdash \theta_1 \Leftarrow \Sigma'$ *and* $\Delta, \Sigma \vdash \theta_2 \Leftarrow \Sigma'$ *and* θ_1 *and* θ_2 *are incompatible and* $\Delta, \Sigma; \Gamma \vdash M_1 \Leftarrow A$, $\Delta; \Gamma \vdash M_2 \Leftarrow A$, *and* $\Delta, \Sigma'; \Gamma \vdash M \Leftarrow A$ *and* $M_1 = [\![\theta_1]\!]M$ *and* $M_2 = [\![\theta_2]\!]M$
then there exists a contextual substitution θ_1^*, θ_2^*, *and a modal context* Σ^*, *such that* $(\Delta, \Sigma); \Gamma \vdash M_1 \sqcup M_2 : A \Longrightarrow M/(\Sigma^*, \theta_1^*, \theta_2^*)$ *and* $\theta_1^* \subseteq \theta_1$, $\theta_2^* \subseteq \theta_2$ *and* $\Sigma^* \subseteq \Sigma'$

(2) *If* $\Delta, \Sigma \vdash \theta_1 \Leftarrow \Sigma'$ *and* $\Delta, \Sigma \vdash \theta_2 \Leftarrow \Sigma'$ *and* θ_1 *and* θ_2 *are incompatible and* $\Delta, \Sigma; \Gamma \vdash R_1 \Rightarrow P_1$, $\Delta; \Gamma \vdash R_2 \Rightarrow P_2$, *and* $\Sigma', \Delta; \Gamma \vdash R \Rightarrow P$ *and* $P_1 = P_2 = P$ *and* $R_1 = [\![\theta_1]\!]R$ *and* $R_2 = [\![\theta_2]\!]R$
then there exists a contextual substitution θ_1^*, θ_2^*, *and a modal context* Σ^*, *such that* $(\Delta, \Sigma); \Gamma \vdash R_1 \sqcup R_2 : P \Longrightarrow R/(\Sigma^*, \theta_1^*, \theta_2^*)$ *and* $\theta_1^* \subseteq \theta_1$, $\theta_2^* \subseteq \theta_2$ *and* $\Sigma^* \subseteq \Sigma'$

(3) *If* $\Delta, \Sigma \vdash \theta_1 \Leftarrow \Sigma'$ *and* $\Delta, \Sigma \vdash \theta_2 \Leftarrow \Sigma'$ *and* θ_1 *and* θ_2 *are incompatible and* $(\Delta, \Sigma); \Gamma \vdash S_1 > A \Rightarrow P$, $(\Delta, \Sigma); \Gamma \vdash S_2 > A \Rightarrow P$, *and* $(\Delta, \Sigma'); \Gamma \vdash S > A \Rightarrow P$ *and* $S_1 = [\![\theta_1]\!]S$ *and* $S_2 = [\![\theta_2]\!]S$
then there exists a contextual substitution θ_1^*, θ_2^*, *and a modal context* Σ^*, *such that* $(\Delta, \Sigma); \Gamma \vdash S_1 \sqcup S_2 : A \Longrightarrow S/(\Sigma^*, \theta_1^*, \theta_2^*)$ *and* $\theta_1^* \subseteq \theta_1$, $\theta_2^* \subseteq \theta_2$ *and* $\Sigma^* \subseteq \Sigma'$.

PROOF. Simultaneous induction on the structure of M, R, and S.

Case. $R = u[\pi]$ and $u{::}P[\Phi] \in \Delta$
$(\Delta, \Sigma); \Gamma \vdash u[\pi] \Rightarrow P$ by assumption
$R_1 = [\![\theta_1]\!](u[\pi])$ by assumption
$R_1 = u[\pi]$ by contextual substitution definition
$R_2 = [\![\theta_2]\!](u[\pi])$ by assumption
$R_2 = u[\pi]$ by contextual substitution definition
$(\Delta, \Sigma); \Gamma \vdash u[\pi] \sqcup u[\pi] : P \Longrightarrow u[\pi]/(\cdot, \cdot, \cdot)$ by rule
$\cdot \subseteq \Sigma'$, $\cdot \subseteq \theta_1$, $\cdot \subseteq \theta_2$

Case. $M = \lambda x.M'$.
$M_1 = [\![\theta_1]\!](\lambda x.M')$ by assumption
$M_1 = \lambda x.[\![\theta_1]\!]M'$ by contextual substitution definition
$M_1' = [\![\theta_1]\!]M'$ and $M_1 = \lambda x.M_1'$ by inversion
$M_2 = [\![\theta_2]\!](\lambda x.M')$ by assumption
$M_2 = \lambda x.[\![\theta_2]\!]M'$ by contextual substitution definition
$M_2' = [\![\theta_2]\!]M'$ and $M_2 = \lambda x.M_2'$ by inversion
$(\Delta, \Sigma'); \Gamma \vdash \lambda x.M' \Leftarrow A_1 \to A_2$ by assumption
$(\Delta, \Sigma'); \Gamma, x{:}A_1 \vdash M' \Leftarrow A_2$ by inversion
$(\Delta, \Sigma); \Gamma \vdash \lambda x.M_1' \Leftarrow A_1 \to A_2$ by assumption

$(\Delta, \Sigma); \Gamma, x{:}A_1 \vdash M_1' \Leftarrow A_2$ by inversion
$(\Delta, \Sigma); \Gamma \vdash \lambda x.M_2' \Leftarrow A_1 \to A_2$ by assumption
$(\Delta, \Sigma); \Gamma, x{:}A_1 \vdash M_2' \Leftarrow A_2$ by inversion
$(\Delta, \Sigma); \Gamma, x{:}A_1 \vdash M_1' \sqcup M_2' : A_2 \Longrightarrow M'/(\Sigma^*, \theta_1^*, \theta_2^*)$ by i.h.
$\Sigma^* \subseteq \Sigma'$, $\theta_1^* \subseteq \theta_1$, $\theta_2^* \subseteq \theta_2$
$(\Delta, \Sigma); \Gamma \vdash \lambda x.M_1' \sqcup \lambda x.M_2' : A_1 \to A_2 \Longrightarrow \lambda x.M'/(\Sigma^*, \theta_1^*, \theta_2^*)$ by rule

Case. $R = i[\mathsf{id}_\gamma]$
$(\Delta; \Sigma); \Gamma \vdash i[\mathsf{id}_\gamma] \Rightarrow P$ by assumption
$i{::}P[\Gamma] \in \Sigma$ by inversion
$R_1 = [\![\theta_1]\!](i[\mathsf{id}_\gamma])$ by assumption
$R_2 = [\![\theta_2]\!](i[\mathsf{id}_\gamma])$ by assumption
$\hat{\Gamma}.R'/i \in \theta_1$ and $\hat{\Gamma}.R''/i \in \theta_2$ by assumption
R' and R'' are incompatible by assumption
$R_1 = R'$ by contextual substitution definition
$R_2 = R''$ by contextual substitution definition

Sub-Case 1. : $R_1 = u[\pi]$ and $R_2 = R''$
$(\Delta, \Sigma); \Gamma \vdash u[\pi] \sqcup R'' : P \Longrightarrow i[\mathsf{id}_\gamma]/(i{::}P[\Gamma], \hat{\Gamma}.u[\pi]/i, \hat{\Gamma}.R''/i)$ by rule
$i{::}P[\Gamma] \subseteq \Sigma'$, $(\hat{\Gamma}.u[\pi]/i) \subseteq \theta_1$, $(\hat{\Gamma}.R''/i) \subseteq \theta_2$

Sub-Case 2. : $R_1 = R'$ and $R_2 = u[\pi]$
$(\Delta, \Sigma); \Gamma \vdash R' \sqcup u[\pi] : P \Longrightarrow i[\mathsf{id}_\gamma]/(i{::}P[\Gamma], \hat{\Gamma}.R'/i, \hat{\Gamma}.u[\pi]/i)$ by rule
$(i{::}P[\Gamma] \subseteq \Sigma'$, $(\hat{\Gamma}.u[\pi]/i) \subseteq \theta_2$, $(\hat{\Gamma}.R'/i) \subseteq \theta_1$

Sub-Case 3. : $R_1 = H_1 \cdot S_1$ and $R_2 = H_2 \cdot S_2$
$H_1 \cdot S_1$ is incompatible with $H_2 \cdot S_2$ and $H_1 \neq H_2$ by assumption
$(\Delta, \Sigma); \Gamma \vdash H_1 \cdot S_1 \sqcup H_2 \cdot S_2 : P \Longrightarrow i[\mathsf{id}_\gamma]/(i{::}P[\Gamma], \hat{\Gamma}.(H_1 \cdot S_1)/i, \hat{\Gamma}.(H_2 \cdot S_2)/i)$ by rule
$(i{::}P[\Gamma]) \subseteq \Sigma'$, $(\hat{\Gamma}.H_1 \cdot S_1/i) \subseteq \theta_1$, $(\hat{\Gamma}.H_2 \cdot S_2/i) \subseteq \theta_2$ □

In the next Section, we extend the soundness and completeness property to substitutions.

5.2 Most specific generalization of two contextual substitutions

Building on the previous algorithm for computing the most specific generalization of two linear higher-order patterns, we extend the algorithm to contextual substitutions. We begin by giving the judgments for computing the most specific linear generalization of two contextual substitutions.

$$\Delta, \Sigma_1 \vdash \rho_1 \sqcup \rho_2 : \Sigma_2 \Longrightarrow \rho/(\Sigma, \theta_1, \theta_2) \qquad \rho \text{ is the mslg of } \rho_1 \text{ and } \rho_2$$

Intuitively, we will be able to obtain ρ_1 by composing θ_1 with ρ, and we yield ρ_2 by composing θ_2 with ρ. We assume ρ_1 and ρ_2 are well-typed, and have the domain Σ_2 and range (Σ_1, Δ).

We think of ρ_1 as the contextual substitution which is already in the index, while the contextual substitution ρ_2 is to be inserted. As a consequence, only ρ_1 will refer to the internal meta-variables in Σ_1, while ρ_2 only depends on the meta-variables

in Δ. The result of the mslg are the contextual substitution θ_1 and θ_2, where $\Delta, \Sigma_1 \vdash \theta_1 \Leftarrow \Sigma$ and $\Delta, \Sigma_1 \vdash \theta_2 \Leftarrow \Sigma$. In other words, θ_1 (resp. θ_2) only replaces internal meta-variables in Σ. Note that any contextual substitution ρ or θ with domain Σ, can be extended to a contextual substitution $(\mathsf{id}_\Delta, \rho)$ (or $(\mathsf{id}_\Delta, \theta)$ resp.) with domain (Δ, Σ).

First, we give the rules for computing the most specific linear generalization of two contextual substitutions.

$$\frac{}{(\Delta,\Sigma) \vdash \cdot \sqcup \cdot : \cdot \Longrightarrow \cdot/(\cdot,\cdot,\cdot)}$$

$$\frac{\begin{array}{llll}(\Delta,\Sigma_1) & \vdash \rho_1 \sqcup \rho_2 & : \Sigma_2 & \Longrightarrow \rho \,/\, (\Sigma_1', \theta_1, \theta_2) \\ (\Delta,\Sigma_1);\Psi & \vdash R_1 \sqcup R_2 & : P & \Longrightarrow R \,/\, (\Sigma_2', \theta_1', \theta_2') \\ \Sigma = (\Sigma_1', \Sigma_2') & \theta = (\theta_1, \theta_1') & \theta' = (\theta_2, \theta_2') & \end{array}}{(\Delta,\Sigma_1) \vdash (\rho_1, \hat{\Psi}.R_1/i) \sqcup (\rho_2, \hat{\Psi}.R_2/i) : (\Sigma_2, i{::}P[\Psi]) \Longrightarrow (\rho, \hat{\Psi}.R/i) \,/\, (\Sigma, \theta, \theta'))}$$

Note, that we are allowed to just combine the contextual substitutions θ_1 (θ_2 resp.) and θ_1' (θ_2' resp.) since we require that they refer to distinct meta-variables and all the meta-variables occur uniquely.

Similar to the compatibility of two terms, we can define the compatibility of two substitutions.

Definition 5.4 *Compatibility of contextual substitutions.*
If $(\Delta,\Sigma_1) \vdash \rho_1 \sqcup \rho_2 : \Sigma_2 \Longrightarrow \mathsf{id}_{\Sigma_1}/(\Sigma, \rho_1, \rho_2)$, then two contextual substitutions ρ_1 and ρ_2 are incompatible. Otherwise, we call ρ_1 and ρ_2 compatible.

As a consequence, if ρ_1 and ρ_2 are incompatible, then for any $\hat{\Psi}.R/i \in \rho_1$ and $\hat{\Psi}.R'/i \in \rho_2$, we know that R and R' are incompatible. Next, we prove soundness and completeness of this algorithm.

Theorem 5.5 Soundness for mslg of substitutions.
If $(\Delta,\Sigma_1) \vdash \rho_1 \sqcup \rho_2 : \Sigma_2 \Longrightarrow \rho/(\Sigma, \theta_1, \theta_2)$ *and*
$(\Delta,\Sigma_1) \vdash \rho_1 \Leftarrow \Sigma_2$ *and* $(\Delta,\Sigma_1) \vdash \rho_2 \Leftarrow \Sigma_2$
then $(\Delta,\Sigma) \vdash \rho \Leftarrow \Sigma_2$, $(\Delta,\Sigma_1) \vdash \theta_1 \Leftarrow \Sigma$, $(\Delta,\Sigma_1) \vdash \theta_2 \Leftarrow \Sigma$, *and*
$[\![\theta_1]\!]\rho = \rho_1$ *and* $[\![\theta_2]\!]\rho = \rho_2$

Proof. Induction on the first derivation.

Case. $\mathcal{D} = (\Delta,\Sigma_1) \vdash \cdot : \cdot \Longrightarrow \cdot/(\cdot,\cdot,\cdot)$

$\cdot = \cdot$ by syntactic equality
$\cdot = [\![\cdot]\!](\cdot)$ contextual substitution definition

Case. $\mathcal{D} = (\Delta,\Sigma_1) \vdash (\rho_1, \hat{\Psi}.R_1/i) \sqcup (\rho_2, \hat{\Psi}.R_2/i) : (\Sigma_2, i{::}P[\Psi]) \Longrightarrow (\rho, \hat{\Psi}.R/i)/((\Sigma,\Sigma'),(\theta_1,\theta_1'),(\theta_2,\theta_2'))$

$(\Delta,\Sigma_1) \vdash \rho_1 \sqcup \rho_2 : \Sigma_2 \Longrightarrow \rho/(\Sigma,\theta_1,\theta_2)$ by premise
$(\Delta,\Sigma_1);\Psi \vdash R_1 \sqcup R_2 : P \Longrightarrow R/(\Sigma',\theta_1',\theta_2')$ by premise
$(\Delta,\Sigma_1) \vdash (\rho_1, \hat{\Psi}.R_1/i) \Leftarrow (\Sigma_2, i{::}P[\Psi])$ by assumption
$(\Delta,\Sigma_1) \vdash \rho_1 \Leftarrow \Sigma_2$ by inversion
$(\Delta,\Sigma_1);\Psi \vdash R_1 \Leftarrow P$
$(\Delta,\Sigma_1) \vdash (\rho_2, \hat{\Psi}.R_2/i) \Leftarrow (\Sigma_2, i{::}P[\Psi])$ by assumption
$(\Delta,\Sigma_1) \vdash \rho_2 \Leftarrow \Sigma_2$ by inversion
$(\Delta,\Sigma_1);\Psi \vdash R_2 \Leftarrow P$

$(\Delta, \Sigma'); \Psi \vdash R \Leftarrow P$ by soundness theorem 5.2
$R_1 = [\![\theta_1']\!]R,\ \Delta, \Sigma_1 \vdash \theta_1' \Leftarrow \Sigma'$ by soundness theorem 5.2
$R_2 = [\![\theta_2']\!]R,\ \Delta, \Sigma_1 \vdash \theta_2' \Leftarrow \Sigma'$ by soundness theorem 5.2
$R_1 = [\![\theta_1, \theta_1']\!]R$ by weakening
$R_2 = [\![\theta_2, \theta_2']\!]R$ by weakening
$\rho_1 = [\![\theta_1]\!]\rho$ by i.h.
$\rho_2 = [\![\theta_2]\!]\rho$ by i.h.
$\rho_1 = [\![\theta_1, \theta_1']\!]\rho$ by weakening lemma
$\rho_2 = [\![\theta_2, \theta_2']\!]\rho$ by weakening lemma
$(\rho_1, \hat{\Psi}.R_1/i) = ([\![\theta_1, \theta_1']\!]\rho,\ [\![\theta_1, \theta_1']\!]\hat{\Psi}.R/i)$ by rule
$(\rho_2, \hat{\Psi}.R_2/i) = ([\![\theta_2, \theta_2']\!]\rho,\ [\![\theta_2, \theta_2']\!]\hat{\Psi}.R/i)$ by rule
$(\rho_1, \hat{\Psi}.R_1/i) = [\![\theta_1, \theta_1']\!](\rho, \hat{\Psi}.R/i)$ by contextual substitution definition
$(\rho_2, \hat{\Psi}.R_2/i) = [\![\theta_2, \theta_2']\!](\rho, \hat{\Psi}.R/i)$ by contextual substitution definition
$(\Delta, \Sigma) \vdash \rho \Leftarrow \Sigma_2$ by i.h.
$(\Delta, \Sigma, \Sigma') \vdash \rho \Leftarrow \Sigma_2$ by weakening
$(\Delta, \Sigma, \Sigma'); \Psi \vdash R \Leftarrow P$ by weakening
$(\Delta, \Sigma, \Sigma') \vdash (\rho, \hat{\Psi}.R/i) \Leftarrow (\Sigma_2, i{::}P[\Psi])$ by rule
$\Delta, \Sigma_1) \vdash (\theta_1, \theta_1') \Leftarrow (\Sigma, \Sigma')$ by typing rules
$\Delta, \Sigma_1) \vdash (\theta_2, \theta_2') \Leftarrow (\Sigma, \Sigma')$ by typing rules □

THEOREM 5.6 COMPLETENESS FOR MSLG OF CONTEXTUAL SUBSTITUTIONS. *If* $(\Delta, \Sigma) \vdash \theta_1 \Leftarrow \Sigma'$ *and* $(\Delta, \Sigma) \vdash \theta_2 \Leftarrow \Sigma'$ *and* θ_1 *and* θ_2 *are incompatible and* $\rho_1 = [\![\theta_1]\!]\rho$ *and* $\rho_2 = [\![\theta_2]\!]\rho$ *then* $(\Delta, \Sigma) \vdash \rho_1 \sqcup \rho_2 : \Sigma_1 \Longrightarrow \rho/(\Sigma^*, \theta_1^*, \theta_2^*)$ *such that* $\Sigma^* \subseteq \Sigma'$, $\theta_1^* \subseteq \theta_1$, $\theta_2^* \subseteq \theta_2$.

PROOF. Induction on the structure of ρ.

Case. $\rho = \cdot$

$\rho_1 = [\![\theta_1]\!](\cdot)$ by assumption
$\rho_1 = \cdot$ and $\Sigma_1 = \cdot$ by inversion
$\rho_2 = [\![\theta_2]\!](\cdot)$ by assumption
$\rho_2 = \cdot$ and $\Sigma_1 = \cdot$ by inversion
$(\Delta, \Sigma) \vdash \cdot \sqcup \cdot : \cdot \Longrightarrow \cdot/(\cdot, \cdot, \cdot)$ by rule
$\cdot \subseteq \Sigma_1,\ \cdot \subseteq \theta_1,\ \cdot \subseteq \theta_2$

Case. $\rho = (\rho', \hat{\Psi}.R/i)$

$\rho_1' = [\![\theta_1]\!](\rho', \hat{\Psi}.R/i)$
$\rho_1' = ([\![\theta_1]\!](\rho'),\ \hat{\Psi}.[\![\theta_1]\!]R/i)$ by contextual substitution definition
$\rho_1' = (\rho_1, \hat{\Psi}.R_1/i)$
$\rho_1 = [\![\theta_1]\!]\rho'$
$R_1 = [\![\theta_1]\!]R$
$\rho_2' = [\![\theta_2]\!](\rho', \hat{\Psi}.R/i)$
$\rho_2' = ([\![\theta_2]\!](\rho'),\ \hat{\Psi}.[\![\theta_2]\!]R/i)$ by contextual substitution definition
$\rho_2' = (\rho_2, \hat{\Psi}.R_2/i)$
$\rho_2 = [\![\theta_2]\!]\rho'$
$R_2 = [\![\theta_2]\!]R$
$(\Delta, \Sigma); \Psi \vdash R_1 \sqcup R_2 : P \Longrightarrow R/(\Sigma^*, \theta_1^*, \theta_2^*)$ by completeness lemma 5.3
$\Sigma^* \subseteq \Sigma',\ \theta_1^* \subseteq \theta_1,\ \theta_2^* \subseteq \theta_2$

$(\Delta, \Sigma) \vdash \rho_1 \sqcup \rho_2 : \Sigma_1 \Longrightarrow \rho'/(\Sigma^{**}, \theta_1^{**}, \theta_2^{**})$ by i.h.
$\Sigma^{**} \subseteq \Sigma'$, $\theta_1^{**} \subseteq \theta_1$, $\theta_2^{**} \subseteq \theta_2$
$(\Delta, \Sigma) \vdash (\rho_1, \hat{\Psi}.R_1/i) \sqcup (\rho_2, \hat{\Psi}.R_2/i) : (\Sigma_1, i{::}P[\Psi])$
$\Longrightarrow (\rho', \hat{\Psi}.R/i)/((\Sigma^{**}, \Sigma^*),\ (\theta_1^{**}, \theta_1^*),\ (\theta_2^{**}, \theta_2^*))$ by rule
$(\Sigma^{**}, \Sigma^*) \subseteq \Sigma'$, $(\theta_1^{**}, \theta_1^*) \subseteq \theta_1$, $(\theta_2^{**}, \theta_2^*) \subseteq \theta_2$ □

5.3 Insertion into substitution tree

In this Section we describe the final layer, namely the traversal of the substitution tree to insert a substitution δ. To insert the contextual substitution δ into a substitution tree, we need to traverse the index tree and compute at each node N with substitution ρ the mslg between ρ and δ. Before we describe the traversal more formally, we give a more formal definition of substitution trees.

$$\begin{array}{lll} \text{Node N} & ::= & (\Sigma \vdash \rho \twoheadrightarrow C) \\ \text{Children C} & ::= & [N, C] \mid \mathsf{nil} \end{array}$$

A tree is a node N with a contextual substitution ρ and a list of children C. If the list of children is empty, we have reached a leaf. In general, we will write $\Delta \vdash N : \Sigma'$ where $N = (\Sigma \vdash \rho \twoheadrightarrow C)$ which means that $(\Delta, \Sigma) \vdash \rho : \Sigma'$ and all the children N_i in C, $\Delta \vdash N_i : \Sigma$.

To insert a new substitution δ into the substitution tree N, we proceed in two steps. First, we inspect all the children N_i of a parent node N, where $N_i = \Sigma_i \vdash \rho_i \twoheadrightarrow C_i$ and check if ρ_i is compatible with δ. This compatibility check has three possible results:

(1) $(\Delta, \Sigma_i) \vdash \rho_i \sqcup \delta : \Sigma \Longrightarrow \mathsf{id}_\Sigma/(\Sigma', \rho_i, \delta)$:
This means ρ_i and δ are not compatible, and id_Σ is just a renaming of the meta-variables in Σ to Σ'.

(2) $(\Delta, \Sigma_i) \vdash \rho_i \sqcup \delta : \Sigma \Longrightarrow \rho_i/(\Sigma_i, \mathsf{id}_{\Sigma_i}, \theta_2)$
This means δ is an instance of ρ_i and we continue to insert θ_2 into the children C_i. In this case $[\![\theta_2]\!]\delta$ is equivalent to ρ_i and we call δ *fully compatible* with ρ_i.

(3) $(\Delta, \Sigma_i) \vdash \rho_i \sqcup \delta : \Sigma \Longrightarrow \rho'/(\Sigma'', \theta_1, \theta_2)$
ρ_i and δ are compatible, but we need to replace node N_i with a new node $\Sigma'' \vdash \rho' \twoheadrightarrow C'$ where C' contains two children, the child node $\Sigma_i \vdash \theta_1 \twoheadrightarrow C_i$ and the child node $\Sigma_i \vdash \theta_2 \twoheadrightarrow \mathsf{nil}$. In this case we call δ *partially compatible* with ρ_i.

The idea is to iterate through the list of children and collect all nodes which are fully compatible or at least partially compatible. If there is a fully compatible child N, we continue to our insertion considering following node N. If there are no fully compatible children but a partially compatible node N we need to split N. If no node is compatible, we simply create a new node with the substitution we intended to insert. To simplify the algorithms we only consider trees with at least one entry and a default identity substitution at the root. The substitution tree which contains as the only entry the term $\hat{\Psi}.R$ for example, is a tree with the default identity substitution $\hat{\Psi}.i_0[\mathsf{id}]/i_0$ at the root and one child with the substitution $\hat{\Psi}.R/i_0$.

The filtering process to collect all nodes which are fully and partially compatible can be formalized by using the following judgment. We will distinguish between

the fully compatible children, which we collect in V, and the partially compatible children, which we collect in S.

$$\begin{array}{lll} \text{Fully compatible children} & V & ::= \cdot \mid V, (N,\, \theta) \\ \text{Partially compatible children} & S & ::= \cdot \mid S,\, (N, \Sigma \vdash \rho, \theta_1, \theta_2) \end{array}$$

Note that it is not quite enough to just identify the children nodes N which are fully compatible or partially compatible. Instead, we need to track more information. For example, if we identify a child node N where $N = \Sigma' \vdash \rho_i \twoheadrightarrow C_i$ is fully compatible with δ, then this means that δ is an instance of ρ_i and there exists a contextual substitution θ s.t. $[\![\theta]\!]\rho_i = \delta$. Similarly, if we identify a child node N where $N = \Sigma' \vdash \rho_i \twoheadrightarrow C_i$ is partially compatible with δ, then this means that mslg between ρ_i and δ is the contextual substitution ρ, and $\rho_i = [\![\theta_1]\!]\rho$ and $\delta = [\![\theta_2]\!]\rho$.

Then we can define the following judgment:

$$\Delta \vdash C \sqcup \delta : \Sigma \Longrightarrow (V, S)$$

Given some children C and a contextual substitution δ, where the domain of each child in C and of the contextual substitution δ is Σ, we can compute fully compatible children V and the partially compatible children S.

δ is a contextual substitution such that $\Delta \vdash \delta \Leftarrow \Sigma$, and for all the children $C_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C')$ in C, we have $\Delta, \Sigma_i \vdash \rho_i \Leftarrow \Sigma$. Then V and S describe all the children from C which are compatible with δ. We distinguish three cases.

$$\frac{}{\Delta \vdash \mathsf{nil} \sqcup \delta : \Sigma \Longrightarrow (\mathsf{nil}, \mathsf{nil})}$$

$$\frac{\Delta \vdash C \sqcup \delta : \Sigma \Longrightarrow (V, S) \quad \Delta, \Sigma_1 \vdash \rho_1 \sqcup \delta : \Sigma \Longrightarrow \mathsf{id}_\Sigma/(\Sigma, \rho_1, \delta)}{\Delta \vdash [(\Sigma_1 \vdash \rho_1 \twoheadrightarrow C_1), C] \sqcup \delta : \Sigma \Longrightarrow (V, S)}\ NC$$

$$\frac{\Delta \vdash C \sqcup \delta : \Sigma \Longrightarrow (V, S) \quad \Delta, \Sigma_1 \vdash \rho_1 \sqcup \delta : \Sigma \Longrightarrow \rho_1/(\Sigma_1, \mathsf{id}_{\Sigma_1}, \theta_2) \quad \rho_1 \neq \mathsf{id}_{\Sigma_1}}{\Delta \vdash [(\Sigma_1 \vdash \rho_1 \twoheadrightarrow C_1), C] \sqcup \delta : \Sigma \Longrightarrow ((V,\ ((\Sigma_1 \vdash \rho_1 \twoheadrightarrow C_1), \theta_2)),\ S)}\ FC$$

$$\frac{\Delta \vdash C \sqcup \delta : \Sigma \Longrightarrow (V, S) \quad \Delta, \Sigma_1 \vdash \rho_1 \sqcup \delta : \Sigma \Longrightarrow \rho/(\Sigma_2, \theta_1, \theta_2) \quad \rho \neq \rho_1 \neq \mathsf{id}_{\Sigma_2}}{\Delta \vdash [(\Sigma_1 \vdash \rho_1 \twoheadrightarrow C_1), C] \sqcup \delta : \Sigma \Longrightarrow (V, (S\ ,\ ((\Sigma_1 \vdash \rho_1 \twoheadrightarrow C_1), \Sigma_2 \vdash \rho, \theta_1, \theta_2))}\ PC$$

The NC rule describes the case where the child C_i is not compatible with δ. Rule FC describes the case where δ is fully compatible with the child C_i and the rule PC describes the case where δ is partially compatible with C_i. Before we describe the traversal of the substitution tree, we prove some straightforward soundness properties about these rules. The first lemma essentially states that δ is an instance of all nodes collected in V. Moreover, for every node N_i with substitution ρ_i in S, there exists a rho' which is the most specific generalization of ρ_i and δ.

Lemma 5.7.
If $\Delta \vdash C \sqcup \delta : \Sigma \Longrightarrow (V, S)$ *and* $\Delta \vdash \delta : \Sigma$ *and for any* $(\Sigma_i \vdash \rho_i \twoheadrightarrow C') \in C$ *with* $\Delta, \Sigma_i \vdash \rho_i : \Sigma$ *then*

(1) for any $(N_i, \theta_2) \in V$ *where* $N_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C_i)$, *we have* $[\![\theta_2]\!]\rho_i = \delta$.

(2) *for any* $(N_i, \Sigma' \vdash \rho', \theta_1, \theta_2) \in S$ *where* $N_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C_i)$, *we have* $[\![\theta_2]\!]\rho' = \delta$ *and* $[\![\theta_1]\!]\rho' = \rho_i$.

PROOF. By structural induction on the first derivation and by using the previous soundness lemma for mslg of substitutions (lemma 5.5).

Case. $\mathcal{D} = \dfrac{}{\Delta \vdash \mathsf{nil} \sqcup \delta : \Sigma \Longrightarrow (\mathsf{nil}, \mathsf{nil})}$.

Trivially true.

$$\textit{Case. } \mathcal{D} = \frac{\begin{array}{l}\Delta \quad\;\; \vdash C \sqcup \delta : \Sigma \implies (V, S) \\ \Delta, \Sigma_1 \vdash \rho_1 \sqcup \delta : \Sigma \implies \mathsf{id}_\Sigma/(\Sigma, \rho_1, \delta)\end{array}}{\Delta \vdash [(\Sigma_1 \vdash \rho_1 \twoheadrightarrow C_1), C] \sqcup \delta : \Sigma \Longrightarrow (V, S)}\, NC$$

By i.h., for any $(N_i, \theta_2) \in V$, $N_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C_i)$, we have $[\![\theta_2]\!]\rho_i = \delta$ and for any $(N_i, \Sigma' \vdash \rho', \theta_1', \theta_2') \in S$ where $N_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C_i)$, we have $[\![\theta_2']\!]\rho' = \delta$ and $[\![\theta_1']\!]\rho' = \rho_i$.

$$\textit{Case. } \mathcal{D} = \frac{\begin{array}{l}\Delta \quad\;\; \vdash C \sqcup \delta : \Sigma \implies (V, S) \\ \Delta, \Sigma_1 \vdash \rho_1 \sqcup \delta : \Sigma \implies \rho_1/(\Sigma_1, \mathsf{id}_{\Sigma_1}, \theta_2)\end{array}}{\Delta \vdash [(\Sigma_1 \vdash \rho_1 \twoheadrightarrow C_1), C] \sqcup \delta : \Sigma \Longrightarrow ((V\ ,\ (\Sigma_1 \vdash \rho_1 \twoheadrightarrow C_1)), S)}\, FC$$

By i.h., for any $(N_i, \theta_2) \in V$, $N_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C_i)$, we have $[\![\theta_2]\!]\rho_i = \delta$ and for any $(N_i, (\Sigma' \vdash \rho', \theta_1', \theta_2')) \in S$ where $N_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C_i)$, we have $[\![\theta_2']\!]\rho' = \delta$ and $[\![\theta_1']\!]\rho' = \rho_i$. By soundness lemma 5.5, $[\![\theta_2]\!]\rho_1 = \delta$, therefore for any $(N_i, \theta') \in (V,\ ((\Sigma_1 \vdash \rho_1 \twoheadrightarrow C_1), \theta_2))$, where $N_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C_i)$ we have $[\![\theta']\!]\rho_i = \delta$.

Case.

$$\mathcal{D} = \frac{\begin{array}{l}\Delta \quad\;\; \vdash C \sqcup \delta : \Sigma \implies (V, S) \\ \Delta, \Sigma_1 \vdash \rho_1 \sqcup \delta : \Sigma \implies \rho^*/(\Sigma_2, \theta_1, \theta_2)\end{array}}{\Delta \vdash [(\Sigma_1 \vdash \rho_1 \twoheadrightarrow C_1), C] \sqcup \delta : \Sigma \Longrightarrow (V, (S\ ,\ ((\Sigma_1 \vdash \rho_1 \twoheadrightarrow C_1), \Sigma_2 \vdash \rho^*, \theta_1, \theta_2))}\, PC$$

By i.h., for any $(N_i, \theta_2') \in V$, $N_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C_i)$, we have $[\![\theta_2']\!]\rho_i = \delta$ and for any $(N_i, (\Sigma' \vdash \rho', \theta_1', \theta_2')) \in S$ where $N_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C_i)$, we have $[\![\theta_2']\!]\rho' = \delta$ and $[\![\theta_1']\!]\rho' = \rho_i$. By soundness lemma 5.5, $[\![\theta_2]\!]\rho^* = \delta$ and $[\![\theta_1]\!]\rho^* = \rho_1$, therefore for any $(N_i, \Sigma' \vdash \rho', \theta_1', \theta_2') \in (S\ ,\ ((\Sigma_1 \vdash \rho_1 \twoheadrightarrow C_1), \Sigma_2 \vdash \rho^*, \theta_1, \theta_2))$, where $N_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C_i)$ we have $[\![\theta_1']\!]\rho' = \rho_i$ and $[\![\theta_2']\!]\rho' = \delta$. □

Next, we show insertion of a substitution δ into a substitution tree N. The main judgment is the following:

$$\Delta \vdash N \sqcup \delta : \Sigma \Longrightarrow N' \;\text{ Insert } \delta \text{ into the substitution tree } N$$

If N is a substitution tree and δ is not already in the tree, then N' will be the new substitution tree after inserting δ into N. We write $[N_i'/N_i]C$ to indicate that the i-th node N_i in the children C is replaced by the new node N_i'. Recall that the substitution δ which is inserted into the substitution tree N does only refer to meta-variables in Δ and does not contain any internal meta-variables. Therefore, a new leaf with substitution δ must have the following form: $\cdot \vdash \delta \twoheadrightarrow \mathsf{nil}$. Similarly, if we split the current node and create a new leaf $\cdot \vdash \theta_2 \twoheadrightarrow \mathsf{nil}$ (see rule "Split current

node").

Create new leaf

$$\frac{\Delta \vdash C \sqcup \delta : \Sigma \Longrightarrow (\cdot, \cdot)}{\Delta \vdash (\Sigma' \vdash \rho \twoheadrightarrow C) \sqcup \delta : \Sigma \Longrightarrow (\Sigma' \vdash \rho \twoheadrightarrow (C, (\cdot \vdash \delta \twoheadrightarrow \mathsf{nil}))}$$

Recurse

$$\frac{\Delta \vdash C \sqcup \delta : \Sigma \Longrightarrow (V, S) \quad N_i \in C \quad (N_i, \theta_2) \in V \quad \Delta \vdash N \sqcup \theta_2 \Longrightarrow N'}{\Delta \vdash (\Sigma' \vdash \rho \twoheadrightarrow C) \sqcup \delta : \Sigma \Longrightarrow (\Sigma' \vdash \rho \twoheadrightarrow [N'/N_i]C}$$

Split current node

$$\frac{\Delta \vdash C \sqcup \delta : \Sigma \Longrightarrow (\cdot, S) \quad N_i \in C \quad N_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C_i) \quad (N_i, \Sigma^* \vdash \rho, \theta_1, \theta_2) \in S}{\Delta \vdash (\Sigma' \vdash \rho \twoheadrightarrow C) \sqcup \delta : \Sigma \Longrightarrow (\Sigma' \vdash \rho \twoheadrightarrow [(\Sigma^* \vdash \rho \twoheadrightarrow ((\Sigma_i \vdash \theta_1 \twoheadrightarrow C_i), (\cdot \vdash \theta_2 \twoheadrightarrow \mathsf{nil})))/N_i]C)}$$

The above rules always insert a substitution δ into the children C of a node $\Sigma \vdash \rho \twoheadrightarrow C$. We start inserting a substitution $\hat{\Psi}.R/i_0$ into the empty substitution tree which contains the identity substitution $\hat{\Psi}.i_0[\mathsf{id}]/i_0$ and has an empty list of children. After the first insertion, we obtain the substitution tree which contains the identity substitution $\hat{\Psi}.i_0[\mathsf{id}]/i_0$ and the child of this node contains the substitution $\hat{\Psi}.R/i_0$. In other words, we require that the top node of a substitution tree contains a redundant identity substitution which allows us to treat insertion of a substitution δ into a substitution tree uniformly. This leads us to the following soundness statement where we show that if we insert a substitution δ into the children C, then there exists a child $C_i = \Sigma_i \vdash \rho_i \twoheadrightarrow C_i'$ in C and a path from ρ_i to ρ_n, where ρ_n is at a leaf such that $[\![\rho_n]\!][\![\rho_{n-1}]\!] \ldots \rho_i = \delta$.

THEOREM 5.8 SOUNDNESS OF INSERTION.
If $\Delta \vdash (\Sigma' \vdash \rho' \twoheadrightarrow C) \sqcup \delta : \Sigma \Longrightarrow (\Sigma' \vdash \rho' \twoheadrightarrow C')$ *then there exists a child* $C_i = (\Sigma_i \vdash \rho_i \twoheadrightarrow C')$ *and a path from* ρ_i *to* ρ_n *such that* $[\![\rho_n]\!][\![\rho_{n-1}]\!] \ldots \rho_i = \delta$.

PROOF. By induction on the first derivation using the previous lemma 5.7. □

6. RETRIEVAL

In general, we can retrieve all terms from the index which satisfy some property. This property may be finding all terms from the index which unify with a given term M or finding all terms N from the index, s.t. a given term M is an instance or variant of N. One advantage of substitution trees is that all these retrieval operations can be implemented with only small changes. A key challenge in the higher-order setting is to design efficient retrieval algorithms which will perform well in practice. We will show three retrieval algorithms: Variant, unifiable and Instance. All the algorithms can be described as variances from each other.

6.1 Instance checking for linear higher-order patterns

First, we presented an instance checking algorithm for linear higher-order patterns. We treat again internal meta-variables differently than global meta-variables. The principal judgment are as follows:

$$\begin{array}{ll}
\Delta_2; (\Delta, \Sigma); \Gamma \vdash M_1 \doteq M_2 : A/(\theta, \rho) & M_2 \text{ is an instance of } M_1 \\
\Delta_2; (\Delta, \Sigma); \Gamma \vdash R_1 \doteq R_2 : P/(\theta, \rho) & R_2 \text{ is an instance of } R_1 \\
\Delta_2; (\Delta, \Sigma); \Gamma \vdash S_1 \doteq S_2 : A > P/(\theta, \rho) & S_2 \text{ is an instance of } S_2
\end{array}$$

Again we assume that M_1 (R_1, S_1 resp.) must be well-typed in the modal context Δ, Σ and the bound variable context Γ. M_2 (R_2, S_2 resp.) are well-typed in the modal context Δ_2 and the bound variable context Γ. In other words M_1 only contains internal meta-variables and is stored in the index, while M_2 is given, and that the meta-variables occurring in M_1 are distinct from the meta-variables occurring in M_2. More formally this is stated as $(\Delta_1, \Sigma); \Gamma \vdash M_1 \Leftarrow A$ and $\Delta_2; \Gamma \vdash M_2 \Leftarrow A$ and $\Delta = (\Delta_2, \Delta_1)$. ρ is the substitution for some internal meta-variables in Σ while θ is the substitution for some meta-variables in Δ_1. Moreover, we maintain that $[\![\theta, \rho]\!]M_1$ is syntactically equal to M_2. We will treat Σ as a linear context in the formal description given below. This will make it easier to prove the relationship between insertion and retrieval later on.

$$\frac{\Delta_2; (\Delta, \Sigma); \Gamma, x{:}A_1 \vdash M_1 \doteq M_2 : A_2 \;/\; (\theta, \rho)}{\Delta_2; (\Delta, \Sigma); \Gamma \vdash \lambda x.M_1 \doteq \lambda x.M_2 : A_1 \to A_2 \;/\; (\theta, \rho)}\ \mathsf{lam}$$

$$\frac{\Delta_2; (\Delta, \Sigma); \Gamma \vdash R_1 \doteq R_2 : P \;/\; (\theta, \rho)}{\Delta_2; (\Delta, \Sigma); \Gamma \vdash R_1 \doteq R_2 : P \;/\; (\theta, \rho)}\ \mathsf{coe}$$

$$\frac{}{\Delta_2; (\Delta, i{::}P[\Gamma]); \Gamma \vdash i[\mathsf{id}_\Gamma] \doteq R : P \;/\; (\cdot, (\hat{\Gamma}.R/i))}\ \mathsf{meta\text{-}1}$$

$$\frac{u{::}P[\Phi] \in \Delta}{\Delta_2; (\Delta, \Sigma); \Gamma \vdash u[\pi] \doteq R : P \;/\; (\hat{\Gamma}.([\pi]^{-1}\, R/u), \cdot)}\ \mathsf{meta\text{-}2}$$

$$\frac{\Delta_2; (\Delta, \Sigma); \Gamma \Vdash S_1 \doteq S_2 : A > P \;/\; (\theta, \rho)}{\Delta_2; (\Delta, \Sigma); \Gamma \vdash H \cdot S_1 \doteq H \cdot S_2 : a \;/\; (\theta, \rho)}\ \mathsf{head}$$

$$\frac{}{\Delta_2; (\Delta, \cdot); \Gamma \Vdash \mathsf{nil} \doteq \mathsf{nil} : P > P \;/\; (\cdot, \cdot)}\ \mathsf{Snil}$$

$$\frac{\begin{array}{c}\Delta_2; (\Delta, \Sigma_1); \Gamma \vdash M_1 \doteq M_2 : A_1 \;/\; (\theta_1, \rho_1) \\ \Delta_2; (\Delta, \Sigma_2); \Gamma \Vdash S_1 \doteq S_2 : A_2 > P \;/\; (\theta_2, \rho_2)\end{array}}{\Delta_2; (\Delta, \Sigma_1, \Sigma_2); \Gamma \Vdash (M_1; S_1) \doteq (M_2; S_2) : A_1 \to A_2 > P \;/\; ((\theta_1, \theta_2),\ (\rho_1, \rho_2))}\ \mathsf{Scons}$$

Note that we need not worry about capture in the rule for lambda expressions since meta-variables and bound variables are defined in different contexts. In the rule Scons, we are allowed to union the two substitutions θ_1 and θ_2, as the linearity requirement ensures that the domains of both substitutions are disjoint[2]. Note

[2]It is possible to formalize the linearity criteria on the meta-variables declared in Δ in the same way we enforce linearity on the internal meta-variables in Σ.

that the case for matching an meta-variable $u[\pi]$ with another term R is simpler and more efficient than in the general higher-order pattern case. In particular, it does not require a traversal of R (see rules meta-1 and meta-2). Since the inverse of the substitution π can be computed directly and will be total, we know $[\pi]^{-1}\,R$ exists and can simply generate a substitution $\hat{\Gamma}.[\pi]^{-1}\,R/u$. The algorithm can be easily specialized to retrieve variances by requiring in the meta-2 rule that R must be $u[\pi]$. To check unifiability we need to add the dual rule to meta-2 where we unify R with an meta-variable $u[\pi]$. The only complication is that R may contain internal meta-variables which are defined later on the path in the substitution tree. Now we show soundness and completeness of the retrieval algorithm. We first show soundness and completeness of the instance algorithm for terms.

THEOREM 6.1 SOUNDNESS.

(1) *If* $\Delta_2; (\Delta_1, \Sigma); \Gamma \vdash M_1 \doteq M_2 : A/(\theta, \rho)$ *for some* Δ_1 *and* Δ_2 *where* $(\Delta_1, \Sigma); \Gamma \vdash M_1 \Leftarrow A$ *and* $\Delta_2; \Gamma \vdash M_2 \Leftarrow A$ *then* $[\![\theta, \rho]\!]M_1 = M_2$.

(2) *If* $\Delta_2; (\Delta_1, \Sigma); \Gamma \vdash R_1 \doteq R_2 : P/(\theta, \rho)$ *where* $(\Delta_1, \Sigma); \Gamma \vdash R_1 \Rightarrow P_1$ *and* $\Delta_2; \Gamma \vdash R_2 \Rightarrow P_2$ *and* $P_1 = P_2 = P$ *then* $[\![\theta, \rho]\!]R_1 = R_2$.

(3) *If* $\Delta_2; (\Delta_1, \Sigma); \Gamma \vdash S_1 \doteq S_2 > A \Rightarrow P/(\theta, \rho)$ *where* $(\Delta_1, \Sigma); \Gamma \vdash S_1 > A \Rightarrow P$ *and* $\Delta_2; \Gamma \vdash S_2 > A \Rightarrow P$ *then* $[\![\theta, \rho]\!]S_1 = S_2$.

PROOF. Simultaneous structural induction on the first derivation. The proof is straightforward, and we give a few cases here.

$$\textit{Case. } \mathcal{D} = \frac{}{\Delta_2; (\Delta_1, i{::}P[\Gamma]); \Gamma \vdash i[\mathsf{id}_\Gamma] \doteq R : P \;/\; (\cdot, (\hat{\Gamma}.R/i))}\;\textsf{mvar-1}$$

$(\Delta_1, i{::}P[\Gamma]); \Gamma \vdash i[\mathsf{id}_\Gamma] : P$ by assumption
$\Delta_2; \Gamma \vdash R \Rightarrow P_1$ by assumption
$R = R$ by reflexivity
$[\![\hat{\Psi}.R/i]\!](i[\mathsf{id}_\Gamma]) = R$ by substitution definition

$$\textit{Case. } \mathcal{D} = \frac{u{::}\Phi{\vdash}P \in \Delta}{(\Delta, \Sigma); \Gamma \vdash u[\pi] \doteq R : P \;/\; (\hat{\Gamma}.([\pi]^{-1}\,R/u), \cdot)}\;\textsf{mvar-2}$$

$\Delta_1; \Gamma \vdash u[\pi] \Rightarrow P$ where $u{::}\Phi{\vdash}P \in \Delta_1$ by assumption
$\Delta_2; \Gamma \vdash R \Rightarrow P_1$ and $P_1 = P$ by assumption
$[\pi]([\pi]^{-1}\,R) = R$ by property of inversion
$[\![\hat{\Gamma}.[\pi]^{-1}\,R/u]\!](u[\pi]) = R$ by substitution definition

$$\textit{Case. } \mathcal{D} = \frac{\Delta_2; (\Delta_1, \Sigma); \Gamma, x{:}A_1 \vdash M_1 \doteq M_2 : A_2 \;/\; (\theta, \rho)}{\Delta_2; (\Delta_1, \Sigma); \Gamma \vdash \lambda x.M_1 \doteq \lambda x.M_2 : A_1 \to A_2 \;/\; (\theta, \rho)}\;\textsf{lam}$$

$(\Delta_1, \Sigma); \Gamma \vdash \lambda x.M_1 \Leftarrow A_1 \to A_2$ by assumption
$(\Delta_1, \Sigma); \Gamma, x{:}A_1 \vdash M_1 \Leftarrow A_2$ by inversion
$\Delta_2; \Gamma \vdash \lambda x.M_2 \Leftarrow A_1 \to A_2$ by assumption

$\Delta_2, ; \Gamma, x{:}A_1 \vdash M_2 \Leftarrow A_2$ by inversion
$[\![\theta, \rho]\!] M_1 = M_2$ by i.h.

Case. $\mathcal{D} = \dfrac{\Delta_2; (\Delta_1, \Sigma_1, \Sigma_2); \Gamma \Vdash (M_1; S_1) \doteq (M_2; S_2) : A_1 \to A_2 > P}{\Delta_2; (\Delta_1, \Sigma_1); \Gamma \vdash M_1 \doteq M_2 : A_1 \;/\; (\theta_1, \rho_1)} \;/\; ((\theta_1, \theta_2),\, (\rho_1, \rho_2))$

$\Delta_2; (\Delta_1, \Sigma_1); \Gamma \vdash M_1 \doteq M_2 : A_1 \;/\; (\theta_1, \rho_1)$
$\Delta_2; (\Delta_1, \Sigma_2); \Gamma \Vdash S_1 \doteq S_2 : A_2 > P \;/\; (\theta_2, \rho_2)$ by inversion
$(\Delta_1; \Sigma_1, \Sigma_2); \Gamma \vdash (M_1; S_1) < A_1 \to A_2 \Rightarrow P$ by assumption
$(\Delta_1; \Sigma_1); \Gamma \vdash M_1 \Leftarrow A_1$ by inversion
$(\Delta_1; \Sigma_2); \Gamma \vdash S_1 < A_2 \Rightarrow P$
$\Delta_2; \Gamma \vdash M_2 \Leftarrow A_1$ by inversion
$\Delta_2; \Gamma \vdash S_2 < A_2 \Rightarrow P$
$[\![\theta_1, \rho_1]\!] M_1 = M_2$ by i.h.
$[\![\theta_2, \rho_2]\!] S_1 = S_2$ by i.h.
$[\![\theta_1, \theta_2, \rho_1, \rho_2]\!] M_1 = M_2$ by weakening (using linearity condition)
$[\![\theta_1, \theta_2, \rho_1, \rho_2]\!] S_1 = S_2$ by weakening (using linearity condition)
$[\![\theta_1, \theta_2, \rho_1, \rho_2]\!] (M_1\ S_1) = [\![\mathsf{id}_{\Delta_2}\theta_1, \theta_2, \rho_1, \rho_2]\!] (M_2\ S_2)$ by rule and substitution definition □

For completeness we show that if the term M_2 is an instance of a linear term M then the given algorithm will succeed and return substitution θ^* for the meta-variables and a substitution ρ^* for the internal meta-variables occurring in M. This establishes a form of local completeness of the given retrieval algorithm. We will show later a global completeness theorem, which states that any time we compute the msgl of a term M_1 and M_2 to be M, then we can show that M_2 is in fact an instance of M. More generally, we show that any time we insert a substitution $\hat{\Gamma}.M_2/i_0$ we can also retrieve it.

Theorem 6.2 Completeness of instance algorithm for terms.

(1) *If* $(\Delta_1, \Sigma); \Gamma \vdash M_1 \Leftarrow A$ *and* $\Delta_2; \Gamma \vdash M_2 \Leftarrow A$ *and* $\Delta_2 \vdash \theta : \Delta_1$ *and* $\Delta_2 \vdash \rho : \Sigma$ *and* $[\![\theta, \rho]\!] M_1 = M_2$ *then* $\Delta_2; (\Delta_1, \Sigma); \Gamma \vdash M_1 \doteq M_2 : A/(\theta^*, \rho)$ *where* $\theta^* \subseteq \theta$.

(2) *If* $(\Delta_1, \Sigma); \Gamma \vdash R_1 \Rightarrow A$ *and* $\Delta_2; \Gamma \vdash R_2 \Rightarrow A$ *and* $\Delta_2 \vdash \theta : \Delta_1$ *and* $\Delta_2 \vdash \rho : \Sigma$ *and* $[\![\theta, \rho]\!] R_1 = R_2$ *then* $\Delta_2; (\Delta_1, \Sigma); \Gamma \vdash R_1 \doteq R_2 : A/(\theta^*, \rho)$ *where* $\theta^* \subseteq \theta$.

(3) *If* $(\Delta_1, \Sigma); \Gamma \vdash S_1 : A > P$ *and* $\Delta_2; \Gamma \vdash S_2 : A > P$ *and* $\Delta_2 \vdash \theta : \Delta_1$ *and* $\Delta_2 \vdash \rho : \Sigma$ *and* $[\![\theta, \rho]\!] S_1 = S_2$ *then* $\Delta_2; (\Delta_1, \Sigma); \Gamma \vdash S_1 \doteq S_2 : A > P/(\theta^*, \rho)$ *where* $\theta^* \subseteq \theta$.

Proof. Simultaneous structural induction on the first typing derivation.

Case. $\mathcal{D} = \dfrac{(\Delta_1, \Sigma); \Gamma, x{:}A_1 \vdash M_1 : A_2}{(\Delta_1, \Sigma); \Gamma \vdash \lambda x.M_1 : A_1 \to A_2}$

$\Delta_2; \Gamma \vdash \lambda x.M_2 \Leftarrow A_1 \to A_2$ by assumption
$\Delta_2; \Gamma, x{:}A_1 \vdash M_2 \Leftarrow A_2$ by inversion
$[\![\theta, \rho]\!] (\lambda x.M_1) = \lambda x.M_2$ by assumption
$\lambda x.[\![\theta, \rho]\!] (M_1) = \lambda x.M_2$ by substitution definition
$[\![\theta, \rho]\!] (M_1) = M_2$ by syntactic equality
$\Delta_2; (\Delta_1, \Sigma); \Gamma, x{:}A_1 \vdash M_1 \doteq M_2 : A_2/(\theta^*, \rho^*)$ by i.h.

$\theta^* \subseteq \theta$ and $\rho^* \subseteq \rho$
$\Delta_2; (\Delta_1, \Sigma); \Gamma \vdash \lambda x.M_1 \doteq \lambda x.M_2 : A_1 \to A_2/(\theta^*, \rho^*)$ by rule

Case. $\mathcal{D} = \dfrac{}{(\Delta_1, i{::}P[\Gamma]); \Gamma \vdash i[\mathsf{id}_\Gamma] \Rightarrow P}$

$i{::}P[\Gamma]; \Gamma \vdash i[\mathsf{id}_\Gamma] \Rightarrow P$ by rule
$\Delta_2; \Gamma \vdash R_2 \Rightarrow P$ by assumption
$[\![\theta, \rho]\!](i[\mathsf{id}_\Gamma]) = R_2$ by assumption
$\hat{\Gamma}.R_2/i \in \rho$ by assumption
$\Delta_2; (\Delta_1, i{::}P[\Gamma]); \Gamma \vdash i[\mathsf{id}_\Gamma] \doteq R_2 : P/(\cdot, \hat{\Gamma}.R_2/i)$ by rule
$\cdot \subseteq \mathsf{id}_\Delta$ and $(\hat{\Gamma}.R_2/i) \subset \rho$

Case. $\mathcal{D} = \dfrac{u{::}\Phi{\vdash}P \in \Delta_1}{(\Delta_1, \cdot); \Gamma \vdash u[\pi] \Rightarrow P}$

$u{::}\Phi{\vdash}P; \Gamma \vdash u[\pi] \Rightarrow P$ by rule
$\Delta_1 = \Delta_1', u{::}\Phi{\vdash}P, \Delta_1''$
$\Delta_2; \Gamma \vdash R_2 \Rightarrow P$ by assumption
$\theta = (\theta_1, \hat{\Gamma}.R/u, \theta_2)$ by assumption
$[\![\theta, \rho]\!](u[\pi]) = M_2$ by assumption
$[\pi]R = R_2$ by substitution definition
$R = [\pi]^{-1} R_2$ and $[\pi]([\pi]^{-1} R_2) = R_2$ by inverse substitution property
$\Delta_2, u{::}\Phi{\vdash}P; \Gamma \vdash u[\pi] \doteq R_2 : P/(\hat{\Gamma}.[\pi]^{-1} R_2/u, \cdot)$ by rule
$(\hat{\Gamma}.[\pi]^{-1} R_2/u) \subseteq \theta$ and $\cdot \subseteq \rho$

Case. $\mathcal{D} = \dfrac{(\Delta_1, \Sigma); \Gamma \vdash M_1 \Leftarrow A_1 \qquad (\Delta_1, \Sigma); \Gamma \Vdash S_1 : A_1 \to A > P}{(\Delta_1, \Sigma); \Gamma \Vdash (M_1; S_1) : A > P}$

$[\![\theta, \rho]\!](M_1; S_1) = S'$ by assumption
$[\![\theta, \rho]\!](M_1)\ ;\ [\![\theta, \rho]\!](S_1) = S'$ by substitution definition
$S' = (M_2; S_2)$ by inversion
$[\![\theta, \rho]\!](M_1) = M_2$ by inversion
$[\![\theta, \rho]\!](S_1) = S_2$ by inversion
$\Delta_2; \Gamma \vdash (M_2; S_2) : A > P$ by assumption
$\Delta_2; \Gamma \vdash M_2 \Leftarrow A_1$ by inversion
$\Delta_2; \Gamma \vdash S_2 : A_1 \to A > P$
$\Delta_2; (\Delta_1, \Sigma_1); \Gamma \vdash M_1 \doteq M_2 : A_1/(\theta_1^*, \rho_1)$ and $\theta_1^* \subseteq \theta$ by i.h.
$\Delta_2; (\Delta_1, \Sigma_2); \Gamma \vdash S_1 \doteq S_2 : A_1 \to A > P/(\theta_2^*, \rho_2)$ and $\theta_2^* \subseteq \theta$ by i.h.
$(\Delta, \Sigma); \Gamma \vdash (M_1; S_1) \doteq (M_2; S_2) : A > P/((\theta_1^*, \theta_2^*), (\rho_1, \rho_2))$ by rule
$(\theta_1^*, \theta_2^*) \subseteq \theta$ by subset property □

6.2 Instance checking for contextual substitutions

The instance algorithm for terms can be straightforwardly extended to instances of substitutions. We define the following judgment for it:

$$\Delta_2; (\Delta_1, \Sigma) \vdash \rho_1 \doteq \rho_2 : \Sigma'/(\theta, \rho)\ \ \rho_2 \text{ is an instance of } \rho_1$$

We assume that ρ_1 is a contextual substitution from a modal context Σ' to a modal context Δ, Σ, and and ρ_2 is a contextual substitution from Σ' to the modal context Δ_2. Our goal is to check whether ρ_2 is an instance of ρ_1. The result of this is the contextual substitution ρ for the meta-variables in Σ and the contextual substitution θ for the meta-variables in Δ such that $[\![\theta, \sigma]\!]\rho_1$ is syntactically equal to ρ_2. Again we enforce the linearity criteria for internal meta-variables in Σ but leave it implicit for the meta-variables in Δ_1.

$$\frac{}{\Delta_2; (\Delta_1, \cdot) \vdash \cdot \doteq \cdot : \cdot / (\cdot, \cdot)}$$

$$\frac{\Delta_2; (\Delta_1, \Sigma_1'') \vdash \rho_1 \doteq \rho_2 : \Sigma_2/(\theta, \rho) \quad \Delta_2; (\Delta_1, \Sigma_1'); \Gamma \vdash M_1 \doteq M_2 : A/(\theta', \rho')}{\Delta_2; (\Delta_1, \Sigma_1', \Sigma_2'') \vdash (\rho_1, \hat{\Psi}.R_1/i) \doteq (\rho_2, \hat{\Psi}.R_2/i) : (\Sigma_2, i::(\Gamma \vdash A))/((\theta, \theta'), (\rho, \rho'))}$$

Next, we show soundness of retrieval for substitutions.

THEOREM 6.3 SOUNDNESS OF RETRIEVAL FOR SUBSTITUTIONS.
If $(\Delta, \Sigma) \vdash \rho_1 \doteq \rho_2 : \Sigma'/(\theta, \rho)$ *and* $(\Delta_1, \Sigma) \vdash \rho_1 \Leftarrow \Sigma'$ *and* $\Delta_2 \vdash \rho_2 : \Sigma'$ *and* $(\Delta_1, \Delta_2) = \Delta$ *and all the variables in* Σ, Δ_1 *and* Δ_2 *are distinct then* $[\![\theta, \rho]\!]\rho_1 = \rho_2$.

PROOF. Structural induction on the first derivation and using previous lemma 6.1. □

Finally, we show the global completeness of the mslg and instance algorithm which relates insertion and retrieval. We show that if the mslg of object M_1 and M_2 returns the contextual substitutions θ_1 and θ_2 together with the mslg M, then in fact the retrieval algorithm shows that M_1 is an instance of L under θ_1 and M_2 is an instance of L under θ_2. This guarantees that any time we insert a term M_2 we can in fact retrieve it. We assume here that the set of meta-variables in M_1 is distinct from the set of meta-variables in M_2 which simplifies this proof slightly, since essentially the most specific generalization contains only internal meta-variables.

THEOREM 6.4 INTERACTION BETWEEN MSLG AND INSTANCE ALGORITHM.

(1) *If* $(\Delta_1, \Sigma); \Gamma \vdash M_1 \Leftarrow A$ *and* $\Delta_2; \Gamma \vdash M_2 \Leftarrow A$ *and* $(\Delta_2, \Delta_1), \Sigma; \Gamma \vdash M_1 \sqcup M_2 : A \Longrightarrow M/(\Sigma', \rho_1, \rho_2)$ *then*
$(\Delta_1; \Sigma'; \Gamma \vdash M \doteq M_1 : A/(\cdot, \rho_1)$ *and*
$\Delta_2; ; \Sigma'); \Gamma \vdash M \doteq M_2 : A/(\cdot, \rho_2)$.

(2) *If* $(\Delta_1, \Sigma); \Gamma \vdash R_1 \Rightarrow A_1$ *and* $\Delta_2; \Gamma \vdash R_2 \Rightarrow A_2$ *and* $A_1 = A_2 = A$ *and* $(\Delta_2, \Delta_1), \Sigma; \Gamma \vdash R_1 \sqcup R_2 : A \Longrightarrow R/(\Sigma', \rho_1, \rho_2)$ *then*
$\Delta_1; \Sigma'; \Gamma \vdash R \doteq R_1 : A/(\cdot, \rho_1)$ *and*
$\Delta_2; \Sigma'; \Gamma \vdash R \doteq R_2 : A/(\cdot, \rho_2)$.

(3) *If* $(\Delta_1, \Sigma); \Gamma \vdash S_1 > P : A$ *and* $\Delta_2; \Gamma \vdash S_2 : A > P$ *and* $(\Delta_2, \Delta_1), \Sigma; \Gamma \vdash S_1 \sqcup S_2 : A > P \Longrightarrow S/(\Sigma', \rho_1, \rho_2)$ *then*
$\Delta_1; \Sigma'; \Gamma \vdash S \doteq S_1 : A > P/(\cdot, \rho_1)$ *and*
$\Delta_2; \Sigma'; \Gamma \vdash S \doteq S_2 : A > P/(\cdot, \rho_2)$.

PROOF. Simultaneous structural induction on the first derivation. Let $\Delta = \Delta_2, \Delta_1$.

$$\textit{Case. } \mathcal{D} = \frac{(\Delta_2, \Delta_1, \Sigma); \Gamma, x{:}A_1 \vdash M_1 \sqcup M_2 : A_2 \Longrightarrow M/(\Sigma', \rho_1, \rho_2)}{(\Delta, \Sigma); \Gamma \vdash \lambda x..M_1 \sqcup \lambda x..M_2 : A_1 \to A_2 \Longrightarrow \lambda x..M/(\Sigma',\ \rho_1, \rho_2)}$$

$\Delta_1; \Sigma'; \Gamma, x{:}A_1 \vdash M \doteq M_1 : A_2/(\cdot, \rho_1)$ by i.h.
$\Delta_1; \Sigma'; \Gamma \vdash \lambda x.M \doteq \lambda x.M_1 : A_1 \to A_2/(\cdot, \rho_1)$ by rule
$\Delta_2; \Sigma'; \Gamma, x{:}A_1 \vdash M \doteq M_2 : A_2/(\cdot, \rho_2)$ by i.h.
$\Delta_2; \Sigma'; \Gamma \vdash \lambda x.L \doteq \lambda x.M_2 : A_1 \to A_2/(\cdot, \rho_2)$ by rule

$$\textit{Case. } \mathcal{D} = \frac{u{::}(\Phi \vdash P) \in \Delta}{(\Delta, \Sigma); \Gamma \vdash u[\pi] \sqcup R : P \Longrightarrow i[\mathsf{id}_\gamma]/(i{::}P[\Gamma], \hat{\Gamma}.u[\pi]/i, \hat{\Gamma}.R/i)}$$

$\Delta_1; i{::}P[\Gamma]; \Gamma \vdash i[\mathsf{id}_\gamma] \doteq R : P/(\cdot, \hat{\Gamma}.R/i)$ by rule meta-1
$\Delta_1; i{::}P[\Gamma]; \Gamma \vdash i[\mathsf{id}_\gamma] \doteq u[\pi] : P/(\cdot, \hat{\Gamma}.u[\pi]/i)$ by rule meta-1

$\textit{Case. } \mathcal{D} = (\Delta, \Sigma); \Gamma \vdash H_1 \cdot S_1 \sqcup H_2 \cdot S_2 : P \Longrightarrow i[\mathsf{id}_\Gamma]/((i{::}\Gamma \vdash P), (H_1 \cdot S_1/i), (H_2 \cdot S_2/i))$

$H_1 \neq H_2$ and i must be new by inversion
$\Delta_1; \Sigma; \Gamma \vdash i[\mathsf{id}_\Gamma] \doteq H_1 \cdot S_1 : P/(\cdot, \hat{\Gamma}.H_1 \cdot S_1/i)$ by meta-1
$\Delta_2; \Sigma; \Gamma \vdash i[\mathsf{id}_\Gamma] \doteq H_2 \cdot S_2 : P/(\cdot, \hat{\Gamma}.H_2 \cdot S_2/i)$ by meta-1

$\textit{Case. } \mathcal{D} = (\Delta, \Sigma); \Gamma \vdash (M_1; S_1) \mathbin{\underline{\sqcup}} (M_2; S_2) : A_1 \to A_2 > P \Longrightarrow (M; S)/((\Sigma_1, \Sigma_2), (\rho_1, \rho_2), (\rho_1', \rho_2'))$

$(\Delta, \Sigma); \Gamma \vdash M_1 \sqcup M_2 : A_1 \Longrightarrow M/(\Sigma_1,\ \rho_1, \rho_1')$ by inversion
$(\Delta, \Sigma); \Gamma \ \vdash\ S_1 \mathbin{\underline{\sqcup}} S_2 : A_2 > P \Longrightarrow S/(\Sigma_2,\ \rho_2, \rho_2')$
$\Delta_1; \Sigma_1; \Gamma \vdash M \doteq M_1 : A_1/(\cdot, \rho_1)$ by i.h.
$\Delta_2; \Sigma_1; \Gamma \vdash M \doteq M_2 : A_1/(\cdot, \rho_1')$ by i.h.
$\Delta_1; \Sigma_2; \Gamma \ \vdash\ S \doteq S_1 : A_2 > P/(\cdot, \rho_2)$ by i.h.
$\Delta_2, \Sigma_2; \Gamma \ \vdash\ S \mathbin{\underline{\sqcup}} S_2 : A_2 > P/(\cdot, \rho_2)$ by i.h.
$\Delta_1; \Sigma_1, \Sigma_2; \Gamma \vdash (M; S) \sqcup (M_1; S_1) : A_1/(\cdot, (\rho_1, \rho_1'))$ by rule
$\Delta_2, \Sigma_1, Sigma_2; \Gamma \vdash (M; S) \sqcup (M_2; S_2) : A_1/(\cdot, (\rho_2, \rho_2'))$ by rule □

Theorem 6.5 Insertion and retrieval for substitutions.
If $\Delta_2, \Delta_1, \Sigma \vdash \rho_1 \sqcup \rho_2 : \Sigma' \Longrightarrow \rho/(\Sigma'', \theta_1, \theta_2)$ *then* $\Delta_1; \Sigma'' \vdash \rho \ \doteq\ \rho_1 : \Sigma'/(\cdot, \theta_1)$ *and* $\Delta_2; \Sigma'' \vdash \rho \ \doteq\ \rho_2 : \Sigma'/(\cdot, \theta_2)$

Proof. Structural induction on the first derivation and use of lemma 6.4. □

Next, we show how to traverse the tree, to find a path $[\![\rho_n]\!][\![\rho_{n-1}]\!] \ldots \rho_1$ such that ρ_2 is an instance of it and return a contextual substitution θ such that $[\![\theta]\!][\![\rho_n]\!][\![\rho_{n-1}]\!] \ldots \rho_1 = \rho_2$. Traversal of the tree is straightforward.

$$\frac{\Delta, \Sigma \vdash \rho \doteq \rho_2 : \Sigma'/(\theta', \rho') \quad \Delta \vdash C \doteq \rho' : \Sigma/\theta}{\Delta \vdash [(\Sigma \vdash \rho \twoheadrightarrow C), C'] \doteq \rho_2 : \Sigma'/(\theta', \theta)}$$

$$\frac{\text{there is no derivation such that} \Delta, \Sigma \vdash \rho \doteq \rho_2 : \Sigma'/(\theta', \rho') \quad \Delta \vdash C' \doteq \rho : \Sigma/\theta}{\Delta \vdash [(\Sigma \vdash \rho \twoheadrightarrow C), C'] \doteq \rho_2 : \Sigma'/\theta}$$

THEOREM 6.6 SOUNDNESS OF RETRIEVAL.
If $\Delta \vdash C \doteq \rho' : \Sigma'/\theta$ *then there exists a child* C_i *with substitution* ρ_i *in* C *such that the path* $[\![\theta]\!][\![\rho_n]\!][\![\rho_{n-1}]\!] \ldots [\![\rho_i]\!] = \rho'$.

PROOF. By structural induction on the first derivation and use of lemma 6.3. □

Finally, we show that if we insert ρ into a substitution tree and obtain a new tree, then we are able to retrieve ρ from it.

THEOREM 6.7 INTERACTION BETWEEN INSERTION AND RETRIEVAL.
If $\Delta \vdash (\Sigma \vdash \rho \twoheadrightarrow C) \sqcup \rho_2 : \Sigma \Longrightarrow (\Sigma \vdash \rho \twoheadrightarrow C')$ *then* $\Delta \vdash C \doteq \rho_2/\mathsf{id}_\Delta$.

PROOF. Structural induction on the derivation using lemma 6.5. □

7. EXTENSION TO DEPENDENTLY TYPED TERMS

Substitution trees are especially suited for indexing dependently typed terms, since they provide more flexibility than indexing techniques such as discrimination tries which only allow us to share common prefixes. To illustrate this point, we define a data-structure for lists consisting of characters and we keep track of the size of the list by using dependent types.

$$\begin{array}{ll} \mathsf{char} : \mathsf{type}. & \mathsf{list} : \mathsf{char} \rightarrow \mathsf{type}. \\ \mathsf{a} : \mathsf{char}\ . & \mathsf{nil} : \mathsf{list}\ 0. \\ \mathsf{b} : \mathsf{char}\ . & \mathsf{cons} : \Pi n{:}\mathsf{int}\ .\mathsf{char} \rightarrow \mathsf{list}\ n \rightarrow \mathsf{list}\ (n+1). \\ \mathsf{test} : \Pi n{:}\mathsf{int}\ .\mathsf{list}\ n \rightarrow \mathsf{type}. & \end{array}$$

The size of lists is an explicit argument to the predicate test. Hence test takes in two arguments, the first one is the size of the list and the second one is the actual list. The list constructor cons takes in three arguments. The first one denotes the size of the list, the second argument denotes the head and the third one denotes the tail. To illustrate, we give a few examples. We use gray color for the explicit arguments.

$$\begin{array}{l} \mathsf{test}\ 4\ (\mathsf{cons}\ 4\ a\ (\mathsf{cons}\ 3\ a\ (\mathsf{cons}\ 2\ a\ (\mathsf{cons}\ 1\ b\ \mathsf{nil})))) \\ \mathsf{test}\ 5\ (\mathsf{cons}\ 5\ a\ (\mathsf{cons}\ 4\ a\ (\mathsf{cons}\ 3\ a\ (\mathsf{cons}\ 2\ a\ (\mathsf{cons}\ 1\ b\ \mathsf{nil}))))) \\ \mathsf{test}\ 6\ (\mathsf{cons}\ 6\ a\ (\mathsf{cons}\ 5\ a\ (\mathsf{cons}\ 4\ a\ (\mathsf{cons}\ 3\ b\ (\mathsf{cons}\ 2\ a\ (\mathsf{cons}\ 1\ b\ \mathsf{nil})))))) \end{array}$$

If we use non-adaptive indexing techniques such as discrimination tries, we process the term from left to right and we will be able to share common prefixes. In the given example, such a technique discriminates on the first argument, which denotes the size of the list and leads to no sharing between the second argument. The substitution tree on the other hand allows us to share the structure of the second argument. The most specific linear generalization in this example is

$$\mathsf{test}\ i_1[\mathsf{id}]\ (\mathsf{cons}\ i_2[\mathsf{id}]\ a\ (\mathsf{cons}\ i_3[\mathsf{id}]\ a\ (\mathsf{cons}\ i_4[\mathsf{id}]\ a\ (\mathsf{cons}\ i_5[\mathsf{id}]\ i_6[\mathsf{id}]\ \mathsf{nil})))).$$

This allows us to skip over the implicit first argument denoting the size and indexing on the second argument, the actual list. It has been sometimes argued that it is possible to retain the flexibility in non-adaptive indexing techniques by reordering the arguments to test. However, this only works easily in an untyped setting and it is not clear how to maintain typing invariants in a dependently typed setting if we allow arbitrary reordering of arguments. Hence higher-order substitution trees offer a adaptive compact indexing data-structure while maintaining typing invariants.

However, there are unique challenges of designing and implementing higher-order substitution trees for dependently typed terms. First, our substitution tree is designed for linear higher-order patterns. However, transforming dependently typed terms into linear higher-order patterns may result in ill-typed terms – or better linear higher-order patterns are only well-typed modulo variable definitions.

We may think of linear terms as a representation which is only used internally, and all linear terms are well-typed modulo variable definitions. Then we can show that approximate types (e.g. types where dependencies have been erased) are preserved in substitution trees, and all intermediate variables introduced are only used within this data-structure, but do not leak outside. As a consequence, we will always obtain a dependently typed term after composing the contextual substitutions in one branch of the substitution tree and solving the variable definitions.

8. RELATED WORK AND CONCLUSION

We have presented a higher-order term indexing technique, called higher-order substitution trees. We only know of two other attempts to design and implement a higher-order term indexing technique. L. Klein [Klein 1997] developed in his master's thesis a higher-order term indexing technique for simply typed terms where algorithms are based on a fragment of Huet's higher-order unification algorithm, the simplification rules. Since the most specific linear generalization of two higher-order terms does not exist in general, he suggests to maximally decompose a term into its atomic subterms. This approach results in larger substitution trees and stores redundant substitutions. In addition, he does not use explicit substitutions leading to further redundancy in the representation of terms. As no linearity criteria is exploited, the consistency checks need to be performed eagerly, potentially degrading the performance.

Necula and Rahul briefly discuss the use of automata driven indexing for higher-order terms in [Necula and Rahul 2001]. Their approach is to ignore all higher-order features when maintaining the index, and return an imperfect set of candidates. Then full higher-order unification on the original terms is used to filter out the ones which are in fact unifiable in a post-processing step. They also implemented Huet's unification algorithm, which is highly nondeterministic. Although they have achieved substantial speed-up for their application in proof-carrying code, it is not as general as the technique we have presented here. The presented indexing technique is designed as a perfect filter for linear higher-order patterns. For objects which are not linear higher-order patterns, we solve variable definitions via higher-order unification, but avoid calling higher-order unification on the original term.

Higher-order substitution trees provide a very flexible term indexing structure. However, in general there may be multiple ways to insert a term and no optimal substitution trees exist. For example, in the substitution tree given earlier we compare the substitution for the third argument before the substitutions for the first argument when looking up the term (3) and term (4). While we traverse the term (1) and (2) from left to right. This feature leads to very compact substitution trees and better memory usage and retrieval times.

We have implemented and successfully used higher-order substitution trees in the context of higher-order tabled logic programming. The table is a dynamically built index, i.e. when evaluating a query we store intermediate goals encountered during proof search. In our implementation for the tabled logic programming engine, we observed performance improvements up to a factor of 10 for some examples [Pientka 2003b; 2003a]. One interesting use of indexing is in indexing the actual higher-order logic program. For this we can build the index statically. Although the general idea of substitution trees is also applicable in this setting there are several important optimizations. For example, we can compute an optimal substitution tree via unification factoring [Dawson et al. 1995] for a static set of terms to get the best sharing among clause heads. In the future, we plan to adopt and optimize substitution tree indexing for indexing higher-order logic programming clauses.

REFERENCES

Abadi, M., Cardelli, L., Curien, P.-L., and Lèvy, J.-J. 1990. Explicit substitutions. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California*. ACM, 31–46.

Dawson, S., Ramakrishnan, C. R., Skiena, S., and Swift, T. 1995. Principles and practice of unification factoring. *ACM Transactions on Programming Languages and Systems 18,* 6, 528–563.

Dowek, G., Hardin, T., and Kirchner, C. 1995. Higher-order unification via explicit substitutions. In *Proceedings of the Tenth Annual Symposium on Logic in Computer Science*, D. Kozen, Ed. IEEE Computer Society Press, San Diego, California, 366–374.

Graf, P. 1995. Substitution tree indexing. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications, Kaiserslautern, Germany*. Lecture Notes in Computer Science (LNCS) 914. Springer-Verlag, 117–131.

Hanus, M. and Prehofer, C. 1999. Higher-order narrowing with definitional trees. *Journal of Functional Programming 9,* 1, 33–75.

Iliano Cervesato, F. P. 2003. A linear spine calculus. *Journal of Logic and Computation 13,* 5, 639–688.

Klein, L. 1997. Indexing für Terme höherer Stufe. Diplomarbeit, FB 14, Universität des Saarlandes, Saarbrücken, Germany.

Miller, D. 1991a. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation 1,* 4, 497–536.

Miller, D. 1991b. Unification of simply typed lambda-terms as logic programming. In *Eighth International Logic Programming Conference*. MIT Press, Paris, France, 255–269.

Nanevski, A., Pfenning, F., and Pientka, B. 2006. A contextual modal type theory. *ACM Transactions on Computational Logic (accepted, to appear in 2007)*, 56 pages.

Necula, G. and Rahul, S. 2001. Oracle-based checking of untrusted software. In *28th ACM Symposium on Principles of Programming Languages (POPL'01)*. 142–154.

Pfenning, F. 1991. Unification and anti-unification in the Calculus of Constructions. In *Sixth Annual IEEE Symposium on Logic in Computer Science*. Amsterdam, The Netherlands, 74–85.

Pfenning, F. and Schürmann, C. 1999. System description: Twelf — a meta-logical framework for deductive systems. In *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, H. Ganzinger, Ed. Springer-Verlag Lecture Notes in Artificial Intelligence (LNAI) 1632, Trento, Italy, 202–206.

Pientka, B. 2002. A proof-theoretic foundation for tabled higher-order logic programming. In *18th International Conference on Logic Programming, Copenhagen, Denmark*, P. Stuckey, Ed. Lecture Notes in Computer Science (LNCS), 2401. Springer-Verlag, 271 –286.

Pientka, B. 2003a. Higher-order substitution tree indexing. In *19th International Conference on Logic Programming, Mumbai, India*, C. Palamidessi, Ed. Lecture Notes in Computer Science (LNCS 2916). Springer-Verlag, 377–391.

Pientka, B. December 2003b. Tabled higher-order logic programming. Ph.D. thesis, Department of Computer Sciences, Carnegie Mellon University. CMU-CS-03-185.

Pientka, B. and Pfennning, F. July 2003. Optimizing higher-order pattern unification. In *19th International Conference on Automated Deduction, Miami, USA*, F. Baader, Ed. Lecture Notes in Artificial Intelligence (LNAI) 2741. Springer-Verlag, 473–487.

Ramakrishnan, I. V., Sekar, R., and Voronkov, A. 2001. Term indexing. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Vol. 2. Elsevier Science Publishers B.V., 1853–1962.

Sarkar, S., Pientka, B., and Crary, K. 2005. Small proof witnesses for lf. In *21st International Conference on Logic Programming, Sitges, Spain*, M. Gabbrielli and G. Gupta, Eds. Lecture Notes in Computer Science (LNCS), vol. 3668. Springer-Verlag, 387–401.

Watkins, K., Cervesato, I., Pfenning, F., and Walker, D. 2002. A concurrent logical framework I: Judgments and properties. Tech. Rep. CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University. Forthcoming.