Lincx: A Linear Logical Framework with First-class Context

Agata Murawska, Shawn Otis, and Brigitte Pientka

McGill University, Montreal, QC, Canada,
{bpientka}@cs.mcgill.ca

Abstract.

Keywords: Logical frameworks, Dependent Types, Proof Assistant

1 Introduction

Mechanizing formal systems together with proofs about them plays an important role in establishing trust in formal developments. While over the last two decades we have made substantial progress towards providing support to mechanize realistic systems (see for example mechanization of ML), specifying stateful systems and reasoning about systems with state remains very challenging.

Common proof and programming environments such as Coq or Agda support encoding of formal systems together with their meta-theory, but formal systems that have a notion of state can only be translated with a great deal of overhead making such encodings and the reasoning about them more complex than they in fact are.

Linear logic on the other hand provides a concise elegant framework for specifying stateful systems (see Cervesato and Pfenning [2002]). It has been successfully used to model blocks world, games, reasoning about references and file handlers [Wadler, 1990]. Recently, it has been used as a foundation for session types [Caires and Pfenning, 2010, Wadler, 2012]. While linear logical frameworks that support the specification of such systems exist (see for example CELF), how to encode proofs about such systems remains an open challenge. In this paper we take a first step towards building a foundation for mechanizing the meta-theory of stateful systems. Specifically, we introduce Lincx, a linear logical framework that supports first class context, contextual objects, and types. Lincx allows us represent linear specifications and describe abstractly derivation trees that depend on a context of linear and unrestricted assumptions.

With such a foundation in hand we can reason about it using for example the programming and proof environment Beluga [Cave and Pientka, 2012, Pientka and Cave, 2015]. A maybe surprising insight from this work is that we can simply use a first-order logic proof language with fix-points to reason about derivations that depend on a context of assumptions. In Beluga, we use contextual LF with first-class contexts to describe such derivations. However, Beluga's first-order

proof language is parametric in the domain and hence, in principle, we can replace contextual LF with first-class contexts with a linear version. Hence, it can be replaced by Lincx to support specifying and reasoning about stateful systems. Our technical contributions in defining Lincx are as follows:

- Building on [Cervesato and Pfenning, 1996] on the one hand and on Nanevski et al. [2008], Pientka [2008] on the other, we define contextual linear LF with fist-class context together with an equational theory for linear contexts; this equational theory for linear contexts allows us to reason about linear contexts and joining two contexts.
- We define a bi-directional type system modulo context constraints that characterize joins (or splits) of contexts. Following recent tradition, our type system only characterizes normal forms, as these are the only terms of interest in LF, and we define hereditary substitution to guarantee normal forms are preserved.
- Context constraints characterize abstractly the context joins that arise in a given valid typing derivation. They hence abstractly capture the tree-like structure of a given typing derivation reducing it to only the context joins (or splits). We show that our equational theory of context constraints admits reflexivity, symmetry, and transitivity. To establish these properties it is key to reason about equivalence classes of context constraints.
- Dual to context joins, we also define joins of simultaneous substitutions and prove simultaneous substitution principles.
- We define meta-substitution principles, in particular we define meta-substitution for context variables.

Can we define a single meta-substitution principle? -bp

- We prove that type checking in Lincx is decidable. As we only characterize canonical forms in Lincx (modulo equivalent set of context constraints), equality reduces to syntactic equality (modulo α-renaming).

2 Example

2.1 Counting Variables

In the first example, we encode a simple language of arithmetic expressions and let-expressions. Our goal is to enforce that all variables introduced in the let-expression can only be used once. This can be elegantly accomplished by encoding the language in the linear logical framework LF using `-o` to describe the linear function space. As all variables introduced by a let-expression can only be used once, we model the binding structure in the let-expression via the linear function space in LLF. We introduce a linear LF type together with its constructors using the keyword **Linear LF**.

```
Linear LF tm : type =
| zero: tm
| suc : tm -o tm
| add : tm -o tm -o tm
| letv: tm -o (tm -o tm) -o tm;
```

We now define a function `cntV` that counts the occurences of all variables in a given linear expression. As we traverse our arithmetic expression, we accumulate variables in a linear context. Following contextual LF, we pair the LF object of type `tm` together with a context γ that only keeps track of variables of type `tm` where each of these variables can only be used once. We specify the schema of the context γ as

```
schema ctx = (x:^tm);
```

where `:^` describes a linear assumption. Hence, a context of schema `ctx` has the form x_1`:^tm`, ..., x_n`:^tm`.

The function `cntV` takes in a context γ of linear term variables, a term `e` of type `[`γ `⊢ tm]`, i.e. a term which may refer to variables in γ, and returns as a result a natural number, defined via constructors `z:nat` and `s:nat → nat`. By writing the abstraction ``$\gamma$`:ctx`` in round parenthesis, we indicate that γ remains implicit in defining the function and reconstruction will need to infer it.

```
rec cntV : (γ:ctx) [γ ⊢  tm] → [ ⊢ nat] =
fn e ⇒ case e of
| [x:^tm ⊢ x]     ⇒ [ ⊢ s z]
| [      ⊢ zero]  ⇒ [ ⊢ z]
| [γ     ⊢ suc M] ⇒ cntV [γ ⊢ M]
| [γ     ⊢ letv ^ M ^ (λ̂y.N)] where M:[γ1 ⊢ tm], N:[γ2, y:^tm ⊢ tm], γ = γ1 & γ2
  ⇒ plus  (cntV [γ1 ⊢ M])  (cntV [γ2,y:^tm ⊢ N])
| [γ     ⊢ add ^ M ^ N] where M:[γ1 ⊢ tm], N:[γ2 ⊢ tm], γ = γ1 & γ2
  ⇒  plus (cntV [γ1 ⊢ M]) (cntV [γ2 ⊢ N])
;
```

The function `cntV` is defined by case-analysis of the expression in the context γ. There are five possible cases: four cases arise from using one of the constructors, `z`, `suc`, `letv`, and `app`; one additional case arises as we could also encounter a variable from the context. As our context is linear, the only possible scenario is that we encounter a variable `x` in the context only containing `x:^tm` as the only assumption. If we encounter `z`, then the context must be empty. If we encounter `suc M` in the context γ, then we simply count the variables in `M` in the context γ. By default all schematic meta-variables written using capital letters may depend on all the assumptions in the context they appear in. Hence `M` may depend on the context γ. More precisely, all meta-variables are associated with a postponed substitution which we omitted, if it is a trivial weakening or identity substitution. In `[`γ `⊢ suc M]` the meta-variable `M` has type `tm` in the context γ and hence is associated with the identity substitution.

Let us consider the case for `[`γ `⊢ add ^ M ^ N]` where we explicitly specify the type of `M` as `[`γ_1`⊢tm]` and the type of N as `[`γ_2`⊢tm]` where γ `=` γ_1 `&` γ_2, i.e. γ is the join of the contexts γ_1 and γ_2. Again we omit the identity substitution which is implicitly associated with `M` and `N`. We then simply recursively count the variables in `[`γ_1`⊢M]` and `[`γ_2`⊢N]` respectively and add them up using the auxiliary function `plus`.

Finally, let us consider the case for `[`γ `⊢ letv ^ M ^ (`$\widehat{\lambda}$`y.N)]`. We again specify the type of `M` and `N` as well as the context join γ `=` γ_1 `&` γ_2. Note that `N` is meaningful in the context γ_2 extended with the declaration `y:^tm`. We then recursively count variables in `[`γ_1 `⊢ M]` and `[`γ_2`,y:^tm ⊢ N]`.

2.2 Copy Example

We consider next the common copy example, where we want to copy a linear term. How do we proceed? - Let us first define linear lambda-terms using the type family `tm` and the type family `tm'`.

```
Linear LF tm:type =
| lam: (tm -o tm) -o tm
| app: tm -o tm -o tm;
```

```
Linear LF tm':type =
| lam': (tm' -o tm') -o tm'
| app': tm' -o tm' -o tm';
```

Then we define the schema of a linear context that keeps track of both `tm` and `tm'` assumptions as follows:

```
schema jctx = b:^block (x:^tm, y:^tm')
```

Note that a context of schema `jctx` consists of blocks of assumptions, where each block can only be used exactly once. Finally, we write a recursive program that copies `tm` objects to `tm'` objects.

```
rec copy: (γ:jctx) [γ ⊢ tm] → [γ ⊢ tm'] =
fn t ⇒ case t of
| [b:^ block (x:^tm, y:^tm') ⊢ b.1] ⇒ [b:^block (x:^tm, y:^tm') ⊢ b.2]
| [γ ⊢ lam λ^x.M] ⇒
  let [γ, b:^block (x:^tm, x:^tm') ⊢ N[…,b.2] ] = copy [γ, b:^block (x:^tm, y:^tm') ⊢ M[…
     ,b.1]] in
    [γ ⊢ lam' λ^y.N]

| [γ ⊢ app ^ M ^ N] where M:[γ1 ⊢ tm], N:[γ2 ⊢ tm] and γ = γ1 & γ2 ⇒
  let [γ1 ⊢ M'] = copy [γ1 ⊢ M] in
  let [γ2 ⊢ N'] = copy [γ2 ⊢ N] in
    [γ ⊢ app' ^ M' ^ N'];
```

Next, we consider the problem of showing that for each `tm` object there exists a `tm'` object together with a proof that they are equal. We first define a predicate `cp` that relates `tm` and `tm'` objects.

```
Linear LF cp: tm → tm' → type =
| cp_lam: ({x:tm}{y:tm'} cp x y -o cp (M x) (M' y))
       -o cp (lam \\x.M) (lam' \\y.M')
| cp_app: cp M M' -o cp N N'
      -o cp (app ^ M ^ N) (app' ^ M' ^ N').


schema cctx = b:^block (x:tm, y:tm',u:^cp x y)
```

The schema `cctx` captures contexts that consist of block describing the relationship of a given `tm` variable to its corresponding `tm'` variable together with a proof that they are related. Finally, we define an inductive type `Copy` that pairs a `tm'` object together with a proof that is is equivalent to the input `tm` object. The certified copy program is then taking as input a term `M` and returns an object `Copy [γ ⊢ M]`.

```
type Copy: (γ:cctx)[γ ⊢ tm] → ctype =
| Prf: {M':[γ ⊢ tm']} [γ ⊢ cp M M'] → Copy [γ ⊢ M]

rec ccopy: (γ:cctx) {M:[γ ⊢ tm]} Copy [γ ⊢ tm] =
fn t ⇒ case t of
| [b:^ block (x:tm, y:tm',u:^cp x y) ⊢ b.1] ⇒
  Prf [b:^block (x:tm, y:tm',u:^cp x y) ⊢ b.2] [b:^block (x:tm, y:tm'm u:^cp x y) ⊢ b.3]
| [γ ⊢ lam λ^x.M] ⇒
  let Prf [γ, b:^block (x:tm, x:tm',u:^cp x y) ⊢ N[…,b.2] ]
          [γ, b:^block (x:tm, x:tm',u:^cp x y) ⊢ D]
```

```
      = ccopy [γ, b:^block (x:tm, y:tm',u:^cp x y) ⊢ M[…,b.1]] in
    Prf [γ ⊢ lam' λ^y.N]
        [γ ⊢ cp_lam λx.λy.λ^u.D]

| [γ ⊢ app ^ M ^ N] where M:[γ1 ⊢ tm], N:[γ2 ⊢ tm] and γ = γ1 & γ2 ⇒
  let Prf [γ1 ⊢ M'] [γ1 ⊢ D1] = ccopy [γ1 ⊢ M] in
  let Prf [γ2 ⊢ N'] [γ2 ⊢ D2] = ccopy [γ2 ⊢ N] in
    Prf [γ ⊢ app' ^ M' ^ N'] [γ ⊢ cp_app ^ D1 ^ D2];
```

2.3 Type Uniqueness

As our third example, we consider the implementation of type uniqueness for linear lambda-terms. We encode terms using the intuitionistic fragment of our logical framework and only the typing rules guarantee that well-typed terms are in fact linear, i.e. use their variables only once. The encoding is otherwise standard following higher-order abstract syntax representations.

```
LF tp: type =
| i  : tp
| arr: tp → tp → tp;

LF term: type =
| lam: tp → (term → term) → term
| app: term → term → term

Linear LF oft: term → tp → type =
t_lam: ({x:term}oft x T -o oft (M x) S)
     -o oft (lam T M) (arr T S)
t_app: oft M (arr T S) -o oft N T
     -o oft (app M N) S
```

To prove that every term has a unique type, i.e. we must show that given $\gamma \vdash$ `oft M T` and $\gamma \vdash$ `oft M S`, the type `T` and `S` are equal. We should however exercise some care as to how the context γ can be defined and what the term `M` and the types `S` and `T` depend on. The typing context γ contains the intuitionistic unrestricted assumptions `x:term` and the linear assumption `oft x T`. Both assumptions are introduced at the same time. This is captured in the schema definition `tctx` by pairing them using the **block**-keyword – more concretly, it defines a Σ-type. But in what context does the term `M` make sense? - Terms can only refer to term assumptions, but not typing assumptions. The schema `ctx` defines the schema of such contexts. As we only consider simple types, they are always closed.

```
schema ctx  = term ;
schema tctx = some [t:tp] b:^block x:term ; u:^oft x t;
```

Revisiting our previous informal statement of type uniqueness, we therefore state more clearly that the types `T` and `S` are closed by associating them with the weakening substitution written `[]` whenever they occur within the context γ. The term `M` is only allowed to depend on the unrestricted parts of γ. Intuitively, when checking that a given contextual type such as `[`$\gamma \vdash$ `oft M T[]]` is well-kinded, we check that `oft M T[]` is well-kinded in the unrestricted part of γ which is denoted by $\overline{\gamma}$, i.e. it only depends on term assumptions, but not typing assumptions. Hence, `M` has type `[`$\overline{\gamma} \vdash$`term]` while `T` and `S` have type `[ ⊢tp]`.

```
rec unique: (γ:tctx)[γ ⊢oft M T[]] → [γ ⊢oft M S[]] → [ ⊢eq T S] =
fn d ⇒ fn e ⇒ case d, e of
```

```
| [b:^block (x:term, u:^oft x T) ⊢ b.u ] , [b:^block (x:term, u:^oft x T) ⊢ b.u ]
  ⇒ [ ⊢ ref]

| [γ ⊢ t_app ^ D1 ^ D2] , [γ ⊢ t_app ^ E1 ^ E2]
  where M:[γ̄ ⊢term], N:[γ̄ ⊢term], T:[ ⊢tp], R:[ ⊢tp], S:[ ⊢tp], P:[ ⊢tp]
        D1:[γ1 ⊢oft M (arr R[] T[])], D2:[γ2 ⊢oft N R[]],
        E1:[γ1 ⊢oft M (arr P[] S[])], E2:[γ2 ⊢oft N P[]], γ = γ1 & γ2
  ⇒ let [ ⊢ref] = unique [γ1 ⊢ D1] [γ1 ⊢ E1] in
      [ ⊢ref]

| [γ ⊢ t_lam ^ (λx.λ̂u.D)] ,  [γ ⊢ t_lam ^ (λx.λ̂u.E)]
  where M:[γ̄, x:term ⊢ term], R:[ ⊢tp], T:[ ⊢tp],
         D:[γ,x:term, u:^oft x R[] ⊢ oft M[…,x] T[]]
         E:[γ,x:term, u:^oft x R[] ⊢ oft M[…,x] S[]]
  ⇒ let [⊢ref] = unique [γ,b:^block (x:term, u:^oft x R[]) ⊢ D[…,b.x,b.u]]
                        [γ,b:^block (x:term, u:^oft x R[]) ⊢ E[…,b.x,b.u]] in
      [ ⊢ref]
;
```

The proof proceeds by simultaneously pattern matching on the derivations `d` and `e`. There are three cases.

Variable Case As our context γ is linear, this case can only arise when we have exactly one assumption, i.e. `b^:block (x:term, u:^oft x T)`. As both derivations `d` and `e` refer to the same context, we are guaranteed that they refer to the same typing assumption `u`. We access parts of a block by using projections writing `b.u` to extract the second field of the block. Therefore, we can simply provide a witness for `[ ⊢eq T T]` by referring to reflexivity.

Application Case We consider here the case where `d` stands for a derivation `[γ ⊢ oft (app M N) T[]]` and `e` stands for a derivation `[γ ⊢ oft (app M N) S[]]`. By inversion on both derivations using `t_app`, we obtain sub-derivations `D1` and `E1` which are meaningful in the context γ_1, and sub-derivations `D2` and `E2` which are meaningful in the context γ_2 where γ is the join of the contexts γ_1 and γ_2. We explicitly define the types of all the schematic variables and context joins in the program using a **where** clause. We note that the terms `M` and `N` are meaningful terms in the unrestricted context $\overline{\gamma}$. Further, as context joins only pertains linear assumptions we will have that $\overline{\gamma} = \overline{\gamma_1} = \overline{\gamma_2}$. This is essential to successful kind checking of the types we are working with. Apart from the reasoning about contexts, the case is straightforward. We appeal to the induction hypothesis by making a recursive call on `[γ1 ⊢ D1]` and `[γ1 ⊢ E1]` obtaining a proof for `[ ⊢ eq (arr R T) (arr P S)]`. By inversion, we know we could only have proven it by reflexivity. As a consequence we learn that `T` and `S` are not only provably equal but they are the same object. We hence return as a witness the reflexivity proof of `[ ⊢ eq T S]`.

Lambda Case Finally, we consider the case where `d` stands for a derivation `[γ ⊢ oft (lam R[] λx.M) (arr R[] T[])]` and `e` stands for a derivation `[γ ⊢ oft (lam R[] λx.M) (arr R[] S[])]`. By inversion using `t_lam`, we obtain sub-derivation `D` standing for `[γ, x:term, u:^oft x R[] ⊢ oft M S[]]` and sub-derivation `E` standing for `[γ, x:term, u:^oft x R[] ⊢ oft M S[]]`. We now appeal to the induction hypothesis on `D` and `E` by making a recursive call in the context `γ, b:^block x:term, u:^oft x R[]`. To use both `D` and `E` in this extended context, we associate them with the substitution `…, b.x, b.u`. As a

result we obtain a proof that `[ ⊢ eq T S]`. By inversion we could only have derived such a fact by reflexivity, hence `T` and `S` are the same object and we therefore have a proof for `[ ⊢ eq T S]`. While the main argument is straightforward, the delicate and intricate aspect in the proof again pertains the reasoning with contexts. In particular, we need that $\overline{\gamma}$,`x:term`

In the remaining part of the paper, we will describe contextual linear logical framework with first-class contexts and equational theory on contexts which supports dependent types. This will be sufficient to guarantee that the given program `unique` is well-typed, as type checking of recursive programs in BELUGA is paramerized by an index domain. We will leave to the future how to guarantee that the function `unique` is total and covers all cases.

3 Contextual Linear LF with First-Class Contexts

We present contextual linear logical framework with first-class contexts (i.e. context variables) which generalizes the work by Cervesato and Pfenning [1996]. We follow recent presentations where we only describe canonical LF objects using hereditary substitutions.

3.1 Core Language

Our language (see Fig. 1) supports linear types, written $A \multimap B$, and dependent types $\Pi x{:}A.B$ where x may be used unrestricted in B. While our previous encoding of type uniqueness relied on the ability to describe blocks of types (i.e. Σ-types), we concentrate for simplicity here on contexts that only contain assumptions of type A. This can be however generalized to support typing assumptions of the form $\Sigma x_1{:}A_1, \ldots, \Sigma x_n{:}A_n.A_{n+1}$ we believe.

As usual, we only consider objects in η-long β-normal form, since these are the only meaningful objects. While our grammar only characterizes objects in β-normal form, our bi-directional typing rules will also ensure that objects are η-long. Normal canonical terms are either intuitionistic lambda- abstractions, linear lambda-abstractions, or neutral atomic terms. We define neutral atomic terms using a spine representation [Cervesato and Pfenning, 2003], as it makes the termination of hereditary substitution easier to establish. For example, instead of $x\ M_1 \ldots M_n$, we write $x \cdot M_1;\ \ldots;\ M_n;\ \epsilon$.

Our framework contains *ordinary bound variables* x which may refer to a variable declaration in the context or may be bound by either the unrestricted or linear lambda-abstraction. Contextual LF also allows two kinds of *contextual variables*: the meta-variable u has type $(\Psi \vdash P)$ and stands for a general LF object that has atomic type P and may use the variables declared in Ψ; the parameter variable p has type $(\Psi \vdash \#A)$ and stands for a variable object of type A from the context Ψ. The tag # on the type of parameter variables is a simple syntactic device to distinguish between the type of meta-variables and parameter variables. It does not introduce a subtyping relationship between the type $(\Psi \vdash \#A)$ and the type $(\Psi \vdash A)$.

Contextual Linear LF

Kinds	K	$::=$	$\mathsf{type} \mid \Pi x{:}A.K$
Types	A, B	$::=$	$P \mid \Pi x{:}A.B \mid A \multimap B$
Atomic Types	P, Q	$::=$	$a \cdot S$
Heads	H	$::=$	$x \mid c \mid p[\sigma]$
Spines	S	$::=$	$\epsilon \mid M\,;S \mid M\,\hat{;}\,S$
Atomic Terms	R	$::=$	$H \cdot S \mid u[\sigma]$
Canonical Terms	M, N	$::=$	$R \mid \lambda x.M \mid \widehat{\lambda} x.M$
Variable Declaration	D	$::=$	$x{:}A \mid x\hat{:}A$
Context	Ψ, Φ	$::=$	$\cdot \mid \psi \mid \overline{\psi} \mid \psi^{\uparrow} \mid \Psi,\ D$
Substitution	σ, τ	$::=$	$\cdot \mid \mathsf{id}_{\psi} \mid \mathsf{id}_{\overline{\psi}} \mid \mathsf{il}_{\psi} \mid \sigma, M$

Meta Language

Meta-Variables	X	$::=$	$u \mid p \mid \psi$
Meta-Objects	C	$::=$	$\widehat{\Psi}.R \mid \Psi$
Context Schema Elem.	E	$::=$	$\lambda(\overrightarrow{x_i{:}A_i}).D$
Context Schemata	G	$::=$	$E \mid G + E$
Meta Types	U	$::=$	$\Psi \vdash P \mid \Psi \vdash \#A \mid G$
Meta-Context	Δ	$::=$	$\cdot \mid \Delta, X : U$
Meta-Substitution	Θ	$::=$	$\cdot \mid \Theta, C/X$
Constraints	$\mathcal{C}$	$::=$	$\cdot \mid \mathcal{C}, \psi := \psi_1 \bowtie \psi_2$

Fig. 1. Contextual Linear LF with first-class contexts

Contextual variables are associated with a postponed substitution σ representing a closure. The intention is to apply σ as soon as we know what u (or p resp.) stands for. Intuitively, we can justify the closure as follows: let u describe a hole in a term M; if we were to apply a substitution σ to M, then σ gets stuck when it encounters the hole u. However, as soon as we know what u stands for, we can further apply the substitution σ to it.

Meta-variables are considered neutral terms and our typing rules will enforce that meta-variables are of atomic type. This is not a restriction, since we can always lower a meta-variable; intuitively, we transform $u[\sigma]\,M$ (or in our syntax $u[\sigma] \cdot M; \epsilon$) where u has type $(\Psi \vdash \Pi x{:}A.B)$ into $u'[\sigma, M]$ where u' has type $(\Psi, x{:}A \vdash B)$ by replacing u with $\widehat{\Psi}.\lambda x.u'[\sigma, x]$, possibly η-expanding x if necessary. Here $\widehat{\Psi}$ denotes simply a list of variables which is obtained by dropping all the type information from the declarations, but retaining the information whether a variable is linear or not. This information is irrelevant for typing, but becomes important when we define unification on linear types and terms (see Schack-Nielson and Schürmann [2010]).

$$
\begin{aligned}
\widehat{\cdot} &= \cdot \\
\widehat{\psi} &= \psi \\
\widehat{\overline{\psi}} &= \overline{\psi} \\
\widehat{\psi^{\uparrow}} &= \psi^{\uparrow} \\
\widehat{\Psi, x{:}A} &= \widehat{\Psi}, x \\
\widehat{\Psi, x\hat{:}A} &= \widehat{\Psi}, \widehat{x}
\end{aligned}
$$

We use one mixed context Ψ to contain both intuitionistic and linear assumptions: $x\hat{:}A$ is meant to represent a linear assumption in the context, $x : A$ is an intuitionistic assumption (also called *unrestricted assumption*). As is the case in contextual LF, we allow for a single context variable ψ to be present in the context. We write Ψ^0 for contexts that do not start with a context variable. We write Ψ, Φ^0 or sometimes Ψ, Φ for the extension of context Ψ by the variable declarations of Φ^0 or Φ, resp.

When defining our inference rules we will often need to access the *intuitionistic part* of a context. Following Cervesato and Pfenning [1996] we introduce the function $\overline{\Psi}$ which is defined as follows:

$$
\begin{aligned}
\overline{\cdot} &= \cdot \\
\overline{\psi} &= \overline{\psi} \\
\overline{\overline{\psi}} &= \overline{\psi} \\
\overline{\psi^{\uparrow}} &= \psi^{\uparrow} \\
\overline{\Psi, x{:}A} &= \overline{\Psi}, x{:}A \\
\overline{\Psi, x\hat{:}A} &= \overline{\Psi}
\end{aligned}
$$

Note that when applying this function to a context variable, it gets stuck. After we instantiate ψ with a concrete context, we will apply the operation. Extracting the intuitionistic part of a context is hence simply postponed.

Further, we define notation $\mathsf{unr}(\Psi)$ to denote an unrestricted context, i.e. a context that only contains unrestricted assumptions; while $\overline{\Psi}$ drops all linear assumptions, $\mathsf{unr}(\Psi)$ simply verifies that Ψ is a purely intuitionistic context. In other words, $\mathsf{unr}(\overline{\Psi})$ is always true.

In the simultaneous substitutions σ, we do not make its domain explicit. Rather, we think of a substitution together with its domain Ψ and the i-th element in σ corresponds to the i-th declaration in Ψ.

We build substitutions using a normal term M. We must however be careful. Note that a variable x is only a normal term M if it is of base type. However, as we push a substitution σ through a λ-abstraction $\lambda x.M$, we need to extend σ with x. The resulting substitution σ, x may not be well-formed, since x may not be of base type and in fact we do not know its type. This is taken care of in our definition of substitution following Cave and Pientka [2013].

Expression id_{ψ} denotes the identity substitution with domain ψ; expression $\mathsf{id}_{\overline{\psi}}$ denotes the identity substitution where the domain is restricted to the unrestricted part of the context ψ; $\cdot$ describes the empty substitution. As we sub-

stitute and replace a context variable with a concrete context, we unfold and generate an identity substitution for a given context Ψ.

We also include il_ψ which denotes a variable substitution with a linear context ψ as a domain and an unrestricted context $\psi^\uparrow, \Psi_0$ as a range. This is inspired by [Schack-Nielson and Schürmann, 2010, Schack-Nielson, 2011] who observed that we can replace any linear variable by an unrestricted variable.

add the description of $\psi^\uparrow$ and of the $\Psi^\uparrow$ operation [am]

As is common, we rely on hereditary substitutions [Watkins et al., 2002], written as $[N/x]B$ (or $[\sigma]B$ resp.) to guarantee that when we substitute a term N for the variable x in the type B, we obtain a type B' which is in normal form. Hereditary substitutions continue to substitute, if a redex is created; for example, when replacing naively x by $\lambda y.c\,y$ in the object $x\,z$, we would obtain $(\lambda y.\,c\,y)\,z$ which is not in normal form and hence not a valid term in our grammar. Hereditary substitutions continue to substitute z for y in $c\,y$ to obtain $c\,z$ as a final result. We postpone the discussion of hereditary substitutions to Section 3.6.

The meta-language for contextual linear LF describes meta-objects and meta-types where we package the object (i.e. type) together with the context in which it is meaningful. There are three different kinds of meta-types: 1) the meta-type $\Psi \vdash P$ describes a neutral term of type P in the context Ψ. We write u for *meta-variables* of such type and describe their instantiations as $\widehat{\Psi}.R$ where R is a neutral term of type P in the context Ψ. The operation $\widehat{\Psi}$ drops all the type information of a context Ψ only recording the names of the variables. This is necessary to support α-renaming. 2) the parameter type $\Psi \vdash \#A$ describes a variable head of type A in the context Ψ. We typically write p for a *parameter variable* of such type and describe their instantiations as $\widehat{\Psi}.R$ where R must stand for a variable object of type A in the context Ψ. Note that for $\Phi \vdash \#A$ to be a valid parameter type, either Φ must contain at most one linear variable declaration. 3) the schema G describes a context. We write ψ or ϕ for *context variables*. Following context schemas in Beluga, we can describe them by a simple language of regular expressions where + describes a choice. Intuitively, a context satisfies a schema G if every declaration D in the context is an instance of one of the schema elements. For convenience, we overload the terminology and often use meta-variables X to stand for context variable ψ, parameter variable p, in addition to the meta-variable u. Our previous examples allowed for schemas to describe structured sequences that consist of blocks of assumptions. We consider here only contexts that consist of LF types, not of blocks of LF types.

Finally we support constraint on context variables to define the join between two context variables.

3.2 Well-formed Meta-Context, Meta-Objects, and Context Constraints

As all judgments we subsequently define depend on a well-formed the meta-context and well-formed set of constraints, an important question is how to define these properties. We base our presentation of the meta-level (see Fig. 2)

on the presentation by Cave and Pientka [2012], however as we are considering context constraints there are new considerations.

In essence, a set of constraints describes abstractly a valid derivation tree concentrating only on what happened to the context Ψ. As we build a given derivation tree, we may need to split Ψ into sub-context, as we are working in a linear setting, where linear variables can only be used exactly once. The set of context constraints $\mathcal{C}$ abstracts over the concrete derivation tree by only concentrating on the context joins (or splits).

A set of context constraints describes a tree where we have a context variable that sits at the top which is defined by the multi-join of the context variables at the leafs. Let us look at an example: $\mathcal{C} = \psi := \psi_1 \bowtie \psi_2,\ \psi_1 := \psi_{11} \bowtie \psi_{12},\ \psi_{11} := \psi_{111} \bowtie \psi_{112}$. This set of constraints defines ψ to be $((\psi_{111} \bowtie \psi_{112}) \bowtie \psi_{12}) \bowtie \psi_2$. Hence we say the free context variables in $\mathcal{C}$ are the variables ψ_{111}, ψ_{112}, ψ_{12}, and ψ_2. The context variables labelling sub-joins such as ψ_1 and ψ_{11} are not considered as part of the free variables – they are merely labels for sub-joins. We call the context variable that is defined by a set of context joins the root. In the example above, this is ψ.

Definition 1 (Free Context Variables in $\mathcal{C}$). *Given a set $\mathcal{C}$ of context constraints, the set $\mathsf{FV}(\mathcal{C})$ of free context variables of that set is defined as follows: For each $\psi := \psi_1 \bowtie \psi_2 \in \mathcal{C}$ if $\psi_i := \phi_1 \bowtie \phi_2 \notin \mathcal{C}$, then $\psi_i \in \mathsf{FV}(\mathcal{C})$ for all $i \in \{1, 2\}$.*

It is also often useful to define a dependency path in the graph described by the set of context constraints using the reflexive, transitive, and symmetric closure. In the previous example, there is a dependency path from ψ_1 to the leaf ψ_{111}; there is also a dependency path between ψ_1 to ψ_2, as there is a join $\psi := \psi_1 \bowtie \psi_2$.

$\boxed{\mathcal{C} \vdash \psi \rightsquigarrow^* \phi}$ There is a chain between ψ to ϕ

$$\frac{\psi \in \mathsf{FV}(\mathcal{C})}{\mathcal{C} \vdash \psi \rightsquigarrow^* \psi} \quad \frac{\psi := \psi_1 \bowtie \psi_2 \in \mathcal{C}}{\mathcal{C} \vdash \psi \rightsquigarrow^* \psi_i} \text{ for } i \in \{1, 2\} \quad \frac{\mathcal{C} \vdash \psi \rightsquigarrow^* \gamma \quad \mathcal{C} \vdash \gamma \rightsquigarrow^* \phi}{\mathcal{C} \vdash \psi \rightsquigarrow^* \phi}$$

$\boxed{\mathcal{C} \vdash \psi \leftrightsquigarrow \phi}$ There is dependency path between ψ to ϕ

$$\frac{\mathcal{C} \vdash \psi \leftrightsquigarrow \gamma \quad \mathcal{C} \vdash \gamma \leftrightsquigarrow \phi}{\mathcal{C} \vdash \psi \leftrightsquigarrow \phi} \quad \frac{\mathcal{C} \vdash \psi \rightsquigarrow^* \phi}{\mathcal{C} \vdash \psi \leftrightsquigarrow \phi} \quad \frac{\mathcal{C} \vdash \phi \leftrightsquigarrow \psi}{\mathcal{C} \vdash \psi \leftrightsquigarrow \phi}$$

We now define the well-formedness of context constraints by ensuring that the set of constraints is non-circular and there exists at most one constraint for a given context variable.

Definition 2 (Well-formed Context Constraints). *A set $\mathcal{C}$ is well-formed, iff for all context joins $\psi = \psi_1 \bowtie \psi_2 \in \mathcal{C}$,*

- *there exists no context join $\psi = \phi_1 \bowtie \phi_2 \in \mathcal{C}$ s.t. $\psi_1 \neq \phi_1$ and $\psi_2 \neq \phi_2$.*
- *for $i \in \{1, 2\}$, there exists no path $\psi_i \rightsquigarrow^* \ldots \rightsquigarrow^* \psi$, i.e. $\mathcal{C} \vdash \psi_i \not\rightsquigarrow^* \psi$.*

Definition 3 (Well-formedness of $\mathcal{C}$ with respect to a meta-context Δ). *A set $\mathcal{C}$ of context constraints is well-formed with respect to Δ, iff $\mathcal{C}$ is well-formed and for all $\psi \in \mathsf{FV}(\mathcal{C})$ we have $\psi : G \in \Delta$.*

We now define the well-formedness of Δ modulo context constraints $\mathcal{C}$ (see Fig. 2). This is defined on the structure of Δ and is mostly straightforward. The interesting case arises, when we define that $\Delta, \psi : G$ is well-formed meta-context modulo $\mathcal{C}$. Intuitively, we want to recursively check that Δ is well-formed with respect to $\mathcal{C}$, but $\mathcal{C}$ might refer to ψ. It hence might violates our invariant that all the free context variables in $\mathcal{C}$ are also declared in Δ. We can however observe that a context variable ψ that is declared in Δ is in the set of free variables $\mathsf{FV}(\mathcal{C})$ and hence there exist no other context join in $\mathcal{C}$ s.t. $\psi := \psi_{11} \bowtie \psi_{12}$. Moreover, every context variable in $\mathsf{FV}(C)$ occurs exactly once in $\mathcal{C}$. We hence remove the context join $\phi := \psi \bowtie \gamma$ (or $\phi := \gamma \bowtie \psi$). We use as notation $\mathcal{C}/\{\psi\}$ to describe the removal of all the constraints involving ψ.

$\boxed{\vdash_{\mathcal{C}} \Delta \ \mathsf{mctx}}$ Meta-context Δ is well-formed modulo $\mathcal{C}$

$$\frac{}{\vdash_{\mathcal{C}} \cdot \ \mathsf{mctx}} \qquad \frac{\vdash_{\mathcal{C}} \Delta \ \mathsf{mctx} \quad \Delta; \mathcal{C} \vdash \Psi \ \mathsf{ctx} \quad \Delta; \mathcal{C}; \overline{\Psi} \vdash P \ \mathsf{type}}{\vdash_{\mathcal{C}} \Delta, u : [\Psi \vdash P] \ \mathsf{mctx}}$$

$$\frac{\vdash_{\mathcal{C}/\{\psi\}} \Delta \ \mathsf{mctx} \quad \Delta \vdash G \ \mathsf{schm}}{\vdash_{\mathcal{C}} \Delta, \psi : G \ \mathsf{mctx}} \qquad \frac{\vdash_{\mathcal{C}} \Delta \ \mathsf{mctx} \quad \Delta; \mathcal{C} \vdash \Psi \ \mathsf{ctx} \quad \Delta; \mathcal{C}; \overline{\Psi} \vdash A \ \mathsf{type}}{\vdash_{\mathcal{C}} \Delta, p : [\Psi \vdash \#A] \ \mathsf{mctx}}$$

The definition of meta-contexts relies on defining typing for meta-objects. We adapt here previous presentation of contextual types (see for example Cave and Pientka [2012]) to our setting. First, we consider when a context Ψ is of schema G. There are two interesting cases: the first one arises when Ψ is a context variable ψ. Such a context is of schema G if either $\psi : G \in \Delta$ or there exists a path from ψ to a context variable ϕ s.t. $\phi : G \in \Delta$. The second one arises when Ψ is built out of new variable declarations. We say $\Psi, x{:}A$ (or $\Psi, x\hat{:}A$ checks against schema G, if Ψ checks against schema G, A is an instance of a schema element (written as $x{:}A \in G$ and $x\hat{:}A \in G$ resp.), and A is a well-formed type in the context $\overline{\Psi}$, i.e. A can only depend on unrestricted variable declarations in Ψ.

Typing of meta-objects is straightforward: a meta-object $\widehat{\Psi}.R$ has type $[\Psi \vdash P]$, if R has type P in the context Ψ. The typing of variable objects, i.e. an object of type $[\Psi \vdash \#A]$, must be reconsidered carefully. A parameter type is inhabited only by variable objects, i.e. either concrete variables from Ψ or parameter variable associated with a variable substitution. The typing for parameter variables follows the typing for meta-objects. There are two cases to consider when we have a concrete variable x from the context: either Ψ contains only unrestricted variable declarations and $x : A$ is one of them; or x is in fact a linear variable of type A which forces Ψ to be a context with only one linear declaration $x\hat{:}A$.

Finally, we consider meta-substitutions. They are inductively defined on the structure of the meta-substitution itself and does not change in the linear setting.

$\boxed{\Delta;\mathcal{C} \vdash C \Leftarrow U}$ Meta-level term C checks agains type U

$$\frac{}{\Delta;\mathcal{C} \vdash \cdot \Leftarrow G} \qquad \frac{\mathcal{C} \vdash \psi \rightsquigarrow^* \phi \quad \phi : G \in \Delta}{\Delta;\mathcal{C} \vdash \psi \Leftarrow G} \qquad \frac{\mathcal{C} \vdash \psi \rightsquigarrow^* \phi \quad \phi : G \in \Delta}{\Delta;\mathcal{C} \vdash \overline{\psi} \Leftarrow G}$$

$$\frac{\Delta;\mathcal{C} \vdash \Psi \Leftarrow G \;\exists\overrightarrow{(x_i{:}A_i)}.y{:}B \in G \;\; \Delta;\mathcal{C};\Psi \vdash \sigma \Leftarrow \overrightarrow{(x_i{:}A_i)} \;\; \Delta;\mathcal{C};\overline{\Psi} \vdash A = [\sigma]_{\overrightarrow{(x_i{:}A_i)}}B}{\Delta;\mathcal{C} \vdash \Psi, x{:}A \Leftarrow G}$$

$$\frac{\Delta;\mathcal{C} \vdash \Psi \Leftarrow G \;\exists\overrightarrow{(x_i{:}A_i)}.y{:}\hat{:}B \in G \;\; \Delta;\mathcal{C};\Psi \vdash \sigma \Leftarrow \overrightarrow{(x_i{:}A_i)} \;\; \Delta;\mathcal{C};\overline{\Psi} \vdash A = [\sigma]_{\overrightarrow{(x_i{:}A_i)}}B}{\Delta;\mathcal{C} \vdash \Psi, x{:}\hat{:}A \Leftarrow G}$$

$$\frac{\Delta;\mathcal{C};\Psi \vdash R \Leftarrow P}{\Delta;\mathcal{C} \vdash \widehat{\Psi}.R \Leftarrow [\Psi \vdash P]}$$

$$\frac{x{:}A \in \Psi \quad \mathsf{unr}(\Psi)}{\Delta;\mathcal{C} \vdash \widehat{\Psi}.x \Leftarrow [\Psi \vdash \#A]} \qquad \frac{\mathsf{unr}(\Psi_1) \quad \mathsf{unr}(\Psi_2)}{\Delta;\mathcal{C} \vdash (\widehat{\Psi_1}, x, \widehat{\Psi_2}).x \Leftarrow [\Psi_1, x{:}\hat{:}A, \Psi_2 \vdash \#A]}$$

$$\frac{p : [\Phi \vdash \#A] \in \Delta \quad \Delta;\mathcal{C};\Psi \vdash \pi \Leftarrow \Phi \quad \Delta;\mathcal{C};\overline{\Psi} \vdash B = [\pi]_\Phi(A)}{\Delta;\mathcal{C} \vdash \widehat{\Psi}.p[\pi] \Leftarrow [\Psi \vdash \#B]}$$

Fig. 2. Typing rules for meta-terms

$\boxed{\Delta;\mathcal{C} \vdash \Theta \Leftarrow \Delta'}$ Meta-substitution Θ maps variables in Δ' to variables in Δ

$$\frac{}{\Delta;\mathcal{C} \vdash \cdot \Leftarrow \cdot} \qquad \frac{\Delta;\mathcal{C} \vdash \Theta \Leftarrow \Delta' \quad \Delta;\mathcal{C} \vdash C \Leftarrow [\![\Theta]\!]_{\Delta'}(U)}{\Delta;\mathcal{C} \vdash \Theta, C/X \Leftarrow \Delta', X : U}$$

3.3 Equivalence of Context Constraints

Let us revisit the example: $\mathcal{C} = \psi = \psi_1 \bowtie \psi_2,\ \psi_1 = \psi_{11} \bowtie \psi_{12},\ \psi_{11} = \psi_{111} \bowtie \psi_{112}$. This set of constraints defines ψ to be $((\psi_{111} \bowtie \psi_{112}) \bowtie \psi_{12}) \bowtie \psi_2$. As context joins are associative and commutative, there are other equivalent representations: $(\psi_{111} \bowtie (\psi_{112} \bowtie \psi_{12})) \bowtie \psi_2$, $(\psi_{111} \bowtie \psi_{112}) \bowtie (\psi_{12} \bowtie \psi_2)$, etc. We hence define the equivalence between sets of constraints by defining equivalence of multi-joins over contexts. Overloading notation we write $\mathcal{C}$ for $((\psi_{111} \bowtie \psi_{112}) \bowtie \psi_{12}) \bowtie \psi_2$.

Associativity and commutativity of context joins

$$\mathcal{C}, \psi = \psi_1 \bowtie \psi_2, \psi_1 = \psi_{11} \bowtie \psi_{12} \implies \mathcal{C}, \psi = \psi_0 \bowtie \psi_{11}, \psi_0 = \psi_2 \bowtie \psi_{12}$$
$$\mathcal{C}, \psi = \psi_1 \bowtie \psi_2 \implies \mathcal{C}, \psi = \psi_2 \bowtie \psi_1$$

Equivalence of context joins

$$\frac{\mathcal{C}_0 \implies \mathcal{C}_1}{\mathcal{C}_0 \equiv \mathcal{C}_1} \qquad \frac{\mathcal{C}_0 \equiv \mathcal{C}_1 \quad \mathcal{C}_1 \equiv C_2}{\mathcal{C}_0 \equiv \mathcal{C}_2} \qquad \frac{}{\mathcal{C} \equiv \mathcal{C}}$$

We remark that transforming a set of context joins preserves the set of free context variables, i.e. if $\mathcal{C} \equiv \mathcal{C}'$ then $\mathsf{FV}(\mathcal{C}) = \mathsf{FV}(\mathcal{C}')$.

3.4 Context Joins

$\boxed{\mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2}$ Context Ψ is a join of Ψ_1 and Ψ_2 given the constraints $\mathcal{C}$

$$\frac{\mathcal{C} \equiv \mathcal{C}' \quad (\psi := \psi_1 \bowtie \psi_2) \in \mathcal{C}'}{\mathcal{C} \vdash \psi = \psi_1 \bowtie \psi_2} \qquad \frac{}{\mathcal{C} \vdash \cdot = \cdot \bowtie \cdot}$$

$$\frac{\mathcal{C} \vdash \psi \leftrightsquigarrow \phi}{\mathcal{C} \vdash \psi = \psi \bowtie \overline{\phi}} \quad \frac{\mathcal{C} \vdash \psi \leftrightsquigarrow \phi}{\mathcal{C} \vdash \psi = \overline{\phi} \bowtie \psi} \quad \frac{\mathcal{C} \vdash \psi \leftrightsquigarrow \psi_1 \quad \mathcal{C} \vdash \psi \leftrightsquigarrow \psi_2}{\mathcal{C} \vdash \overline{\psi} = \overline{\psi}_1 \bowtie \overline{\psi}_2}$$

$$\frac{\mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\mathcal{C} \vdash \Psi, x{:}A = \Psi_1, x{:}A \bowtie \Psi_2, x{:}A}$$

$$\frac{\mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\mathcal{C} \vdash \Psi, x\hat{:}A = \Psi_1, x\hat{:}A \bowtie \Psi_2} \qquad \frac{\mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\mathcal{C} \vdash \Psi, x\hat{:}A = \Psi_1 \bowtie \Psi_2, x\hat{:}A}$$

Fig. 3. Rules for joining contexts

Rules for joining contexts (see Fig. 3) follow Schack-Nielson [2011], but are generalized to take into account context variables. They are not deterministic – however it is worth noting that if read as a procedure for splitting contexts, there would not be a unique split but a finite number of splits. For our purpose it suffices that in principle we can in principle generate all splits and try them one after the other. The rules for joining contexts preserve the following properties:

Lemma 1 (Reflexivity, Symmetry, Identity).

1. *If* $\Delta; \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2$ *then there exists* $\mathcal{C} \equiv \mathcal{C}'$ *s.t.* $\Delta; \mathcal{C}' \vdash \Psi = \Psi_2 \bowtie \Psi_1$.
2. *If* $\Delta; \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2$ *and* $\Delta; \mathcal{C} \vdash \Psi_1 = \Psi_{11} \bowtie \Psi_{12}$ *then there exists a context* Ψ_0 *and* $\mathcal{C} \equiv \mathcal{C}'$ *s.t.* $\Delta; \mathcal{C}' \vdash \Psi = \Psi_{11} \bowtie \Psi_0$ *and* $\Delta; \mathcal{C}' \vdash \Psi_0 = \Psi_{12} \bowtie \Psi_2$.
3. *If* $\Delta; \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2$ *and* $\Delta; \mathcal{C} \vdash \Psi_2 = \Psi_{21} \bowtie \Psi_{22}$ *then there exists a context* Ψ_0 *and* $\mathcal{C} \equiv \mathcal{C}'$ *s.t.* $\Delta; \mathcal{C}' \vdash \Psi_0 = \Psi_1 \bowtie \Psi_{21}$ *and* $\Delta; \mathcal{C}' \vdash \Psi = \Psi_0 \bowtie \Psi_{22}$.

4. *If* $\mathcal{C} \equiv \mathcal{C}'$ *and* $\Delta; \mathcal{C}' \vdash \Psi = \Psi_1 \bowtie \Psi_2$ *then* $\Delta; \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2$.
5. *If* $\mathsf{unr}(\Psi_0)$ *and* $\Delta; \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2$ *then* $\Delta; \mathcal{C} \vdash \Psi, \Psi_0 = \Psi_1, \Psi_0 \bowtie \Psi_2, \Psi_0$.

Proof. The statements are proven by induction on the first derivation.

We also state and prove a number of properties of context joins for unrestricted contexts. They are subsequently needed to prove meta-substitution lemmas about our type theory.

Lemma 2 (Properties of context joins for unrestricted contexts).

1. *If* $\overline{\Psi} = \overline{\Phi}$ *then* $\Delta; \mathcal{C} \vdash \Psi = \Psi \bowtie \overline{\Phi}$.
2. *If* $\overline{\Psi} = \overline{\Phi}$ *then* $\Delta; \mathcal{C} \vdash \Psi = \overline{\Phi} \bowtie \Psi$.
3. *If* $\overline{\Psi} = \overline{\Psi_1} = \overline{\Psi_2}$ *then* $\mathcal{C} \vdash \overline{\Psi} = \overline{\Psi_1} \bowtie \overline{\Psi_2}$.

3.5 Typing

We concentrate here on describing the typing for linear contextual LF terms using a bi-directional type system (see Fig.4).

All typing judgments have access to both contexts Ψ and Δ, and to a fixed well-typed signature Σ where we store constants c together with their types and kinds. Contextual Linear LF objects may depend on variables declared in the context Ψ and a fixed meta-context Δ which contains contextual variables such as meta-variables u, parameter variables p, and context variables. In addition, we carry a set $\mathcal{C}$ of context constraints. Although the rules are bi-directional, they do not give a direct algorithm, as we need to split a context Ψ into a context Ψ_1 and Ψ_2 such that $\Psi = \Psi_1 \bowtie \Psi_2$ (see rule for checking $H \cdot S$ against a base type P for example).

Typing rules presented in Figure 4 are, perhaps unsurprisingly, a fusion between Contextual LF and Linear LF. As in contextual LF the typing for meta-variable closures and parameter variable closures is straightforward. A meta-variable $u : (\Psi \vdash P)$ represents an open LF object (a "hole" in a term). As mentioned earlier it has, associated with it, a postponed substitution σ, applied as soon as u is made concrete. Similarly, a parameter variable $p : (\Psi \vdash \#A)$ represents an LF variable – either an unrestricted or linear one. The need for the latter comes from uncertainty about the position of the only linear variable in the context. It is however important that Ψ contains at most one linear variable declaration.

As in linear LF, we have two λ-abstraction rules (one introducing intuitionistic, the other linear assumptions), two corresponding variants of application, two variable cases etc. Moreover, we ensure that types only depend on the unrestricted part of a context when checking that two types are equal. As we rely on hereditary substitutions, this equality check essentially checks for syntactic equality. Similarly, when we consider a spine $M\,;S$ and check it against the dependent type $\Pi x{:}A.B$, we make sure that M has type A in the unrestricted context before continuing to check the spine S against $[M/x]_A B$.

$\boxed{\Delta; \mathcal{C}; \Psi \vdash M \Leftarrow A}$ Term M checks against type A

$$\frac{\Delta; \mathcal{C}; \Psi, x{:}A \vdash M \Leftarrow B}{\Delta; \mathcal{C}; \Psi \vdash \lambda x.M \Leftarrow \Pi x{:}A.B} \qquad \frac{\Delta; \mathcal{C}; \Psi, x\hat{:}A \vdash M \Leftarrow B}{\Delta; \mathcal{C}; \Psi \vdash \widehat{\lambda} x.M \Leftarrow A \multimap B}$$

$$\frac{u : (\Phi \vdash P) \in \Delta \quad \Delta; \mathcal{C}; \Psi \vdash \sigma \Leftarrow \Phi \quad \Delta; \mathcal{C}; \overline{\Psi} \vdash [\sigma]_{\overline{\Phi}} P = Q}{\Delta; \mathcal{C}; \Psi \vdash u[\sigma] \Leftarrow Q}$$

$$\frac{\Delta; \mathcal{C}; \Psi_1 \vdash H \Rightarrow A \quad \Delta; \mathcal{C}; \Psi_2 \vdash S > A \Rightarrow P \quad \Delta; \mathcal{C}; \overline{\Psi} \vdash P = Q \quad \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\Delta; \mathcal{C}; \Psi \vdash H \cdot S \Leftarrow Q}$$

$\boxed{\Delta; \mathcal{C}; \Psi \vdash H \Rightarrow A}$ Head H synthesizes a type A

$$\frac{c{:}A \in \Sigma \quad \mathsf{unr}(\Psi)}{\Delta; \mathcal{C}; \Psi \vdash c \Rightarrow A} \qquad \frac{p : (\Phi \vdash \#A) \in \Delta \quad \Delta; \mathcal{C}; \Psi \vdash \sigma \Leftarrow \Phi}{\Delta; \mathcal{C}; \Psi \vdash p[\sigma] \Rightarrow [\sigma]_\Phi A}$$

$$\frac{\mathsf{unr}(\Psi) \quad x{:}A \in \Psi}{\Delta; \mathcal{C}; \Psi \vdash x \Rightarrow A} \qquad \frac{\mathsf{unr}(\Psi_1) \quad \mathsf{unr}(\Psi_2)}{\Delta; \mathcal{C}; \Psi_1, x\hat{:}A, \Psi_2 \vdash x \Rightarrow A}$$

$\boxed{\Delta; \mathcal{C}; \Psi \vdash S > A \Rightarrow P}$ Spine S synthesises type P

$$\frac{}{\Delta; \mathcal{C}; \Psi \vdash \epsilon > P \Rightarrow P} \qquad \frac{\Delta; \mathcal{C}; \overline{\Psi} \vdash M \Leftarrow A \quad \Delta; \mathcal{C}; \Psi \vdash S > [M/x]_A B \Rightarrow P}{\Delta; \mathcal{C}; \Psi \vdash M\,; S > \Pi x{:}A.B \Rightarrow P}$$

$$\frac{\Delta; \mathcal{C}; \Psi_1 \vdash M \Leftarrow A \quad \Delta; \mathcal{C}; \Psi_2 \vdash S > B \Rightarrow P \quad \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\Delta; \mathcal{C}; \Psi \vdash M\,\hat{;}\,S > A \multimap B \Rightarrow P}$$

Fig. 4. Typing rules for terms

When we encounter a spine $M\,\hat{;}\,S$ and check it against the linear type $A \multimap B$ in the context Ψ, we must show that there exists a split s.t. $\Psi = \Psi_1 \bowtie \Psi_2$ given the constraints $\mathcal{C}$ and then check that the term M has type A in the context Ψ_1 and the remaining spine S is checked against B to synthesize a type B'.

Finally, we consider the typing rules for substitutions, presented in Figure 5. We excercise care in making sure the contexts present in the base cases, i.e. where the substitution is empty or the identity, are unrestricted. This guarantees weakening and contraction for unrestricted contexts.

The substitution σ, M is well-typed with domain $\Phi, x{:}A$ and range Ψ, if σ is a substitution from Φ to the context Ψ and in addition M has type $[\sigma]_\Phi A$ in the unrestricted context $\overline{\Psi}$. The substitution σ, M is well-typed with domain $\Phi, x\hat{:}A$ and range Ψ, there exists a context split $\Psi = \Psi_1 \bowtie \Psi_2$ s.t. σ is a substitution with domain Φ and range Ψ_1 and M is a well-typed term in the context Ψ_2.

$\boxed{\Delta; \mathcal{C}; \Psi \vdash \sigma \Leftarrow \Phi}$ Substitution σ maps variables in Φ to variables in Ψ

$$\frac{\mathsf{unr}(\Psi)}{\Delta; \mathcal{C}; \Psi \vdash \cdot \Leftarrow \cdot} \qquad \frac{\mathsf{unr}(\Psi^0)}{\Delta; \mathcal{C}; \psi, \Psi^0 \vdash \mathsf{id}_\psi \Leftarrow \psi} \qquad \frac{\mathsf{unr}(\Psi^0) \quad \mathcal{C} \vdash \phi \leftrightsquigarrow \psi \quad \mathcal{C} \vdash \psi \leftrightsquigarrow \gamma}{\Delta; \mathcal{C}; \overline{\phi}, \Psi^0 \vdash \mathsf{id}_{\overline{\psi}} \Leftarrow \overline{\gamma}}$$

$$\frac{\mathsf{unr}(\Psi_0) \quad \phi : G \in_{\mathcal{C}} \Delta \quad \psi : G' \in_{\mathcal{C}} \Delta \quad G^{\uparrow} = G'}{\Delta; \mathcal{C}; \phi^{\uparrow}, \Psi^0 \vdash \mathsf{il}_\psi \Leftarrow \psi} \ ?$$

$$\frac{\Delta; \mathcal{C}; \Psi \vdash \sigma \Leftarrow \Phi \quad \Delta; \mathcal{C}; \overline{\Psi} \vdash M \Leftarrow [\sigma]_{\overline{\Phi}} A}{\Delta; \mathcal{C}; \Psi \vdash \sigma, M \Leftarrow \Phi, x{:}A}$$

$$\frac{\Delta; \mathcal{C}; \Psi_1 \vdash \sigma \Leftarrow \Phi \quad \Delta; \mathcal{C}; \Psi_2 \vdash M \Leftarrow [\sigma]_{\overline{\Phi}} A \quad \Delta; \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\Delta; \mathcal{C}; \Psi \vdash \sigma, M \Leftarrow \Phi, x\hat{:}A}$$

Fig. 5. Typing rules for substitutions

3.6 Hereditary Substitution

Next we define hereditary substitution, which performs a substitution and reduces any redexes which arise along the way (see Fig. 6). Hereditary substitution allows us to only consider normal forms in our grammar and typing rules, which makes the decidability of typechecking obvious.

As usual, we annotate hereditary substitutions with an approximation of the type of the term we substitute for to guarantee termination.

$$\text{Type approximations } \alpha, \beta ::= a \mid \alpha \to \beta \mid \alpha \multimap B$$

We then define the dependency erasure operator $(-)^-$ as follows:

$$\begin{aligned} (a \cdot S)^- &= a \\ (\Pi x{:}A.B)^- &= A^- \to B^- \\ (A \multimap B)^- &= A^- \multimap B^- \end{aligned}$$

We will sometimes tacitly apply the dependency erasure operator $(-)^-$ in the following definitions. We can now define hereditary substitution, presented in Figure 6. The definition is mostly standard. As usual, when the substitution would create redexes, we proceed by hereditarily performing substitution. Termination can be readily established:

Lemma 3. *The hereditary substitutions* $[M/x]^*_\alpha(N)$ *where* $* \in \{c, s, l\}$ *and* $\mathsf{reduce}(M : \alpha, S)$ *terminate, either by failing or successfully producing a result.*

Proof. By nested induction on α primarily and N secondarily.

$\boxed{\mathsf{reduce}(M : \alpha, S) = N}$ N is the result of reducing M applied to the spine S

$$
\begin{aligned}
\mathsf{reduce}(\lambda x.M : \alpha \to \beta, (N\,;S)) &= \mathsf{reduce}([N/x]^c_\alpha M : \beta, S) \\
\mathsf{reduce}(\widehat{\lambda} x.M : \alpha \multimap \beta, (N\,\hat{;}\,S)) &= \mathsf{reduce}([N/x]^c_\alpha M : \beta, S) \\
\mathsf{reduce}(R : a, \epsilon) &= R \\
\mathsf{reduce}(M : \alpha, S) &= \bot
\end{aligned}
$$

$\boxed{[M/x]^c_\alpha N = N'}$ N' is a result of substituting M for x in a canonical term N

$$
\begin{aligned}
[M/x]^c_\alpha(\lambda y.N) &= \lambda y.N' && \text{where } [M/x]^c_\alpha N = N' \text{ choosing } y \neq x, y \notin \mathsf{FV}(M) \\
[M/x]^c_\alpha(\widehat{\lambda} y.N) &= \widehat{\lambda} y.N' && \text{where } [M/x]^c_\alpha N = N' \text{ choosing } y \neq x, y \notin \mathsf{FV}(M) \\
[M/x]^c_\alpha(u[\sigma]) &= u[\sigma'] && \text{where } [M/x]^s_\alpha \sigma = \sigma' \\
[M/x]^c_\alpha(c \cdot S) &= c \cdot S' && \text{where } [M/x]^l_\alpha S = S' \\
[M/x]^c_\alpha(x \cdot S) &= N && \text{where } [M/x]^l_\alpha S = S' \text{ and } \mathsf{reduce}(M : \alpha, S') = N \\
[M/x]^c_\alpha(y \cdot S) &= y \cdot S' && \text{where } [M/x]^l_\alpha S = S' \text{ and } x \neq y \\
[M/x]^c_\alpha(p[\sigma] \cdot S) &= p[\sigma'] \cdot S' && \text{where } [M/x]^s_\alpha \sigma = \sigma' \text{ and } [M/x]^l_\alpha S = S'
\end{aligned}
$$

$\boxed{[M/x]^l_\alpha S = S'}$ S' is a result of substituting M for x in a spine S

$$
\begin{aligned}
[M/x]^l_\alpha(\epsilon) &= \epsilon \\
[M/x]^l_\alpha(N\,;S) &= N'\,;S' && \text{where } [M/x]^c_\alpha N = N' \text{ and } [M/x]^l_\alpha S = S' \\
[M/x]^l_\alpha(N\,\hat{;}\,S) &= N'\,\hat{;}\,S' && \text{where } [M/x]^c_\alpha N = N' \text{ and } [M/x]^l_\alpha S = S'
\end{aligned}
$$

$\boxed{[M/x]^s_\alpha \sigma = \sigma'}$ σ' is a result of substituting M for x in a substitution σ

$$
\begin{aligned}
[M/x]^s_\alpha \cdot &= \cdot \\
[M/x]^s_\alpha \mathsf{id}_\psi &= \mathsf{id}_\psi \\
[M/x]^s_\alpha \mathsf{id}_{\overline{\psi}} &= \mathsf{id}_{\overline{\psi}} \\
[M/x]^s_\alpha(\sigma, N) &= \sigma', N' && \text{where } [M/x]^s_\alpha \sigma = \sigma' \text{ and } [M/x]^c_\alpha N = N'
\end{aligned}
$$

Fig. 6. Hereditary single substitution

Now we can generalize hereditary substitution to simultaneous substitutions following Cave and Pientka [2013] (see Fig. 7).

Many previous formulations of contextual LF allow substitutions to be extended with variables in addition to arbitrary terms. This is motivated by the fact that as we push σ under a binder in $\lambda x.M$, we would like to extend it by the identity mapping x to itself. However, since we do not know the type of x, simply extending σ with x may not be meaningful, since x may not be in η-long form. Therefore, previous formulations allow a substitution σ to be extended with M where M is either an η-long normal form or a variable. Unfortunately, this complicated the grammar of substitutions and means that there was not necessarily a unique representation of substitutions. This means that substitutions still had to be compared up to η equality. We avoid these issues following

$\boxed{\sigma_\Psi(x)}$ Variable lookup

$(\sigma, M)_{\Psi, x:A}(x) = M : A^-$

$(\sigma, M)_{\Psi, x\hat{:}A}(x) = M : A^-$

$(\sigma, M)_{\Psi, _}(x) = \sigma_\Psi(x)$

$\sigma_\Psi(x) = \bot$

$\boxed{[\sigma]^{\widehat{\Phi}}_{\Psi} M}$ Substitution of the variables of Ψ in a canonical term (leaving $x \in \widehat{\Phi}$ unchanged)

$[\sigma]^{\widehat{\Phi}}_{\Psi}(\lambda y.N) = \lambda y.N'$ where $[\sigma]^{\widehat{\Phi}, y}_{\Psi} N = N'$choosing $y \notin \Psi, y \notin \mathsf{FV}(\sigma)$

$[\sigma]^{\widehat{\Phi}}_{\Psi}(\widehat{\lambda} y.N) = \widehat{\lambda} y.N'$ where $[\sigma]^{\widehat{\Phi}, \widehat{y}}_{\Psi} N = N'$ choosing $y \notin \Psi, y \notin \mathsf{FV}(\sigma)$

$[\sigma]^{\widehat{\Phi}}_{\Psi}(u[\tau]) = u[\tau']$ where $[\sigma]^{\widehat{\Phi}}_{\Psi} \tau = \tau'$

$[\sigma]^{\widehat{\Phi}}_{\Psi}(c \cdot S) = c \cdot S'$ where $[\sigma]^{\widehat{\Phi}}_{\Psi} S = S'$

$[\sigma]^{\widehat{\Phi}}_{\Psi}(x \cdot S) = \mathsf{reduce}(M : \alpha, S')$ where $\Delta; \mathcal{C} \vdash \sigma : \Psi = \sigma_1 : \Psi_1 \bowtie \sigma_2 : \Psi_2, x \notin \widehat{\Phi}$
$\sigma_{1\,\Psi_1}(x) = M : \alpha$ and $[\sigma_2]^{\widehat{\Phi}}_{\Psi_2} S = S'$

$[\sigma]^{\widehat{\Phi}}_{\Psi}(y \cdot S) = y \cdot S'$ where $y \in \widehat{\Phi}$ and $[\sigma]^{\widehat{\Phi}}_{\Psi} S = S'$

$[\sigma]^{\widehat{\Phi}}_{\Psi}(p[\tau] \cdot S) = p[\tau'] \cdot S'$ where $\Delta; \mathcal{C} \vdash \sigma : \Psi = \sigma_1 : \Psi_1 \bowtie \sigma_2 : \Psi_2$,
$\Delta; \mathcal{C} \vdash \widehat{\Phi} = \widehat{\Phi}_1 \bowtie \widehat{\Phi}_2$,
$\mathsf{FLV}^{\widehat{\Phi}}(\tau) \subseteq \widehat{\Phi_1}$, $\mathsf{FLV}^{\widehat{\Phi}}(S) \subseteq \widehat{\Phi_2}$
$[\sigma_1]^{\widehat{\Phi_1}}_{\Psi_1} \tau = \tau'$ and $[\sigma_2]^{\widehat{\Phi_2}}_{\Psi_2} S = S'$

$\boxed{[\sigma]^{\widehat{\Phi}}_{\Psi} S}$ Substitution of the variables of Ψ in a spine S

$[\sigma]^{\widehat{\Phi}}_{\Psi}(\epsilon) = \epsilon$

$[\sigma]^{\widehat{\Phi}}_{\Psi}(N\,; S) = N'\,; S'$ where $[\sigma]^{\widehat{\Phi}}_{\overline{\Psi}} N = N'$ and $[\sigma]^{\widehat{\Phi}}_{\Psi} S = S'$

$[\sigma]^{\widehat{\Phi}}_{\Psi}(N\,\hat{;}\, S) = N'\,\hat{;}\, S'$ where $\Delta; \mathcal{C} \vdash \sigma : \Psi = \sigma_1 : \Psi_1 \bowtie \sigma_2 : \Psi_2$,
$\Delta; \mathcal{C} \vdash \widehat{\Phi} = \widehat{\Phi}_1 \bowtie \widehat{\Phi}_2$,
$\mathsf{FLV}^{\widehat{\Phi}}(N) \subseteq \widehat{\Phi_1}$, $\mathsf{FLV}^{\widehat{\Phi}}(S) \subseteq \widehat{\Phi_2}$
$[\sigma_1]^{\widehat{\Phi_1}}_{\Psi_1} N = N'$ and $[\sigma_2]^{\widehat{\Phi_2}}_{\Psi_2} S = S'$

$\boxed{[\sigma]^{\widehat{\Phi}}_{\Psi} \tau}$ Substitution of the variables of Ψ in a substitution τ

$[\sigma]^{\widehat{\Phi}}_{\Psi} \cdot = \cdot$

$[\sigma]^{\widehat{\Phi}}_{\Psi} \mathsf{id}_\psi = \mathsf{id}_\psi$

$[\sigma]^{\widehat{\Phi}}_{\Psi} \mathsf{id}_{\overline{\psi}} = \mathsf{id}_{\overline{\psi}}$

$[\sigma]^{\widehat{\Phi}}_{\Psi}(\tau, M) = (\tau', M')$ where $\Delta; \mathcal{C} \vdash \sigma : \Psi = \sigma_1 : \Psi_1 \bowtie \sigma_2 : \Psi_2$,
$\Delta; \mathcal{C} \vdash \widehat{\Phi} = \widehat{\Phi}_1 \bowtie \widehat{\Phi}_2$,
$\mathsf{FLV}^{\widehat{\Phi}}(N) \subseteq \widehat{\Phi_1}$, $\mathsf{FLV}^{\widehat{\Phi}}(S) \subseteq \widehat{\Phi_2}$
$[\sigma]^{\widehat{\Phi}}_{\Psi} \tau = \tau'$ and $[\sigma]^{\widehat{\Phi}}_{\Psi} M = M'$

Fig. 7. Simultaneous substitution

Cave and Pientka [2013] and require that substitutions must be extended with η-long terms, thus guaranteeing unique normal forms for substitutions.

The main complication arises when we push the substitution σ through a spine $N \,\hat{;}\, S$, as the typing for spines involves a context split. While σ has domain Ψ and $N \,\hat{;}\, S$ is well-typed in Ψ, the term N is well-typed in a context Ψ_1 and the spine S is well-typed in a context Ψ_2 where Ψ is the join of Ψ_1 and Ψ_2. As a consequence, $[\sigma]N$ and $[\sigma]S$ would be ill-typed. A context split $\Psi = \Psi_1 \bowtie \Psi_2$ induces a substitution split, i.e. $\sigma = \sigma_1 \bowtie \sigma_2$ where σ_i has domain Ψ_i. We hence define the join of substitution in a type-directed way in Fig. **??**. Computing a substitution split depends also on the constraints we might have available; we parameterize the simultaneous substitution operation with the meta-context Δ and constraints $\mathcal{C}$. As they remain fixed, we do not clutter our definition by carrying them around explicitly, but remark that we have access to them, when needed as in computing the substitution join.

If we are only given a context, splitting it is not necessarily unique; hence, splitting a substitution σ into σ_1 and σ_2 is also not necessarily unique. However, given the linear variables occurring in $N \,\hat{;}\, S$, the substitution σ_1 must provide instantiations for all the free variables in N and σ_2 must provide instantiations for all the free variables in S. In particular, if we started with a well-typed spine $N \,\hat{;}\, S$, then we know the linear free variables in N are distinct from the linear free variables in S. We hence compute the linear free variables using $\mathsf{FLV}_\Psi(N)$ (and $\mathsf{FLV}_\Psi \widehat{\Phi}(S)$ resp.), extracting a subset of linear variables from Ψ and then check that they are a subset of the variables declared in the context Ψ_i (where $i \in \{1, 2\}$. This subset relation only succeeds, if all the linear variables in Ψ_1 are also in the $\mathsf{FLV}_\Psi \widehat{\Phi}(N)$ and all the linear variables in Ψ_2 are also in the $\mathsf{FLV}_\Psi \widehat{\Phi}(S)$. In other words, the only difference between Ψ_i and the set of free linear variables we computed are the unrestricted variables. We proceed similarly for $\widehat{\Phi}$. The operation $\mathsf{FLV}^{\widehat{\Phi}}(N)$ extracts from $\widehat{\Phi}$ those variables that are linear and occur in N.

Lemma 4.

1. *If* $\Delta; \mathcal{C}; \Psi \vdash \sigma \Leftarrow \Phi$ *and* $\mathsf{unr}(\Phi)$ *then for any* M *such that* $\sigma(x) = M : \alpha$, $\Delta; \mathcal{C}; \Psi \vdash M \Leftarrow A$ *where* $x{:}A \in \Phi$.
2. *If* $\Delta; \mathcal{C}; \Psi \vdash \sigma \Leftarrow \Phi$ *and* $\Phi = \Phi_1, x \hat{:} A, \Phi_2$ *where* $\mathsf{unr}(\Phi_i)$ *then if* $\sigma(x)M : \alpha$, $\Delta; \mathcal{C}; \Psi \vdash M \Leftarrow A$.

Lemma 5 (Context Joins).

1. *If* $\Delta; \mathcal{C}; \overline{\Psi} \vdash J$ *and* $\Delta; \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2$ *then* $\Delta; \mathcal{C}; \overline{\Psi}_1 \vdash J$.
2. *If* $\Delta; \mathcal{C}; \overline{\Psi} \vdash J$ *and* $\Delta; \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2$ *then* $\Delta; \mathcal{C}; \overline{\Psi}_2 \vdash J$.

Lemma 6 (Existence of Substitution Splits). *If* $\Delta; \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2$ *and* $\Delta; \mathcal{C}; \Phi \vdash \sigma \Leftarrow \Psi$ *then there exists substitutions* σ_1, σ_2 *and contexts* Φ_1 *and* Φ_2, *s.t.* $\Delta; \mathcal{C}; \Phi_1 \vdash \sigma_1 \Leftarrow \Psi_1$, *and* $\Delta; \mathcal{C}; \Phi_2 \vdash \sigma_2 \Leftarrow \Psi_2$
and $\Delta; \mathcal{C} \vdash \sigma : \Psi = \sigma_1 : \Psi_1 \bowtie \sigma_2 : \Psi_2$ *and* $\Delta; \mathcal{C} \vdash \Phi = \Phi_1 \bowtie \Phi_2$

$\boxed{\mathcal{C} \vdash \sigma : \Psi = \sigma_1 : \Psi_1 \bowtie \sigma_2 : \Psi_2}$ Substitution σ with domain Ψ is the join of σ_1 with domain Ψ_1 and σ_2 with domain Ψ_2 given the constraints $\mathcal{C}$

$$\frac{}{\mathcal{C} \vdash \cdot : \cdot = \cdot : \cdot \bowtie \cdot : \cdot} \qquad \frac{\mathcal{C} \vdash \psi \leftrightsquigarrow \phi}{\mathcal{C} \vdash \mathsf{id}_\psi : \psi = \mathsf{id}_\psi : \psi \bowtie \mathsf{id}_{\overline{\phi}} : \overline{\phi}} \qquad \frac{\mathcal{C} \vdash \psi \leftrightsquigarrow \phi}{\mathcal{C} \vdash \mathsf{id}_\psi : \psi = \mathsf{id}_{\overline{\phi}} : \overline{\phi} \bowtie \mathsf{id}_\psi : \psi}$$

$$\frac{(\psi := \psi_1 \bowtie \psi_2) \in \mathcal{C}}{\mathcal{C} \vdash \mathsf{id}_\psi : \psi = \mathsf{id}_{\psi_1} : \psi_1 \bowtie \mathsf{id}_{\psi_2} : \psi_2} \qquad \frac{\mathcal{C} \vdash \psi \leftrightsquigarrow \psi_1 \quad \mathcal{C} \vdash \psi \leftrightsquigarrow \psi_2}{\mathcal{C} \vdash \mathsf{id}_{\overline{\psi}} : \overline{\psi} = \mathsf{id}_{\overline{\psi_1}} : \overline{\psi}_1 \bowtie \mathsf{id}_{\overline{\psi}_2} : \overline{\psi}_2}$$

$$\frac{\mathcal{C} \vdash \sigma : \Psi = \sigma_1 : \Psi_1 \bowtie \sigma_2 : \Psi_2}{\mathcal{C} \vdash (\sigma, M) : \Psi, x{:}A = \sigma_1, M : \Psi_1, x{:}A \bowtie \sigma_2, M : \Psi_2, x{:}A}$$

$$\frac{\mathcal{C} \vdash \sigma : \Psi = \sigma_1 : \Psi_1 \bowtie \sigma_2 : \Psi_2}{\mathcal{C} \vdash \sigma, M : \Psi, x\hat{:}A = \sigma_1, M : \Psi_1, x\hat{:}A \bowtie \sigma_2 : \Psi_2} \qquad \frac{\mathcal{C} \vdash \sigma : \Psi = \sigma_1 : \Psi_1 \bowtie \sigma_2 : \Psi_2}{\mathcal{C} \vdash \sigma, M : \Psi, x\hat{:}A = \sigma_1 : \Psi_1 \bowtie \sigma_2, M : \Psi_2, x\hat{:}A}$$

$$\frac{\mathcal{C} \equiv \mathcal{C}' \quad \psi := \psi_1 \bowtie \psi_2 \in \mathcal{C}'}{\mathcal{C} \vdash \mathsf{id}_\psi : \psi = \mathsf{id}_{\psi_1} : \psi_1 \bowtie \mathsf{id}_{\psi_2} : \psi_2}$$

Fig. 8. Type-directed rules for joining substitutions

Proof. Proof by induction on typing derivation $\mathcal{D} :: \Delta; \mathcal{C}; \Psi \vdash \sigma \Leftarrow \Phi$. We show a few cases below, the remaining ones are similar.

Case $\mathcal{D} = \dfrac{\mathsf{unr}(\Psi)}{\Delta; \mathcal{C}; \Psi \vdash \cdot \Leftarrow \cdot}$

$\Delta; \mathcal{C} \vdash \cdot = \Psi_1 \bowtie \Psi_2$ by assumption
$\Psi_1 = \cdot$ and $\Psi_2 = \cdot$ by inversion
Let $\sigma_1 = \cdot$, $\sigma_2 = \cdot$, $\Phi_1 = \Psi$ and $\Phi_2 = \Psi$
$\Delta; \mathcal{C} \vdash \Psi = \Psi \bowtie \Psi$ by reflexivity lemma
$\Delta; \mathcal{C}; \Psi \vdash \cdot \Leftarrow \cdot$ by typing rule
$\Delta; \mathcal{C} \vdash \cdot : \cdot = \cdot : \cdot \bowtie \cdot : \cdot$ by subst. join

Case $\mathcal{D} = \dfrac{\mathsf{unr}(\Psi^0)}{\Delta; \mathcal{C}; \psi, \Psi^0 \vdash \mathsf{id}_\psi \Leftarrow \psi}$ and $\mathcal{E} = \dfrac{\mathcal{C} \vdash \psi \leftrightsquigarrow \phi}{\mathcal{C} \vdash \psi = \psi \bowtie \overline{\phi}}$

Let $\sigma_1 = \mathsf{id}_\psi$, $\sigma_2 = \mathsf{id}_{\overline{\phi}}$, $\Phi_1 = \psi, \Psi^0$, and $\Phi_2 = \overline{\phi}, \Psi^0$ where $\mathsf{unr}(\Psi^0)$
$\Delta; \mathcal{C}; \psi, \Psi^0 \vdash \mathsf{id}_\psi \Leftarrow \psi$ by typing rule
$\Delta; \mathcal{C}; \overline{\phi}, \Psi^0 \vdash \mathsf{id}_{\overline{\phi}} \Leftarrow \overline{\psi}$ by typing rule
$\Delta; \mathcal{C} \vdash \psi, \Psi^0 = \psi, \Psi^0 \bowtie \overline{\phi}, \Phi^0$ by ctx. join lemma 1(5)
$\Delta; \mathcal{C} \vdash \mathsf{id}_\psi : \psi = \mathsf{id}_\psi : \psi \bowtie \mathsf{id}_{\overline{\phi}} : \overline{\phi}$ by subst. join rules

Case $\mathcal{D} = \dfrac{\mathsf{unr}(\Psi^0)}{\Delta; \mathcal{C}; \psi, \Psi^0 \vdash \mathsf{id}_\psi \Leftarrow \psi}$ and $\mathcal{E} = \dfrac{\mathcal{C} \equiv \mathcal{C}' \qquad (\psi := \psi_1 \bowtie \psi_2) \in \mathcal{C}'}{\mathcal{C} \vdash \psi = \psi_1 \bowtie \psi_2}$

Let $\sigma_1 = \mathsf{id}_{\psi_1}$ and $\sigma_2 = \mathsf{id}_{\psi_2}$ and $\Phi_1 = \psi_1, \Psi^0$ and $\Phi_2 = \psi_2, \Psi^0$ where $\mathsf{unr}(\Psi^0)$
$\mathcal{C} \vdash \psi, \Psi^0 = \psi_1, \Psi^0 \bowtie \psi_2, \Psi^0$ by ctx. join lemma 1(5)
$\mathcal{C} \vdash \mathsf{id}_\psi : \psi = \mathsf{id}_{\psi_1} : \psi_1 \bowtie \mathsf{id}_{\psi_2} : \psi_2$ by subst. join rule
$\Delta; \mathcal{C}; \psi_1, \Psi^0 \vdash \mathsf{id}_{\psi_1} \Leftarrow \psi_1$ by typing rule
$\Delta; \mathcal{C}; \psi_2, \Psi^0 \vdash \mathsf{id}_{\psi_2} \Leftarrow \psi_2$ by typing rule

$$\textit{Case } \mathcal{D} = \frac{\Delta; \mathcal{C}; \Phi \vdash \sigma \Leftarrow \Psi \qquad \Delta; \mathcal{C}; \overline{\Phi} \vdash M \Leftarrow [\sigma]_\Psi A}{\Delta; \mathcal{C}; \Phi \vdash \sigma, M \Leftarrow \Psi, x{:}A}$$

$$\text{and} \quad \mathcal{E} = \frac{\Delta; \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\Delta; \mathcal{C} \vdash \Psi, x{:}A = \Psi_1, x{:}A \bowtie \Psi_2, x{:}A}$$

There exist substitutions σ_1' and σ_2', and contexts Φ_1' and Φ_2' s.t.
$\Delta; \mathcal{C}; \Phi_1' \vdash \sigma_1' \Leftarrow \Psi_1$
$\Delta; \mathcal{C}; \Phi_2' \vdash \sigma_2' \Leftarrow \Psi_2$
$\Delta; \mathcal{C} \vdash \Phi = \Phi_1' \bowtie \Phi_2'$
$\Delta; \mathcal{C} \vdash \sigma : \Phi = \sigma_1' : \Phi_1' \bowtie \sigma_2' : \Phi_2'$ by IH
$\Delta; \mathcal{C}; \overline{\Phi_i'} \vdash M \Leftarrow [\sigma]_\Psi A$ as $\overline{\Phi} = \overline{\Phi_i'}$
Let $\sigma_1 = \sigma_1', M$, $\sigma_2 = \sigma_2', M$, $\Phi_1 = \Phi_1'$, and $\Phi_2 = \Phi_2'$
$\Delta; \mathcal{C}; \Phi_i' \vdash (\sigma_i', M) \Leftarrow \Psi_i, x{:}A$ by typing rules
$\Delta; \mathcal{C} \vdash (\sigma, M) : \Psi, x{:}A = (\sigma_1', M) : \Psi_1, x{:}a \bowtie (\sigma_2', M) : \Psi_2, x{:}A$ by subst. joins

$$\textit{Case } \mathcal{D} = \frac{\Delta; \mathcal{C}; \Phi_1 \vdash \sigma \Leftarrow \Psi \quad \Delta; \mathcal{C}; \Phi_2 \vdash M \Leftarrow [\sigma]_\Psi A \quad \Delta; \mathcal{C} \vdash \Phi = \Phi_1 \bowtie \Phi_2}{\Delta; \mathcal{C}; \Phi \vdash \sigma, M \Leftarrow \Psi, x\hat{:}A}$$

$$\text{and} \quad \mathcal{E} = \frac{\Delta; \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\Delta; \mathcal{C} \vdash \Psi, x\hat{:}A = \Psi_1, x\hat{:}A \bowtie \Psi_2}$$

There exist substitutions σ_1' and σ_2', and contexts Φ_{11}' and Φ_{12}' s.t.
$\Delta; \mathcal{C}; \Phi_{11}' \vdash \sigma_1' \Leftarrow \Psi_1$
$\Delta; \mathcal{C}; \Phi_{12}' \vdash \sigma_2' \Leftarrow \Psi_2$
$\Delta; \mathcal{C} \vdash \Phi_1 = \Phi_{11}' \bowtie \Phi_{12}'$
$\Delta; \mathcal{C} \vdash \sigma : \Phi_1 = \sigma_1' : \Phi_{11}' \bowtie \sigma_2' : \Phi_{12}'$ by IH
$\Delta; \mathcal{C} \vdash (\sigma, M) : (\Psi, x\hat{:}A) = (\sigma_1', M) : (\Psi_1, x\hat{:}A) \bowtie \sigma_2' : \Psi_2$ by subst. joins
$\Delta; \mathcal{C} \vdash \Phi = \Phi_1 \bowtie \Phi_2$ and $\Delta; \mathcal{C} \vdash \Phi_1 = \Phi_{11}' \bowtie \Phi_{12}'$ by previous lines
There exists a context Φ_0 and $\mathcal{C}'$ s.t. $\mathcal{C} \equiv \mathcal{C}'$ and $\Delta; \mathcal{C}' \vdash \Phi_0 = \Phi_{11}' \bowtie \Phi_2$ and
$\Delta; \mathcal{C}' \vdash \Phi = \Phi_0 \bowtie \Phi_{12}'$ by Transitivity Lemma
$\Delta; \mathcal{C} \vdash \Phi_0 = \Phi_{11}' \bowtie \Phi_2$ by Equiv. Rule
$\Delta; \mathcal{C}; \Phi_0 \vdash \sigma_1', M \Leftarrow \Psi_1, x\hat{:}A$ by typing rule
$\Delta; \mathcal{C} \vdash \Phi = \Phi_0 \bowtie \Phi_{12}'$ by Equiv. Rule

Lemma 7 (Substitution lemma).

1. *If* $\Delta; \mathcal{C}; \Psi_0 \vdash M \Leftarrow A$ *and* $\Delta; \mathcal{C}; \Psi_1, D, \Phi \vdash J$, $D \in \{x{:}A, x\hat{:}A\}$ *and* $\Delta; \mathcal{C} \vdash \Psi = \Psi_0 \bowtie \Psi_1$ *then* $\Delta; \mathcal{C}; \Psi, [M/x]_A \Phi \vdash [M/x]^*_A(J)$ *where* $* \in \{c, s, l\}$.

2. *If* $\Delta; \mathcal{C}; \Psi_1 \vdash M \Leftarrow A$ *and* $\Delta; \mathcal{C}; \Psi_2 \vdash S > A \Rightarrow B$ *and* $\Delta; \mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2$ *and* $\mathsf{reduce}(M : A^-, S) = M'$ *then* $\Delta; \mathcal{C}; \Psi \vdash M' \Leftarrow B$

Proof. To do

Lemma 8 (Simultaneous substitution lemma).
If $\Delta; \mathcal{C}; \Psi, \Phi \vdash J$ *and* $\Delta; \mathcal{C}; \Psi' \vdash \sigma \Leftarrow \Psi$ *then* $\Delta; \mathcal{C}; \Psi', [\sigma]^{\cdot}_{\Psi}\Phi \vdash [\sigma]^{\widehat{\Phi}}_{\Psi}(J)$.

Proof. Proof by induction on the first typing derivation. Below we show one case. Others to be filled in.

$$\textit{Case } \mathcal{D} = \frac{\Delta; \mathcal{C}; \Psi_1 \vdash M \Leftarrow A \quad \Delta; \mathcal{C}; \Psi_2 \vdash S > B \Rightarrow P \quad \mathcal{C} \vdash \Psi, \Phi = \Psi_1 \bowtie \Psi_2}{\Delta; \mathcal{C}; \Psi, \Phi \vdash M \,\hat{;}\, S > A \multimap B \Rightarrow P}$$

$\Delta; \mathcal{C}; \Gamma \vdash \sigma \Leftarrow \Psi$ by assumption
There exists Φ_1 and Φ_2 s.t. $\mathcal{C} \vdash \Phi = \Phi_1 \bowtie \Phi_2$ and $\Psi_1 = \Psi_1', \Phi_1$ and $\Psi_2 = \Psi_2', \Phi_2$ and $\mathcal{C} \vdash \Psi, \Phi = (\Psi_1', \Phi_1) \bowtie (\Psi_2', \Phi_2)$ by inversion lemma on ctx joins
$\Delta; \mathcal{C} \vdash \Psi = \Psi_1' \bowtie \Psi_2'$ by inversion lemma on ctx joins
$\Delta; \mathcal{C}; \Gamma_1 \vdash \sigma_1 \Leftarrow \Psi_1'$
$\Delta; \mathcal{C}; \Gamma_2 \vdash \sigma_2 \Leftarrow \Psi_2'$
$\mathcal{C} \vdash \Gamma = \Gamma_1 \bowtie \Gamma_2$
$\mathcal{C} \vdash \sigma : \Psi = \sigma_1 : \Psi_1' \bowtie \sigma_2 : \Psi_2'$ by existence of substitution split (Lemma 6)
$\Delta; \mathcal{C}; (\Psi_1', \Phi_1) \vdash M \Leftarrow A$ by premise
$\Delta; \mathcal{C}; \Gamma_1, [\sigma_1]^{\cdot}_{\Psi_1'}\Phi_1 \vdash [\sigma_1]^{\widehat{\Phi_1}}_{\Psi_1'} M \Leftarrow [\sigma_1]^{\widehat{\Phi_1}}_{\Psi_1'} A$ by IH
$\Delta; \mathcal{C}; (\Psi_2', \Phi_2) \vdash S > B \Rightarrow P$ by premise
$\Delta; \mathcal{C}; \Gamma_2, [\sigma_2]^{\cdot}_{\Psi_2'}\Phi_2 \vdash [\sigma_2]^{\widehat{\Phi_2}}_{\Psi_2'} S > [\sigma_2]^{\widehat{\Phi_2}}_{\Psi_2'} B \Rightarrow [\sigma_2]^{\widehat{\Phi_2}}_{\Psi_2'} P$ by IH
$\mathcal{C} \vdash [\sigma]^{\cdot}_{\Psi}\Phi = [\sigma_1]^{\cdot}_{\Psi_1'}\Phi_1) \bowtie [\sigma_2]^{\cdot}_{\Psi_2'}\Phi_2)$ by subst. prop. of ctx joins
$\mathcal{C} \vdash (\Gamma,\ [\sigma]^{\cdot}_{\Psi}\Phi) = (\Gamma_1,\ [\sigma_1]^{\cdot}_{\Psi_1'}\Phi_1) \bowtie (\Gamma_2,\ [\sigma_2]^{\cdot}_{\Psi_2'}\Phi_2)$ by ctx. join lemma 1(5)
let $M' = [\sigma_1]^{\widehat{\Phi_1}}_{\Psi_1'} M$ and $S' = [\sigma_2]^{\widehat{\Phi_2}}_{\Psi_2'} S$
and $A' = [\sigma_1]^{\widehat{\Phi_1}}_{\Psi_1'} A$, and $B' = [\sigma_2]^{\widehat{\Phi_2}}_{\Psi_2'} B$, and $P' = [\sigma_2]^{\widehat{\Phi_2}}_{\Psi_2'} P$
We note that types only depend on the unrestricted parts in a context, and the unrestricted parts of the context is shared among context joins; hence we have:
$[\sigma]^{\widehat{\Phi}}_{\Psi} A = A'$, $[\sigma]^{\widehat{\Phi}}_{\Psi} B = B'$, and $[\sigma]^{\widehat{\Phi}}_{\Psi} P = P'$
$[\sigma]^{\widehat{\Phi}}_{\Psi}(M \,\hat{;}\, S) = M' \,\hat{;}\, S'$ by subst. definition
$\Delta; \mathcal{C}; (\Gamma,\ [\sigma]^{\cdot}_{\Psi}\Phi) \vdash [\sigma]^{\widehat{\Phi}}_{\Psi}(M \,\hat{;}\, S) > [\sigma]^{\widehat{\Phi}}_{\Psi}(A \multimap B) \Rightarrow [\sigma]^{\widehat{\Phi}}_{\Psi} P$

3.7 Meta-Substitutions

We define next the meta-substitution operation (see Fig. 9) where we replace a contextual variable X with a meta-object. We annotate the meta-substitution with the type of the object it is substituting, since this information needs to be passed on to the hereditary substitution. The case for meta-variables and

Reword it to describe simult.meta-subst. [am]

We will also remember $\mathcal{C}$ and $\mathcal{C}'$ now [am]

parameter variables are standard. Replacing a meta-variable u with a term $\widehat{\Psi}.R$ in a term M is straightforward: we scan M for the occurrence of a closure $u[\sigma]$ and then replace the closure by $[\sigma]_\Psi R$. Similarly replacing a parameter variable p with a variable object $\widehat{\Psi}.H$ in a term M is straightforward: we scan M for the occurrence of a closure $p[\sigma]$ and then replace the closure by $[\sigma]_\Psi H$. Note that this might trigger other reductions.

The most interesting aspect however is replacing a context variable ψ with a context Ψ. There are several aspects that we must carefully consider. First, replacing a context variable ψ with a concrete context Ψ in an identity substitution id_ψ or $\mathsf{id}_{\overline{\psi}}$ requires us to unfold the substitution. This operation can be inductively defined as follows:

$\boxed{\mathsf{id}(\Psi)}$ Identity substitution

$$\begin{aligned}
\mathsf{id}(\cdot) &= \cdot \\
\mathsf{id}(\mathsf{id}_\psi) &= \mathsf{id}_\psi \\
\mathsf{id}(\mathsf{id}_{\overline{\psi}}) &= \mathsf{id}_{\overline{\psi}} \\
\mathsf{id}(\Psi, x{:}A) &= \mathsf{id}(\Psi), \eta\text{-}\mathsf{exp}_{(\mathsf{A}^-)}(x, \epsilon) \\
\mathsf{id}(\Psi, x\hat{:}A) &= \mathsf{id}(\Psi), \eta\text{-}\mathsf{exp}_{(\mathsf{A}^-)}(x, \epsilon)
\end{aligned}$$

$\boxed{\eta\text{-}\mathsf{exp}_{\mathsf{A}}(H, S)}$ η-expansion

$$\begin{aligned}
\eta\text{-}\mathsf{exp}_{\mathsf{a}}(H, S) &= H \cdot S \\
\eta\text{-}\mathsf{exp}_{\alpha\to\beta}(H, S) &= \lambda x.\eta\text{-}\mathsf{exp}_\beta(H, S@\eta\text{-}\mathsf{exp}_\alpha(x)) \\
\eta\text{-}\mathsf{exp}_{\alpha\multimap\beta}(H, S) &= \widehat{\lambda} x.\eta\text{-}\mathsf{exp}_\beta(H, S\widehat{@}\eta\text{-}\mathsf{exp}_\alpha(x))
\end{aligned}$$

Note that we η-expand a variable along the way.

Second, if we are replacing a variable that is not in the domain of Δ, i.e. is not free in the set of constraints $\mathcal{C}$, we have to calculate the context to substitute. For this we use the following operation:

$\boxed{\mathsf{ctx\text{-}subst}_{\mathcal{C}'}(\psi; \Theta; \mathcal{C})}$ Context variable substitution

$$\begin{aligned}
\mathsf{ctx\text{-}subst}_{\mathcal{C}'}(\psi; \Theta; (\mathcal{C}, \psi := \psi_1 \bowtie \psi_2)) &= \text{let } \Psi_1 = \mathsf{ctx\text{-}subst}_{\mathcal{C}'}(\psi_1; \Theta; \mathcal{C}) \\
&\quad\ \text{let } \Psi_2 = \mathsf{ctx\text{-}subst}_{\mathcal{C}'}(\psi_2; \Theta; \mathcal{C}) \\
&\quad\ \text{return } \Psi \text{ such that } \mathcal{C}' \vdash \Psi = \Psi_1 \bowtie \Psi_2 \\
\mathsf{ctx\text{-}subst}_{\mathcal{C}'}(\psi; (\Theta, \Psi/\psi, \Theta'); \mathcal{C}) &= \Psi \\
\mathsf{ctx\text{-}subst}_{\mathcal{C}'}(\psi; \Theta; \mathcal{C}) &= \bot
\end{aligned}$$

Lemma 9.

1. *For any Θ, $[\![\Theta]\!]\overline{\Psi} = \overline{[\![\Theta]\!]\Psi}$.*
2. *For any Θ, if $\mathsf{unr}(\Psi)$ then $\mathsf{unr}([\![\Theta]\!]\Psi)$.*

Lemma 10. *For any Φ, Ψ^0, if $\mathsf{unr}(\Psi^0)$ then $\Delta; \mathcal{C}; \Phi, \Psi^0 \vdash \mathsf{id}(\Phi) \Leftarrow \Phi$*

Definition 4 (Correctness of meta-substitution).
Meta-substitution $\Delta; \mathcal{C} \vdash \Theta \Leftarrow \Delta'$ is correct with respect to the set of constraints $\mathcal{C}'$ associated with Δ', if $\mathsf{ctx\text{-}subst}_{\mathcal{C}}(\psi; \Theta; \mathcal{C}')$ exists for every $\psi := \psi_1 \bowtie \psi_2 \in \mathcal{C}'$.

Lemma 11. *If $\Delta; \mathcal{C} \vdash \Theta \Leftarrow \Delta'$ and Θ is correct with respect to $\mathcal{C}'$, then for any $\mathcal{C}' \equiv \mathcal{C}''$, Θ is correct with respect to $\mathcal{C}''$.*

$\boxed{\Theta_\Delta(X)}$ Meta-variable lookup

$$
\begin{aligned}
&(\Theta, C/X)_{\Delta,X:U}(X) = C : U \\
&(\Theta, C/Y)_{\Delta,Y:_}(X) = \Theta_\Delta(X) \\
&\Theta_\Delta(X) \qquad\qquad\quad = \bot
\end{aligned}
$$

$\boxed{[\![\Theta]\!]_\Delta M}$ Simultaneous meta-substitution for terms

$$
\begin{array}{lll}
[\![\Theta]\!]_\Delta(\lambda x.M) & = \lambda x.M' & \text{where } [\![\Theta]\!]_\Delta M = M' \\
[\![\Theta]\!]_\Delta(\widehat{\lambda} x.M) & = \widehat{\lambda} x.M' & \text{where } [\![\Theta]\!]_\Delta M = M' \\
[\![\Theta]\!]_\Delta(u[\sigma]) & = R' & \text{where } \Theta_\Delta(u) = \widehat{\Psi}.R : [\Psi \vdash P] \\
 & & \text{and } [\![\Theta]\!]_\Delta\, \sigma = \sigma' \\
 & & \text{and } [\sigma']_\Psi R = R' \\
[\![\Theta]\!]_\Delta(c \cdot S) & = c \cdot S' & \text{where } [\![\Theta]\!]_\Delta\, S = S' \\
[\![\Theta]\!]_\Delta(x \cdot S) & = x \cdot S' & \text{where } [\![\Theta]\!]_\Delta\, S = S' \\
[\![\Theta]\!]_\Delta(p[\sigma] \cdot S) & = M' & \text{where } \Theta_\Delta(p) = \widehat{\Psi}.x : [\Psi \vdash \#A] \\
 & & \text{and } [\![\Theta]\!]_\Delta\, \sigma = \sigma' \\
 & & \text{and } [\![\Theta]\!]_\Delta\, S = S' \\
 & & \text{and } \sigma'_\Psi(x) = M : \alpha \\
 & & \text{and } \mathsf{reduce}(M : \alpha, S') = M' \\
[\![\Theta]\!]_\Delta(p[\sigma] \cdot S) & = q[\tau'] \cdot S' & \text{where } \Theta_\Delta(p) = \widehat{\Psi}.q[\pi] : [\Psi \vdash \#A] \\
 & & \text{and } [\![\Theta]\!]_\Delta\, \sigma = \sigma' \\
 & & \text{and } [\![\Theta]\!]_\Delta\, S = S' \\
 & & \text{and } [\sigma']\pi = \tau
\end{array}
$$

$\boxed{[\![\Theta]\!]_\Delta \sigma}$ Simultaneous meta-substitution for substitutions

$$
\begin{array}{lll}
[\![\Theta]\!]_\Delta(\cdot) & = \cdot & \\
[\![\Theta]\!]_\Delta(\mathsf{id}_\psi) & = \mathsf{id}(\Psi) & \text{where } \mathsf{ctx\text{-}subst}_{\mathcal{C}'}(\psi; \Theta; \mathcal{C}) = \Psi \\
[\![\Theta]\!]_\Delta(\mathsf{id}_{\overline{\psi}}) & = \mathsf{id}(\overline{\Psi}) & \text{where } \mathsf{ctx\text{-}subst}_{\mathcal{C}'}(\psi; \Theta; \mathcal{C}) = \Psi \\
[\![\Theta]\!]_\Delta(\sigma; M) & = \sigma', M' & \text{where } [\![\Theta]\!]_\Delta\, \sigma = \sigma' \\
 & & \text{and } [\![\Theta]\!]_\Delta\, M = M'
\end{array}
$$

$\boxed{[\![\Theta]\!]_\Delta \Psi}$ Simultaneous meta-substitution for contexts

$$
\begin{array}{lll}
[\![\Theta]\!]_\Delta(\cdot) & = \cdot & \\
[\![\Theta]\!]_\Delta(\psi) & = \Psi & \text{where } \mathsf{ctx\text{-}subst}_{\mathcal{C}'}(\psi; \Theta; \mathcal{C}) = \Psi \\
[\![\Theta]\!]_\Delta(\overline{\psi}) & = \overline{\Psi} & \text{where } \mathsf{ctx\text{-}subst}_{\mathcal{C}'}(\psi; \Theta; \mathcal{C}) = \Psi \\
[\![\Theta]\!]_\Delta(\Psi, x{:}A) & = \Psi', x{:}A' & \text{where } [\![\Theta]\!]_\Delta\, \Psi = \Psi' \\
 & & \text{and } [\![\Theta]\!]_\Delta\, A = A' \\
[\![\Theta]\!]_\Delta(\Psi, x\hat{:}A) & = \Psi', x\hat{:}A' & \text{where } [\![\Theta]\!]_\Delta\, \Psi = \Psi' \\
 & & \text{and } [\![\Theta]\!]_\Delta\, A = A'
\end{array}
$$

Fig. 9. Simultaneous meta-substitution

Lemma 12. *If $\Delta; \mathcal{C} \vdash \Theta \Leftarrow \Delta'$, $\mathcal{C}' \vdash \psi \leftrightsquigarrow \phi$ and Θ is correct with respect to $\mathcal{C}'$, then $\overline{[\![\Theta]\!]\psi} = \overline{[\![\Theta]\!]\phi}$.*

Lemma 13.

1. $[\![\Theta]\!][M/x]J = [[\![\Theta]\!]M/x]([\![\Theta]\!]J)$.
2. $[\![\Theta]\!][\sigma]J = [[\![\Theta]\!]\sigma]([\![\Theta]\!]J)$

From now on we will permute substitutions silently.

Lemma 14. *If $\Delta; \mathcal{C} \vdash \Theta \Leftarrow \Delta'$, $\mathcal{C}' \vdash \Psi = \Psi_1 \bowtie \Psi_2$ and Θ is correct with respect to $\mathcal{C}'$, then $\mathcal{C} \vdash [\![\Theta]\!]_{\Delta'}\Psi = [\![\Theta]\!]_{\Delta'}\Psi_1 \bowtie [\![\Theta]\!]_{\Delta'}\Psi_2$*

Proof. Proof by induction on typing derivation $\mathcal{D} :: \mathcal{C}' \vdash \Psi = \Psi_1 \bowtie \Psi_2$. We show a few representative cases below.

$$\textit{Case } \mathcal{D} = \frac{(\psi := \psi_1 \bowtie \psi_2) \in \mathcal{C}'}{\mathcal{C}' \vdash \psi = \psi_1 \bowtie \psi_2}$$

Directly from the definition of correct meta-substitution.

$$\textit{Case } \mathcal{D} = \frac{\mathcal{C}' \vdash \psi \leftrightsquigarrow \phi}{\mathcal{C}' \vdash \psi = \psi \bowtie \overline{\phi}}$$

Using Lemmas 12 and **??**.1.

$$\textit{Case } \mathcal{D} = \frac{\mathcal{C}' \vdash \psi \leftrightsquigarrow \psi_1 \quad \mathcal{C}' \vdash \psi \leftrightsquigarrow \psi_2}{\mathcal{C}' \vdash \overline{\psi} = \overline{\psi}_1 \bowtie \overline{\psi}_2}$$

Using Lemmas 12 and **??**.3.

$$\textit{Case } \mathcal{D} = \frac{\mathcal{C}' \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\mathcal{C}' \vdash \Psi, x{:}A = \Psi_1, x{:}A \bowtie \Psi_2, x{:}A}$$

Using induction hypothesis.

$$\textit{Case } \mathcal{D} = \frac{\mathcal{C}' \equiv \mathcal{C}_0 \quad (\psi := \psi_1 \bowtie \psi_2) \in \mathcal{C}_0}{\mathcal{C}' \vdash \psi = \psi_1 \bowtie \psi_2}$$

Using Lemmas 11, **??** and the definition of correct meta-substitution.

Lemma 15. *If $\Delta; \mathcal{C} \vdash \Theta \Leftarrow \Delta'$, Θ is correct with respect to $\mathcal{C}'$, and $\mathsf{unr}(\Psi)$ then also $\mathsf{unr}([\![\Theta]\!]\Psi)$.*

Theorem 1 (Simultaneous meta-substitution property).
If $\Delta; \mathcal{C} \vdash \Theta \Leftarrow \Delta'$, Θ is correct with respect to $\mathcal{C}'$ and $\Delta'; \mathcal{C}'; \Psi \vdash J$, then $\Delta; \mathcal{C}; [\![\Theta]\!]_{\Delta'}\Psi \vdash [\![\Theta]\!]_{\Delta'}J$.

Proof. Proof by induction on typing derivation $\mathcal{D} :: \Delta'; \mathcal{C}'; \Psi \vdash J$.

$$\textit{Case } \mathcal{D} = \frac{\Delta'; \mathcal{C}'; \Psi, x{:}A \vdash M \Leftarrow B}{\Delta'; \mathcal{C}'; \Psi \vdash \lambda x.M \Leftarrow \Pi x{:}A.B}$$

Let $\Psi' = [\![\Theta]\!]\Psi$, $M' = [\![\Theta]\!]M$, $A' = [\![\Theta]\!]A$, $B' = [\![\Theta]\!]B$

$\Delta; \mathcal{C}; \Psi', x{:}A' \vdash M' \Leftarrow B'$ by IH

$\Delta; \mathcal{C}; \Psi' \vdash \lambda x.M' \Leftarrow \Pi x{:}A.'B'$ by typing rule

$$\textit{Case } \mathcal{D} = \frac{\Delta; \mathcal{C}; \Psi, x\hat{:}A \vdash M \Leftarrow B}{\Delta; \mathcal{C}; \Psi \vdash \widehat{\lambda} x.M \Leftarrow A \multimap B}$$

Let $\Psi' = [\![\Theta]\!]\Psi$, $M' = [\![\Theta]\!]M$, $A' = [\![\Theta]\!]A$, $B' = [\![\Theta]\!]B$

$\Delta; \mathcal{C}; \Psi', x\hat{:}A' \vdash M' \Leftarrow B'$ by IH

$\Delta; \mathcal{C}; \Psi' \vdash \widehat{\lambda} x.M' \Leftarrow A' \multimap B'$ by typing rule

$$\textit{Case } \mathcal{D} = \frac{u : (\Phi \vdash P) \in \Delta' \quad \Delta'; \mathcal{C}'; \Psi \vdash \sigma \Leftarrow \Phi \quad \Delta'; \mathcal{C}'; \overline{\Psi} \vdash [\sigma]_\Phi P = Q}{\Delta'; \mathcal{C}'; \Psi \vdash u[\sigma] \Leftarrow Q}$$

Let $\Psi' = [\![\Theta]\!]\Psi$, $\Phi' = [\![\Theta]\!]\Phi$, $\sigma' = [\![\Theta]\!]\sigma$, $P' = [\![\Theta]\!]P$, $Q' = [\![\Theta]\!]Q$

Let $\Theta(u) = \widehat{\Phi'}.R : (\Phi' \vdash P')$

$\Delta; \mathcal{C}; \Phi' \vdash R \Leftarrow P'$ by inversion on the typing of Θ

$\Delta; \mathcal{C}; \Psi' \vdash \sigma' \Leftarrow \Phi'$ by IH

$\Delta; \mathcal{C}; \overline{\Psi'} \vdash [\sigma']_{\Phi'} P' = Q'$ by IH

Let $R' = [\sigma']_{\Phi'} R$

$\Delta; \mathcal{C}; \Psi' \vdash R' \Leftarrow Q'$ by Lemma 8

$$\textit{Case } \mathcal{D} = \frac{\mathcal{D}' \;\; ; \mathcal{C}'; \Psi_2 \vdash S > A \Rightarrow P \quad \Delta'; \mathcal{C}'; \overline{\Psi} \vdash P = Q \quad \mathcal{C}' \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\Delta; \mathcal{C}; \Psi \vdash c \cdot S \Leftarrow Q}$$

where $\mathcal{D}' = \dfrac{c{:}A \in \Sigma \quad \mathsf{unr}(\Psi_1)}{\Delta; \mathcal{C}; \Psi_1 \vdash c \Rightarrow A}$

Let $\Psi_1' = [\![\Theta]\!]\Psi_1$, $\Phi_1' = [\![\Theta]\!]\Phi_1$, $\Psi_2' = [\![\Theta]\!]\Psi_2$, $\Psi' = [\![\Theta]\!]\Psi$, $S' = [\![\Theta]\!]S$, $P' = [\![\Theta]\!]P$, $Q' = [\![\Theta]\!]Q$

$\mathsf{unr}({\Psi_1}')$ by Lemma 15

$\Delta; \mathcal{C}; {\Psi_1}' \vdash c \Rightarrow A$ by typing rule

$\Delta; \mathcal{C}; {\Psi_2}' \vdash S' > A \Rightarrow P'$ by IH

$\Delta; \mathcal{C}; \overline{\Psi'} \vdash P' = Q'$ by IH

$\mathcal{C} \vdash \Psi' = {\Psi_1}' \bowtie {\Psi_2}'$ by Lemma 14

$\Delta; \mathcal{C}; \Psi' \vdash c \cdot S' \Leftarrow Q'$ by typing rule

$$\textit{Case } \mathcal{D} = \frac{\mathcal{D}' \;\; \Delta'; \mathcal{C}'; \Psi_2 \vdash S{>}[\sigma]_\Phi A \Rightarrow P \;\; \Delta'; \mathcal{C}'; \overline{\Psi} \vdash P = Q \;\; \mathcal{C}' \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\Delta'; \mathcal{C}'; \Psi \vdash p[\sigma] \cdot S \Leftarrow Q}$$

where $\mathcal{D}' = \dfrac{p : (\Phi_1 \vdash \#A) \in \Delta' \quad \Delta'; \mathcal{C}'; \Psi_1 \vdash \sigma \Leftarrow \Phi_1}{\Delta'; \mathcal{C}'; \Psi_1 \vdash p[\sigma] \Rightarrow [\sigma]_\Phi A}$

Let $\Psi_1' = [\![\Theta]\!]\Psi_1$, $\Psi_2' = [\![\Theta]\!]\Psi_2$, $\Psi' = [\![\Theta]\!]\Psi$, $S' = [\![\Theta]\!]S$, $P' = [\![\Theta]\!]P$, $Q' = [\![\Theta]\!]Q$, $\sigma' = [\![\Theta]\!]\sigma$, $A' = [\![\Theta]\!]A$

There are two subcases to consider, depending on the value of $\Theta(p)$.

Subcase $\Theta(p) = \widehat{\Phi_1}'.x : (\Phi_1 \vdash \#A)$

$\Delta; \mathcal{C}; \Psi_1' \vdash \sigma' \Leftarrow \Phi_1'$ by IH

Let $\sigma'(x) = M : \alpha$

Either $\mathsf{unr}(\Phi_1')$ or $\Phi_1' = \Phi_{11}', x\hat{:}A', \Phi_{12}'$ and $\mathsf{unr}(\Phi_{1i}')$

by inversion on typing of $\widehat{\Phi_1}'.x$

$\Delta; \mathcal{C}; \Psi_1' \vdash M \Leftarrow [\sigma']A'$ by Lemma 4

$\Delta; \mathcal{C}; \Psi_2' \vdash S' > [\sigma']A' \Rightarrow P'$ by IH

$\Delta; \mathcal{C}; \overline{\Psi}' \vdash P' = Q'$ by IH

$\mathcal{C} \vdash \Psi' = \Psi_1' \bowtie \Psi_2'$ by Lemma 14

Let $\mathsf{reduce}(M, S') = M' : A'^{-}$

$\Delta; \mathcal{C}; \Psi' \vdash M' \Leftarrow Q'$ by Lemma 7

Subcase $\Theta(p) = \widehat{\Phi_1}'.q[\pi] : (\Phi_1 \vdash \#A)$

$\Delta; \mathcal{C}; \Psi_1' \vdash \sigma' \Leftarrow \Phi_1'$ by IH

$q : (\Phi_0 \vdash \#B) \in \Delta$

$\Delta; \mathcal{C}; \Phi_1' \vdash \pi \Leftarrow \Phi_0$

$\Delta; \mathcal{C}; \overline{\Phi_1}' \vdash A' = [\pi]B$ by inversion on typing of $\widehat{\Phi_1}'.q[\pi]$

Let $[\sigma']\pi = \tau$

$\Delta; \mathcal{C}; \Psi_1' \vdash \tau \Leftarrow \Phi_0$ by Lemma 8

$\Delta; \mathcal{C}; \Psi_1' \vdash q[\tau] \Rightarrow [\tau]B$ by typing rule

$\Delta; \mathcal{C}; \Psi_2' \vdash S' > [\sigma']A' \Rightarrow P'$ by IH

$\Delta; \mathcal{C}; \overline{\Psi_1}' \vdash [\sigma']A' = [\sigma']([\pi]B)$ by Lemma 8

$\Delta; \mathcal{C}; \overline{\Psi_1}' \vdash [\sigma']([\pi]B) = [\tau]B$ by ???

$\mathcal{C} \vdash \Psi' = \Psi_1' \bowtie \Psi_2'$ by Lemma 14

$\Delta; \mathcal{C}; \overline{\Psi}' \vdash P' = Q'$ by IH

$\Delta; \mathcal{C}; \Psi' \vdash q[\tau] \cdot S' > [\sigma']A' \Leftarrow Q'$ by typing rule

Case $\mathcal{D} = \dfrac{\mathcal{D}' \quad \Delta'; \mathcal{C}'; \Psi_2 \vdash S > A \Rightarrow P \quad \Delta'; \mathcal{C}'; \overline{\Psi} \vdash P = Q \quad \mathcal{C}' \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\Delta'; \mathcal{C}'; \Psi \vdash x \cdot S \Leftarrow Q}$

where $\mathcal{D}' = \dfrac{\mathsf{unr}(\Psi_1) \quad x{:}A \in \Psi_1}{\Delta'; \mathcal{C}'; \Psi_1 \vdash x \Rightarrow A}$

Let $\Psi_1' = [\![\Theta]\!]\Psi_1$, $\Psi_2' = [\![\Theta]\!]\Psi_2$, $\Psi' = [\![\Theta]\!]\Psi$, $S' = [\![\Theta]\!]S$, $P' = [\![\Theta]\!]P$, $Q' = [\![\Theta]\!]Q$, $A' = [\![\Theta]\!]A$

$\mathsf{unr}(\Psi_1')$ by Lemma 15

$\Delta; \mathcal{C}; \Psi_1' \vdash x \Rightarrow A'$ by typing rule
$\Delta; \mathcal{C}; \Psi_2{}' \vdash S' > [\sigma']A' \Rightarrow P'$ by IH
$\mathcal{C} \vdash \Psi' = \Psi_1{}' \bowtie \Psi_2{}'$ by Lemma 14
$\Delta; \mathcal{C}; \overline{\Psi}' \vdash P' = Q'$ by IH
$\Delta; \mathcal{C}; \Psi' \vdash x \cdot S' > [\sigma']A' \Leftarrow Q'$ by typing rule

$$\textit{Case } \mathcal{D} = \frac{\mathcal{D}' \quad \Delta; \mathcal{C}; \Psi_2 \vdash S > A \Rightarrow P \quad \Delta; \mathcal{C}; \overline{\Psi} \vdash P = Q \quad \mathcal{C} \vdash \Psi = \Psi_{11}, x\hat{:}A, \Psi_{12} \bowtie \Psi_2}{\Delta; \mathcal{C}; \Psi \vdash x \cdot S \Leftarrow Q}$$

where $\mathcal{D}' = \dfrac{\mathsf{unr}(\Psi_{11}) \quad \mathsf{unr}(\Psi_{12})}{\Delta; \mathcal{C}; \Psi_{11}, x\hat{:}A, \Psi_{12} \vdash x \Rightarrow A}$

Let $\Psi_{11}' = [\![\Theta]\!]\Psi_{11}$, $\Psi_{12}' = [\![\Theta]\!]\Psi_{12}$, $\Psi_2' = [\![\Theta]\!]\Psi_2$, $\Psi' = [\![\Theta]\!]\Psi$, $S' = [\![\Theta]\!]S$, $P' = [\![\Theta]\!]P$, $Q' = [\![\Theta]\!]Q$, $\sigma' = [\![\Theta]\!]\sigma$, $A' = [\![\Theta]\!]A$
$\mathsf{unr}(\Psi_{1i}')$ by Lemma 15
$\Delta; \mathcal{C}; \Psi_{11}', x\hat{:}A', \Psi_{12}' \vdash x \Rightarrow A'$ by typing rule
$\Delta; \mathcal{C}; \Psi_2{}' \vdash S' > [\sigma']A' \Rightarrow P'$ by IH
$\mathcal{C} \vdash \Psi' = (\Psi_{11}{}', x\hat{:}A', \Psi_{12}{}') \bowtie \Psi_2{}'$ by Lemma 14
$\Delta; \mathcal{C}; \overline{\Psi}' \vdash P' = Q'$ by IH
$\Delta; \mathcal{C}; \Psi' \vdash x \cdot S' > [\sigma']A' \Leftarrow Q'$ by typing rule

$$\textit{Case } \mathcal{D} = \frac{}{\Delta'; \mathcal{C}'; \Psi \vdash \epsilon > P \Rightarrow P}$$

Let $[\![\Theta]\!]\Psi = \Psi'$, $[\![\Theta]\!]P = P'$
$\Delta; \mathcal{C}; \Psi' \vdash \epsilon > P' \Rightarrow P'$ by typing rule

$$\textit{Case } \mathcal{D} = \frac{\Delta'; \mathcal{C}'; \overline{\Psi} \vdash M \Leftarrow A \quad \Delta'; \mathcal{C}'; \Psi \vdash S > [M/x]_A B \Rightarrow P}{\Delta'; \mathcal{C}'; \Psi \vdash M\,; S > \Pi x{:}A.B \Rightarrow P}$$

Let $\Psi' = [\![\Theta]\!]\Psi$, $S' = [\![\Theta]\!]S$, $M' = [\![\Theta]\!]M$, $\Phi' = [\![\Theta]\!]\Phi$, $A' = [\![\Theta]\!]A$ $B' = [\![\Theta]\!]B$, $P' = [\![\Theta]\!]P$
$\Delta; \mathcal{C}; \Psi' \vdash S' > [M'/x]B'$ by IH
$\Delta; \mathcal{C}; [\![\Theta]\!]\overline{\Psi} \vdash M' \Leftarrow A'$ by IH
$\Delta; \mathcal{C}; \overline{\Psi'} \vdash M' \Leftarrow A'$ by Lemma 9.1
$\Delta; \mathcal{C}; \Psi' \vdash M'\,; S' > \Pi x{:}A.'B' \Rightarrow P'$ by typing rule

$$\textit{Case } \mathcal{D} = \frac{\Delta'; \mathcal{C}'; \Psi_1 \vdash M \Leftarrow A \quad \Delta'; \mathcal{C}'; \Psi_2 \vdash S > B \Rightarrow P \quad \mathcal{C}' \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\Delta'; \mathcal{C}'; \Psi \vdash M\,\hat{;}\,S > A \multimap B \Rightarrow P}$$

Let $\Psi_1' = [\![\Theta]\!]\Psi_1$, $\Psi_2' = [\![\Theta]\!]\Psi_2$, $\Psi' = [\![\Theta]\!]\Psi$, $S' = [\![\Theta]\!]S$, $M' = [\![\Theta]\!]M$, $\Phi' = [\![\Theta]\!]\Phi$, $A' = [\![\Theta]\!]A$ $B' = [\![\Theta]\!]B$, $P' = [\![\Theta]\!]P$
$\Delta; \mathcal{C}; \Psi_1' \vdash S' > B' \Rightarrow P'$ by IH
$\Delta; \mathcal{C}; \Psi_2' \vdash M' \Leftarrow A'$ by IH
$\mathcal{C} \vdash \Psi' = \Psi_1' \bowtie \Psi_2'$ by Lemma 14
$\Delta; \mathcal{C}; \Psi' \vdash M'\,\hat{;}\,S' > A' \multimap B' \Rightarrow P'$ by typing rule

$$\textit{Case } \mathcal{D} = \frac{\mathsf{unr}(\Psi)}{\Delta'; \mathcal{C}'; \Psi \vdash \cdot \Leftarrow \cdot}$$

Let $\Psi' = [\![\Theta]\!]\Psi$

$\mathsf{unr}(\Psi')$	by Lemma 9.2
$\Delta; \mathcal{C}; \Psi' \vdash \cdot \Leftarrow \cdot$	by typing rule

$$\textit{Case } \mathcal{D} = \frac{\mathsf{unr}(\Psi^0)}{\Delta'; \mathcal{C}'; \psi, \Psi^0 \vdash \mathsf{id}_\psi \Leftarrow \psi}$$

Let $\Psi^{0\prime} = [\![\Theta]\!]\Psi^0$, $\Psi = [\![\Theta]\!]\psi$

$\mathsf{unr}(\Psi^{0\prime})$	by Lemma 9.2
$\Delta; \mathcal{C}; \Psi, \Psi^{0\prime} \vdash \mathsf{id}(\Psi) \Leftarrow \Psi$	by Lemma 10

$$\textit{Case } \mathcal{D} = \frac{\mathsf{unr}(\Psi^0) \quad \mathcal{C}' \vdash \phi \leftrightsquigarrow \psi \quad \mathcal{C}' \vdash \psi \leftrightsquigarrow \gamma}{\Delta'; \mathcal{C}'; \overline{\phi}, \Psi^0 \vdash \mathsf{id}_{\overline{\psi}} \Leftarrow \overline{\gamma}}$$

Let $\Psi = [\![\Theta]\!]\psi$, $\Phi = [\![\Theta]\!]\phi$, $\Gamma = [\![\Theta]\!]\gamma$

$\mathsf{unr}(\Psi^{0\prime})$	by Lemma 9.2
$\Delta; \mathcal{C}; \overline{\Psi}, \Psi^{0\prime} \vdash \mathsf{id}(\overline{\Psi}) \Leftarrow \overline{\Psi}$	by Lemma 10
$\overline{\Psi} = \overline{\Phi} = \overline{\Gamma}$	by Lemma 12
$\Delta; \mathcal{C}; \overline{\Phi}, \Psi^{0\prime} \vdash \mathsf{id}(\overline{\Psi}) \Leftarrow \overline{\Gamma}$	

$$\textit{Case } \mathcal{D} = \frac{\Delta'; \mathcal{C}'; \Psi \vdash \sigma \Leftarrow \Phi \quad \Delta'; \mathcal{C}'; \overline{\Psi} \vdash M \Leftarrow [\sigma]A}{\Delta'; \mathcal{C}'; \Psi \vdash \sigma, M \Leftarrow \Phi, x{:}A}$$

Let $\Psi' = [\![\Theta]\!]\Psi$, $\sigma' = [\![\Theta]\!]\sigma$, $M' = [\![\Theta]\!]M$, $\Phi' = [\![\Theta]\!]\Phi$, $A' = [\![\Theta]\!]A$

$\Delta; \mathcal{C}; \Psi' \vdash \sigma' \Leftarrow \Phi'$	by IH
$\Delta; \mathcal{C}; [\![\Theta]\!]\overline{\Psi} \vdash M' \Leftarrow [\sigma']A'$	by IH
$\Delta; \mathcal{C}; \overline{\Psi'} \vdash M' \Leftarrow [\sigma']A'$	by Lemma 9.1
$\Delta; \mathcal{C}; \Psi' \vdash \sigma', M' \Leftarrow \Phi', x{:}A'$	by typing rule

$$\textit{Case } \mathcal{D} = \frac{\Delta'; \mathcal{C}'; \Psi_1 \vdash \sigma \Leftarrow \Phi \quad \Delta'; \mathcal{C}'; \Psi_2 \vdash M \Leftarrow [\sigma]A \quad \mathcal{C}' \vdash \Psi = \Psi_1 \bowtie \Psi_2}{\Delta'; \mathcal{C}'; \Psi \vdash \sigma, M \Leftarrow \Phi, x\hat{:}A}$$

Let $\Psi_1' = [\![\Theta]\!]\Psi_1$, $\Psi_2' = [\![\Theta]\!]\Psi_2$, $\sigma' = [\![\Theta]\!]\sigma$, $M' = [\![\Theta]\!]M$, $\Phi' = [\![\Theta]\!]\Phi$, $A' = [\![\Theta]\!]A$

$\Delta; \mathcal{C}; \Psi_1' \vdash \sigma' \Leftarrow \Phi'$	by IH
$\Delta; \mathcal{C}; \Psi_2' \vdash M' \Leftarrow [\sigma']A'$	by IH
$\mathcal{C} \vdash \Psi' = \Psi_1' \bowtie \Psi_2'$	by Lemma 14
$\Delta; \mathcal{C}; \Psi' \vdash \sigma', M' \Leftarrow \Phi', x\hat{:}A'$	by typing rule

4 Related Work

- McCreight and Schürmann [2004] propose to write meta-theoretic proofs about linear systems within the linear logical framework itself. This is mired with difficulties, as such meta-theoretic proofs are in fact not linear.
- Bock and Schürmann [2015] present a contextual logical framework for linear logic. To avoid problems with context splitting in the presence of context variables, they explicitly mention the set of names that a given context variable should be disjoint from. This makes the resulting presentation almost nominal in nature.

5 Conclusion and Future Work

Bibliography

P. B. Bock and C. Schürmann. A contextual logical framework. In *Proceedings of he 20th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'15)*, Lecture Notes in Computer Science, page to appear. Springer, 2015.

L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In P. Gastin and F. Laroussinie, editors, *21th International Conference on Concurrency Theory CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2010. doi: 10.1007/978-3-642-15375-4_16.

A. Cave and B. Pientka. Programming with binders and indexed data-types. In *39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'12)*, pages 413–424. ACM Press, 2012.

A. Cave and B. Pientka. First-class substitutions in contextual type theory. In *Proceedings of the Eighth ACM SIGPLAN International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP'13)*, pages 15–24. ACM Press, 2013. ISBN 978-1-4503-2382-6.

I. Cervesato and F. Pfenning. A linear logical framework. In E. Clarke, editor, *11th Annual Symposium on Logic in Computer Science (LICS'96)*, pages 264–275. IEEE Computer Society Press, 1996.

I. Cervesato and F. Pfenning. A linear logical framework. *Information & Computation*, 179(1):19–75, Nov. 2002.

I. Cervesato and F. Pfenning. A linear spine calculus. *Journal of Logic and Computation*, 13(5):639–688, 2003.

R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.

A. McCreight and C. Schürmann. A meta-linear logical framework. In *4th International Workshop on Logical Frameworks and Meta-Languages (LFM'04)*, 2004.

A. Nanevski, F. Pfenning, and B. Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008.

B. Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'08)*, pages 371–382. ACM Press, 2008.

B. Pientka and A. Cave. Inductive beluga:programming proofs (system description). In A. P. Felty and A. Middeldorp, editors, *25th International Conference on Automated Deduction (CADE-25)*, volume 9195 of *Lecture Notes in Computer Science*, pages 272–281. Springer, 2015.

A. Schack-Nielson. *Implementing Substructural Logical Frameworks*. PhD thesis, IT University Copenhagen, Denmark, 2011.

A. Schack-Nielson and C. Schürmann. Pattern unification for the lambda calculus with linear and affine types. In K. Crary and M. Miculan, editors, *International Workshop on Logical Frameworks and Meta-Languages: Theory*

and Practice (LFMTP'10), volume 34 of *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, pages 101–116, July 2010.

P. Wadler. Linear types can change the world! In M. Broy and C. Jones, editors, *Programming Concepts and Methods*. North, 1990.

P. Wadler. Propositions as sessions. In *17th ACM SIGPLAN International Conference on Functional Programming (ICFP'12)*, pages 273–286. ACM, 2012. ISBN 978-1-4503-1054-3. doi: 10.1145/2364527.2364568.

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

A Metatheory

$\boxed{\Delta;\mathcal{C};\Psi \vdash A\ \mathsf{type}}$ A is a valid type in an unrestricted context Ψ

$$\frac{a{:}K \in \Sigma \quad \Delta;\mathcal{C};\Psi \vdash S > K \Rightarrow \mathsf{type}}{\Delta;\mathcal{C};\Psi \vdash a \cdot S\ \mathsf{type}} \qquad \frac{\Delta;\mathcal{C};\Psi \vdash A\ \mathsf{type} \quad \Delta;\mathcal{C};\Psi \vdash B\ \mathsf{type}}{\Delta;\mathcal{C};\Psi \vdash A \multimap B\ \mathsf{type}}$$

$$\frac{\Delta;\mathcal{C};\Psi \vdash A\ \mathsf{type} \quad \Delta;\mathcal{C};\Psi, x{:}A \vdash B\ \mathsf{type}}{\Delta;\mathcal{C};\Psi \vdash \Pi A{:}B.\ \mathsf{type}}$$

$\boxed{\Delta;\mathcal{C};\Psi \vdash S > K \Rightarrow \mathsf{type}}$ TODO ...in an unrestricted context Ψ

$$\frac{}{\Delta;\mathcal{C};\Psi \vdash \epsilon > \mathsf{type} \Rightarrow \mathsf{type}} \qquad \frac{\Delta;\mathcal{C};\Psi \vdash M \Leftarrow A \quad \Delta;\mathcal{C};\Psi \vdash S > [M/x]_A K \Rightarrow \mathsf{type}}{\Delta;\mathcal{C};\Psi \vdash M\,;S > \Pi x{:}A.K \Rightarrow \mathsf{type}}$$

$\boxed{\Delta;\mathcal{C};\Psi \vdash K\ \mathsf{kind}}$ K is a valid kind in an unrestricted context Ψ

$$\frac{}{\Delta;\mathcal{C};\Psi \vdash \mathsf{type\ kind}} \qquad \frac{\Delta;\mathcal{C};\Psi, x{:}A \vdash K\ \mathsf{kind}}{\Delta;\mathcal{C};\Psi \vdash \Pi x{:}A.K\ \mathsf{kind}}$$

Fig. 10. Rules for kinds and types

Remark 1 (Unrestricted join). If $\Delta;\mathcal{C} \vdash \Phi = \Phi_1 \bowtie \Phi_2$ and $\mathsf{unr}(\Psi^0)$ then $\Delta;\mathcal{C} \vdash (\Phi, \Psi^0) = (\Phi_1, \Psi^0) \bowtie (\Phi_2, \Psi^0)$.

Remark 2. If $\Delta;\mathcal{C} \vdash \Phi = \Phi_1 \bowtie \Phi_2$ then $\overline{\Phi} =_{\mathcal{C}} \overline{\Phi_1} =_{\mathcal{C}} \overline{\Phi_2}$.
The context variables occuring in Φ, Φ_1, Φ_2 may be different, but their unrestricted parts will be equal under any substitution. I.e. $\overline{\psi} = \overline{\psi_1} = \overline{\psi_2}$ for $\psi := \psi_1 \bowtie \psi_2 \in \mathcal{C}$.

Remark 3. If $\Delta;\mathcal{C} \vdash \Phi = \Phi_1 \bowtie \Phi_2$, $\Delta;\mathcal{C} \vdash \Psi = \Psi_1 \bowtie \Psi_2$ $\Delta;\mathcal{C};\Psi \vdash \sigma \Leftarrow \Phi$, and $\sigma = \sigma_1 \bowtie \sigma_2$ then for any type A, $[\sigma]_\Phi A = [\sigma_i]_{\Phi_i} A$.

$\boxed{\Delta; \mathcal{C} \vdash \Psi \ \mathsf{ctx}}$ Ψ is a valid context

$$\frac{}{\Delta; \mathcal{C} \vdash \cdot \ \mathsf{ctx}} \qquad \frac{\psi : G \in \Delta}{\Delta; \mathcal{C} \vdash \psi \ \mathsf{ctx}} \qquad \frac{\psi : G \in \Delta}{\Delta; \mathcal{C} \vdash \overline{\psi} \ \mathsf{ctx}}$$

$$\frac{\phi : G \in \Delta \quad \mathcal{C} \vdash \psi \rightsquigarrow^* \phi}{\Delta; \mathcal{C} \vdash \psi \ \mathsf{ctx}} \qquad \frac{\phi : G \in \Delta \quad \mathcal{C} \vdash \psi \rightsquigarrow^* \phi}{\Delta; \mathcal{C} \vdash \overline{\psi} \ \mathsf{ctx}}$$

$$\frac{\Delta; \mathcal{C} \vdash \Psi \ \mathsf{ctx} \quad \Delta; \mathcal{C}; \overline{\Psi} \vdash A \ \mathsf{type}}{\Delta; \mathcal{C} \vdash \Psi, x{:}A \ \mathsf{ctx}} \qquad \frac{\Delta; \mathcal{C} \vdash \Psi \ \mathsf{ctx} \quad \Delta; \mathcal{C}; \overline{\Psi} \vdash A \ \mathsf{type}}{\Delta; \mathcal{C} \vdash \Psi, x\hat{:}A \ \mathsf{ctx}}$$

Fig. 11. Rules for well-formed contexts

$\boxed{[M/x]_\alpha A}$ Single substitution for types

$$\begin{array}{lll}
[M/x]_\alpha(a \cdot S) & = a \cdot S' & \text{where } [M/x]^l_\alpha S = S' \\
[M/x]_\alpha(\Pi y{:}A.B) & = \Pi y{:}A'.B' & \text{where } [M/x]_\alpha A = A' \\
 & & \text{and } [M/x]_\alpha B = B' \\
[M/x]_\alpha(A \multimap B) & = A' \multimap B' & \text{where } [M/x]_\alpha A = A' \\
 & & \text{and } [M/x]_\alpha B = B'
\end{array}$$

$\boxed{[\sigma]^{\widehat{\Phi}}_{\Psi} A}$ Simultaneous substitution for types

$$\begin{array}{lll}
[\sigma]^{\widehat{\Phi}}_{\Psi}(a \cdot S) & = a \cdot S' & \text{where } [\sigma]^{\widehat{\Phi}}_{\Psi} S = S' \\
[\sigma]^{\widehat{\Phi}}_{\Psi}(\Pi x{:}A.B) & = \Pi x{:}A'.B' & \text{where } [\sigma]^{\widehat{\Phi}}_{\Psi} A = A' \\
 & & \text{and } [\sigma]^{\widehat{\Phi},x}_{\Psi} B = B' \\
[\sigma]^{\widehat{\Phi}}_{\Psi}(A \multimap B) & = A' \multimap B' & \text{where } [\sigma]^{\widehat{\Phi}}_{\Psi} A = A' \\
 & & \text{and } [\sigma]^{\widehat{\Phi}}_{\Psi} B = B'
\end{array}$$

$\boxed{[\sigma]^{\widehat{\Phi}}_{\Psi} \Psi'}$ Simultaneous substitution for contexts

$$\begin{array}{lll}
[\sigma]^{\widehat{\Phi}}_{\Psi}(\cdot) & = \cdot & \\
[\sigma]^{\widehat{\Phi}}_{\Psi}(\psi) & = \psi & \\
[\sigma]^{\widehat{\Phi}}_{\Psi}(\overline{\psi}) & = \overline{\psi} & \\
[\sigma]^{\widehat{\Phi}}_{\Psi}(\Psi', x{:}A) & = \Psi'', x{:}A' & \text{where } [\sigma]^{\widehat{\Phi}}_{\Psi} \Psi' = \Psi'' \\
 & & \text{and } [\sigma]^{\widehat{\Phi},\widehat{\Psi'}}_{\Psi} A = A' \\
[\sigma]^{\widehat{\Phi}}_{\Psi}(\Psi', x\hat{:}A) & = \Psi'', x\hat{:}A' & \text{where } [\sigma]^{\widehat{\Phi}}_{\Psi} \Psi' = \Psi'' \\
 & & \text{and } [\sigma]^{\widehat{\Phi},\widehat{\Psi'}}_{\Psi} A = A'
\end{array}$$

Fig. 12. Substitution for types and contexts

$\boxed{[\![C/X]\!]_U S}$ Single meta-substitution for spines

$$
\begin{array}{lll}
[\![C/X]\!]_U(\epsilon) & = \epsilon & \\
[\![C/X]\!]_U(N\,;S) & = N'\,;S' & \text{where } [\![C/X]\!]_U\ N = N' \\
 & & \text{and } [\![C/X]\!]_U\ S = S' \\
[\![C/X]\!]_U(N\,\hat{;}\,S) & = N'\,\hat{;}\,S' & \text{where } [\![C/X]\!]_U\ N = N' \\
 & & \text{and } [\![C/X]\!]_U\ S = S'
\end{array}
$$

$\boxed{[\![\Theta]\!]_\Delta S}$ Simultaneous meta-substitution for spines

$$
\begin{array}{lll}
[\![\Theta]\!]_\Delta(\epsilon) & = \epsilon & \\
[\![\Theta]\!]_\Delta(N\,;S) & = N'\,;S' & \text{where } [\![\Theta]\!]_\Delta\ N = N' \\
 & & \text{and } [\![\Theta]\!]_\Delta\ S = S' \\
[\![\Theta]\!]_\Delta(N\,\hat{;}\,S) & = N'\,\hat{;}\,S' & \text{where } [\![\Theta]\!]_\Delta\ N = N' \\
 & & \text{and } [\![\Theta]\!]_\Delta\ S = S'
\end{array}
$$

Fig. 13. Meta-substitution for spines

$\boxed{[\![C/X]\!]_U A}$ Single meta-substitution for types

$$
\begin{array}{lll}
[\![C/X]\!]_U(a \cdot S) & = a \cdot S' & \text{where } [\![C/X]\!]_U\ S = S' \\
[\![C/X]\!]_U(\Pi y{:}A.B) & = \Pi y{:}A'.B' & \text{where } [\![C/X]\!]_U\ A = A' \\
 & & \text{and } [\![C/X]\!]_U\ B = B' \\
[\![C/X]\!]_U(A \multimap B) & = A' \multimap B' & \text{where } [\![C/X]\!]_U\ A = A' \\
 & & \text{and } [\![C/X]\!]_U\ B = B'
\end{array}
$$

$\boxed{[\![\Theta]\!]_\Delta A}$ Simultaneous meta-substitution for types

$$
\begin{array}{lll}
[\![\Theta]\!]_\Delta(a \cdot S) & = a \cdot S' & \text{where } [\![\Theta]\!]_\Delta\ S = S' \\
[\![\Theta]\!]_\Delta(\Pi y{:}A.B) & = \Pi y{:}A'.B' & \text{where } [\![\Theta]\!]_\Delta\ A = A' \\
 & & \text{and } [\![\Theta]\!]_\Delta\ B = B' \\
[\![\Theta]\!]_\Delta(A \multimap B) & = A' \multimap B' & \text{where } [\![\Theta]\!]_\Delta\ A = A' \\
 & & \text{and } [\![\Theta]\!]_\Delta\ B = B'
\end{array}
$$

Fig. 14. Meta-substitution for types