Static and user-extensible proof checking

Extended Version

Antonis Stampoulis Zhong Shao

Department of Computer Science
Yale University
New Haven, CT 06520, USA

{antonis.stampoulis,zhong.shao}@yale.edu

Abstract

Despite recent successes, large-scale proof development within proof assistants remains an arcane art that is extremely time-consuming. We argue that this can be attributed to two profound shortcomings in the architecture of modern proof assistants. The first is that proofs need to include a large amount of minute detail; this is due to the rigidity of the proof checking process, which cannot be extended with domain-specific knowledge. In order to avoid these details, we rely on developing and using tactics, specialized procedures that produce proofs. Unfortunately, tactics are both hard to write and hard to use, revealing the second shortcoming of modern proof assistants. This is because there is no static knowledge about their expected use and behavior.

As has recently been demonstrated, languages that allow type-safe manipulation of proofs, like Beluga, Delphin and VeriML, can be used to partly mitigate this second issue, by assigning rich types to tactics. Still, the architectural issues remain. In this paper, we build on this existing work, and demonstrate two novel ideas: an *extensible conversion rule* and support for *static proof scripts*. Together, these ideas enable us to support both user-extensible proof checking, and sophisticated static checking of tactics, leading to a new point in the design space of future proof assistants. Both ideas are based on the interplay between a light-weight staging construct and the rich type information available.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory

General Terms Languages, Verification

1. Introduction

There have been various recent successes in using proof assistants to construct foundational proofs of large software, like a C compiler [Leroy 2009] and an OS microkernel [Klein et al. 2009], as well as complicated mathematical proofs [Gonthier 2008]. Despite this success, the process of large-scale proof development using the foundational approach remains a complicated endeavor that requires significant manual effort and is plagued by various architectural issues.

The big benefit of using a foundational proof assistant is that the proofs involved can be checked for validity using a very small proof checking procedure. The downside is that these proofs are very large, since proof checking is fixed. There is no way to add domain-specific knowledge to the proof checker, which would enable proofs that spell out less details. There is good reason for this, too: if we allowed arbitrary extensions of the proof checker, we could very easily permit it to accept invalid proofs.

Because of this lack of extensibility in the proof checker, users rely on tactics: procedures that produce proofs. Users are free to write their own tactics, that can create domain-specific proofs. In fact, developing domain-

specific tactics is considered to be good engineering when doing large developments, leading to significantly decreased overall effort – as shown, e.g. in Chlipala [2011]. Still, using and developing tactics is error-prone. Tactics are essentially untyped functions that manipulate logical terms, and thus tactic programming is untyped. This means that common errors, like passing the wrong argument, or expecting the wrong result, are not caught statically. Exacerbating this, proofs contained within tactics are not checked statically, when the tactic is defined. Therefore, even if the tactic is used correctly, it could contain serious bugs that manifest only under some conditions.

With the recent advent of programming languages that support strongly typed manipulation of logical terms, such as Beluga [Pientka and Dunfield 2008], Delphin [Poswolsky and Schürmann 2008] and VeriML [Stampoulis and Shao 2010], this situation can be somewhat mitigated. It has been shown in Stampoulis and Shao [2010] that we can specify what kinds of arguments a tactic expects and what kind of proof it produces, leading to a type-safe programming style. Still, this does not address the fundamental problem of proof checking being fixed – users still have to rely on using tactics. Furthermore, the proofs contained within the type-safe tactics are in fact proof-producing programs, which need to be evaluated upon invocation of the tactic. Therefore proofs within tactics are not checked statically, and they can still cause the tactics to fail upon invocation.

In this paper, we build on the past work on these languages, aiming to solve both of these issues regarding the architecture of modern proof assistants. We introduce two novel ideas: support for an **extensible conversion rule** and **static proof scripts** inside tactics. The former technique enables proof checking to become user-extensible, while maintaining the guarantee that only logically sound proofs are admitted. The latter technique allows for statically checking the proofs contained within tactics, leading to increased guarantees about their runtime behavior. Both techniques are based on the same mechanism, which consists of a light-weight staging construct. There is also a deep synergy between them, allowing us to use the one to the benefit of the other.

Our main contributions are the following:

- First, we present what we believe is the first technique for having an extensible conversion rule, which combines the following characteristics: it is safe, meaning that it preserves logical soundness; it is user-extensible, using a familiar, generic programming model; and, it does not require metatheoretic additions to the logic, but can be used to simplify the logic instead.
- Second, building on existing work for typed tactic development, we introduce static checking of the proof scripts contained within tactics. This significantly reduces the development effort required, allowing us to write tactics that benefit from existing tactics and from the rich type information available.
- Third, we show how typed proof scripts can be seen as an alternative form of proof witness, which falls between a proof object and a proof script. Receivers of the certificate are able to decide on the tradeoff between the level of trust they show and the amount of resources needed to check its validity.

In terms of technical contributions, we present a number of technical advances in the metatheory of the aforementioned programming languages. These include a simple staging construct that is crucial to our development and a new technique for variable representation. We also show a condition under which static checking of proof scripts inside tactics is possible. Last, we have extended an existing prototype implementation with a significant number of features, enabling it to support our claims, while also rendering its use as a proof assistant more practical.

2. Informal presentation

Glossary of terms. We will start off by introducing some concepts that will be used throughout the paper. The first fundamental concept we will consider is the notion of a *proof object*: given a derivation of a proposition inside a formal logic, a proof object is a term representation of this derivation. A *proof checker* is a program that can decide whether a given proof object is a valid derivation of a specific proposition or not. Proof objects are extremely verbose and are thus hard to write by hand. For this reason, we use *tactics*: functions that produce

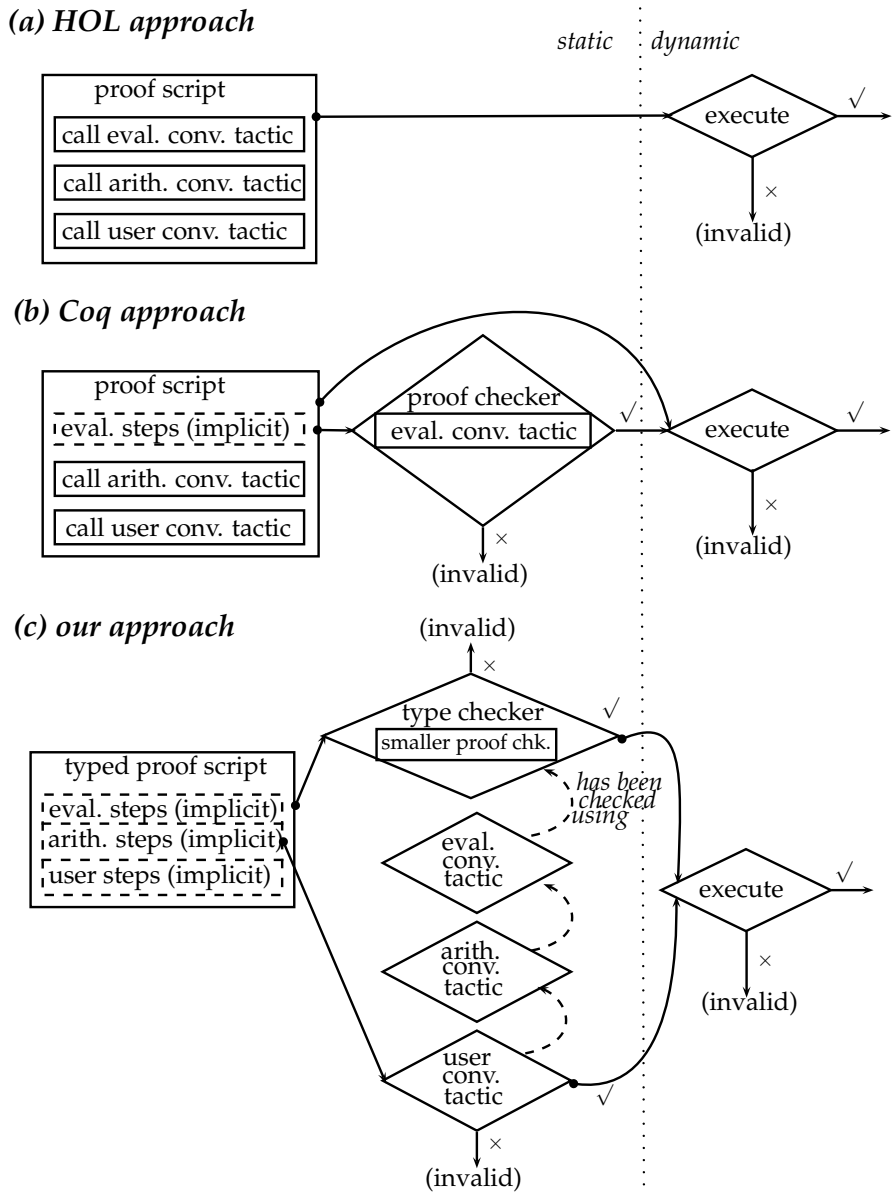

Figure 1. Checking proof scripts in various proof assistants

proof objects. By combining tactics together, we create proof-producing programs, which we call *proof scripts*. If a proof script is evaluated, and the evaluation completes successfully, the resulting proof object can be checked using the original proof checker. In this way, the trusted base of the system is kept at the absolute minimum. The language environment where proof scripts and tactics are written and evaluated is called a *proof assistant*; evidently, it needs to include a proof checker.

Checking proof objects. In order to keep the size of proof objects manageable, many of the logics used for mechanized proof checking include a *conversion rule*. This rule is used implicitly by the proof checker to decide whether any two propositions are equivalent; if it determines that they are indeed so, the proof of their equivalence can be omitted. We can thus think of it as a special tactic that is embedded within the proof checker, and used implicitly.

The more sophisticated the relation supported by the conversion rule is, the simpler are proof objects to write, since more details can be omitted. On the other hand, the proof checker becomes more complicated, as does the metatheory proof showing the soundness of the associated logic. The choice in Coq [Barras et al. 2010], one of the most widely used proof assistants, with respect to this trade-off, is to have a conversion rule that identifies propositions up to evaluation. Nevertheless, extended notions of conversion are desirable, leading to proposals like CoqMT [Strub 2010], where equivalence up to first-order theories is supported. In both cases, the conversion rule is fixed, and extending it requires significant amounts of work. It is thus not possible for users to extend it using their own, domain-specific tactics, and proof objects are thus bound to get large. This is why we have to resort to writing proof scripts.

Checking proof scripts. As mentioned earlier, in order to validate a proof script we need to evaluate it (see Fig. 1a); this is the modus operandi in proof assistants of the HOL family [Harrison 1996; Slind and Norrish 2008]. Therefore, it is easy to extend the checking procedure for proof scripts by writing a new tactic, and calling it as part of a script. The price that this comes to is that there is no way to have any sort of static guarantee about the validity of the script, as proof scripts are completely untyped. This can be somewhat mitigated in Coq by utilizing the static checking that it already supports: the proof checker, and especially, the conversion rule it contains (see Fig. 1b). We can employ proof objects in our scripts; this is especially useful when the proof objects are trivial to write but trigger complex conversion checks. This is the essential idea behind techniques like proof-by-reflection [Boutin 1997], which lead to more robust proof scripts.

In previous work [Stampoulis and Shao 2010] we introduced VeriML, a language that enables programming tactics and proof scripts in a typeful manner using a general-purpose, side-effectful programming model. Combining typed tactics leads to *typed proof scripts*. These are still programs producing proof objects, but the proposition they prove is carried within their type. Information about the current proof state (the set of hypotheses and goals) is also available statically at every intermediate point of the proof script. In this way, the static assurances about proof scripts are significantly increased and many potential sources of type errors are removed. On the other hand, the proof objects contained within the scripts are still checked using a fixed proof checker; this ultimately means that the set of possible static guarantees is still fixed.

Extensible conversion rule. In this paper, we build on our earlier work on VeriML. In order to further increase the amount of static checking of proof scripts that is possible within this language, we propose the notion of an extensible conversion rule (see Fig. 1c). It enables users to write their own domain-specific conversion checks that get included in the conversion rule. This leads to simpler proof scripts, as more parts of the proof can be inferred by the conversion rule and can therefore be omitted. Also, it leads to increased static guarantees for proof scripts, since the conversion checks happen before the rest of the proof script is evaluated.

The way we achieve this is by programming the conversion checks as type-safe tactics within VeriML, and then evaluating them statically using a simple staging mechanism (see Fig. 2). The type of the conversion tactics requires that they produce a proof object which proves the claimed equivalence of the propositions. In this way, type safety of VeriML guarantees that soundness is maintained. At the same time, users are free to extend the conversion rule with their own conversion tactics written in a familiar programming model, without requiring any metatheoretic additions or termination proofs. Such proofs are only necessary if decidability of the extra conversion checks is desired. Furthermore, this approach allows for metatheoretic reductions as the original conversion rule can be programmed within the language. Thus it can be removed from the logic, and replaced by the simpler notion of explicit equalities, leading to both simpler metatheory and a smaller trusted base.

Checking tactics. The above approach addresses the issue of being able to extend the amount of static checking possible for proof scripts. But what about tactics? Our existing work on VeriML shows how the increased type information addresses some of the issues of tactic development using current proof assistants, where tactics are programmed in a completely untyped manner.

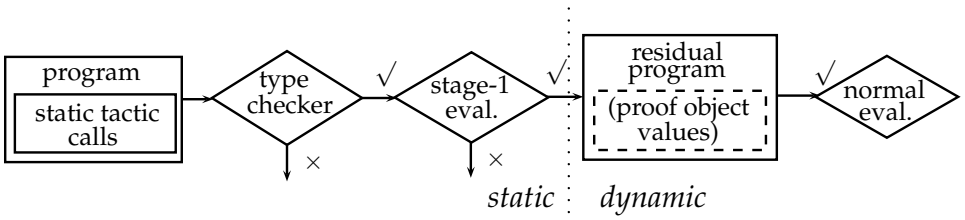

Figure 2. Staging in VeriML

Still, if we consider the case of tactics more closely, we will see that there is a limitation to the amount of checking that is done statically, even using this language. When programming a new tactic, we would like to reuse existing tactics to produce the required proofs. Therefore, rather than writing proof objects by hand inside the code of a tactic, we would rather use proof scripts. The issue is that in order to check whether the contained proof scripts are valid, they need to be evaluated – but this only happens when an invocation of the tactic reaches the point where the proof script is used. Therefore, the static guarantees that this approach provides are severely limited by the fact that the proof scripts inside the tactics cannot be checked statically, when the tactic is defined.

Static proof scripts. This is the second fundamental issue we address in this paper. We show that the same staging construct utilized for introducing the extensible conversion rule, can be leveraged to perform *static proof checking for tactics*. The crucial point of our approach is the proof of existence of a transformation between proof objects, which suggests that under reasonable conditions, a proof script contained within a tactic can be transformed into a static proof script. This static script can then be evaluated at tactic definition time, to be checked for validity.

Last, we will show that this approach lends itself well to writing extensions of the conversion rule. We show that we can create a layering of conversion rules: using a basic conversion rule as a starting point, we can utilize it inside static proof scripts to implicitly prove the required obligations of a more advanced version, and so on. This minimizes the required user effort for writing new conversion rules, and enables truly modular proof checking.

3. Our toolbox

In this section, we will present the essential ingredients that are needed for the rest of our development. The main requirement is a language that supports type-safe manipulation of terms of a particular logic, as well as a general-purpose programming model that includes general recursion and other side-effectful operations. Two recently proposed languages for manipulating LF terms, Beluga [Pientka and Dunfield 2008] and Delphin [Poswolsky and Schürmann 2008], fit this requirement, as does VeriML [Stampoulis and Shao 2010], which is a language used to write type-safe tactics. Our discussion is focused on the latter, as it supports a richer ML-style calculus compared to the others, something useful for our purposes. Still, our results apply to all three.

We will now briefly describe the constructs that these languages support, as well as some new extensions that we propose. The interested reader can read more about these constructs in Sec. 6 and in the appendix.

A formal logic. The computational language we are presenting is centered around manipulation of terms of a specific formal logic. We will see more details about this logic in Sec. 4. For the time being, it will suffice to present a set of assumptions about the syntactic classes and typing judgements of this logic, shown in Fig. 3. Logical terms are represented by the syntactic class t, and include proof objects, propositions, terms corresponding to the domain of discourse (e.g. natural numbers), and the needed sorts and type constructors to classify such terms. Their variables are assigned types through an ordered context Φ. A package of a logical term t together with the variables context it inhabits Φ is called a contextual term and denoted as $T = [\Phi]\, t$. Our computational language works over contextual terms for reasons that will be evident later. The logic incorporates

$t ::= \textit{proof object constructors} \mid \textit{propositions} \mid \textit{natural numbers, lists, etc.} \mid \textit{sorts and types} \mid X/\sigma$
$\Phi ::= \bullet \mid \Phi,\ x : t \qquad\qquad T ::= [\Phi]t$
$\Psi ::= \bullet \mid \Psi,\ X : T \qquad\qquad \sigma ::= \bullet \mid \sigma,\ x \mapsto t$
main judgement: $\Psi;\ \Phi \vdash t : t'$ *(type of a logical term)*

Figure 3. Assumptions about the logic language

$k ::= * \mid k_1 \to k_2$
$\tau ::= \mathsf{unit} \mid \mathsf{int} \mid \mathsf{bool} \mid \tau_1 \to \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \times \tau_2 \mid \mu\alpha : k.\tau \mid \forall\alpha : k.\tau \mid \alpha \mid \mathsf{array}\ \tau \mid \lambda\alpha : k.\tau \mid \tau_1\ \tau_2 \mid \cdots$
$e ::= () \mid n \mid e_1 + e_2 \mid e_1 \le e_2 \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else} e_2 \mid \lambda\mathbf{x} : \tau.e \mid e_1\ e_2 \mid (e_1,\ e_2) \mid \mathsf{proj}_i\ e \mid \mathsf{inj}_i\ e$
$\quad \mid \mathsf{case}(e, \mathbf{x}_1.e_1, \mathbf{x}_2.e_2) \mid \mathsf{fold}\ e \mid \mathsf{unfold}\ e \mid \Lambda\alpha : k.e \mid e\ \tau \mid \mathsf{fix}\ \mathbf{x} : \tau.e \mid \mathsf{mkarray}(e, e') \mid e[e'] \mid e[e'] := e''$
$\quad \mid l \mid \mathsf{error} \mid \cdots$
$\Gamma ::= \bullet \mid \Gamma,\ \mathbf{x} : \tau \mid \Gamma,\ \alpha : k \qquad\qquad \Sigma ::= \bullet \mid \Sigma,\ l : \mathsf{array}\ \tau$

Figure 4. Syntax for the computational language (ML fragment)

$\tau ::= \cdots \mid (X : T) \to \tau \mid (X : T) \times \tau \mid (\phi : \mathsf{ctx}) \to \tau$
$e ::= \cdots \mid \lambda X : T.e \mid e\ T \mid \lambda\phi : \mathsf{ctx}.e \mid e\ \Phi \mid \langle T,\ e\rangle \mid \mathsf{let}\ \langle X,\ \mathbf{x}\rangle = e\ \mathsf{in}\ e'$
$\quad \mid \mathsf{holcase}\ T\ \mathsf{return}\ \tau\ \mathsf{of}\ (T_1 \mapsto e_1) \cdots (T_n \mapsto e_n) \mid \mathsf{ctxcase}\ \Phi\ \mathsf{return}\ \tau\ \mathsf{of}\ (\Phi_1 \mapsto e_1) \cdots (\Phi_n \mapsto e_n)$

Figure 5. Syntax for the computational language (logical term constructs)

such terms by allowing them to get substituted for *meta-variables* X, using the constructor X/σ. When a term $T = [\Phi']t$ gets substituted for X, we go from the Φ' context to the current context Φ using the substitution σ.

Logical terms are classified using other logical terms, based on the normal variables environment Φ, and also an environment Ψ that types meta-variables, thus leading to the $\Psi;\ \Phi \vdash t : t'$ judgement. For example, a term t representing a closed proposition will be typed as $\bullet;\ \bullet \vdash t : \mathsf{Prop}$, while a proof object t_{pf} proving that proposition will satisfy the judgement $\bullet;\ \bullet \vdash t_{\text{pf}} : t$.

ML-style functional programming. We move on to the computational language. As its main core, we assume an ML-style functional language, supporting general recursion, algebraic data types and mutable references (see Fig. 4). Terms of this fragment are typed under a computational variables environment Γ and a store typing environment Σ, mapping mutable locations to types. Typing judgements are entirely standard, leading to a $\Sigma;\ \Gamma \vdash e : \tau$ judgement for typing expressions.

Dependently-typed programming over logical terms. As shown in Fig. 5, the first important additions to the ML computational core are constructs for dependent functions and products over contextual terms T. Abstraction over contextual terms is denoted as $\lambda X : T.e$. It has the dependent function type $(X : T) \to \tau$. The type is dependent since the introduced logical term might be used as the type of another term. An example would be a function that receives a proposition plus a proof object for that proposition, with type: $(P : \mathsf{Prop}) \to (X : P) \to \tau$. Dependent products that package a contextual logical term with an expression are introduced through the $\langle T,\ e\rangle$ construct and eliminated using $\mathsf{let}\ \langle X,\ \mathbf{x}\rangle = e\ \mathsf{in}\ e'$; their type is denoted as $(X : T) \times \tau$. Especially for packages of proof objects with the unit type, we introduce the syntax $\mathsf{LT}(T)$.

Last, in order to be able to support functions that work over terms in any context, we introduce context polymorphism, through a similarly dependent function type over contexts. With these in mind, we can define a simple tactic that gets a packaged proof of a universally quantified formula, and an instantiation term, and returns a proof of the instantiated formula as follows:

$$\begin{aligned}&\mathsf{instantiate}: (\phi : \mathsf{ctx},\ T : [\phi]\,\mathsf{Type},\ P : [\phi,\ x : T]\,\mathsf{Prop},\ a : [\phi]\,T) \to \\ &\qquad\qquad \mathsf{LT}([\phi]\,\forall x : T, P) \to \mathsf{LT}([\phi]\,P/[\mathsf{id}_\phi,\ a])\\ &\mathsf{instantiate}\ \phi\ T\ P\ a\ \mathsf{pf}\ = \mathsf{let}\ \langle H \rangle\ = \mathsf{pf}\ \mathsf{in}\ \langle H\ a \rangle\end{aligned}$$

From here on, we will omit details about contexts and substitutions in the interest of presentation.

Pattern matching over terms. The most important new construct that VeriML supports is a pattern matching construct over logical terms denoted as holcase. This construct is used for dependent matching of a logical term against a set of patterns. The return clause specifies its return type; we omit it when it is easy to infer. Patterns are normal terms that include unification variables, which can be present under binders. This is the essential reason why contextual terms are needed.

Pattern matching over environments. For the purposes of our development, it is very useful to support one more pattern matching construct: matching over logical variable contexts. When trying to construct a certain proof, the logical environment represents what the current proof context is: what the current logical hypotheses at hand are, what types of terms have been quantified over, etc. By being able to pattern match over the environment, we can "look up" things in our current set of hypotheses, in order to prove further propositions. We can thus view the current environment as representing a simple form of the current *proof state*; the pattern matching construct enables us to manipulate it in a type-safe manner.

One example is an "assumption" tactic, that tries to prove a proposition by searching for a matching hypotheses in the context:

$$\begin{aligned}&\mathsf{assumption}: (\phi : \mathsf{ctx}, P : \mathsf{Prop}) \to \mathsf{option}\ \mathsf{LT}(P)\\ &\mathsf{assumption}\ \phi\ P =\\ &\quad \mathsf{ctxcase}\ \phi\ \mathsf{of}\\ &\qquad \phi',\ H : P \mapsto \mathsf{return}\ \langle H \rangle\\ &\qquad \mid \phi',\ \_ \qquad \mapsto \mathsf{assumption}\ \phi'\ P\end{aligned}$$

Proof object erasure semantics (new feature). The only construct that can influence the evaluation of a program based on the structure of a logical term is the pattern matching construct. For our purposes, pattern matching on proof objects is not necessary – we never look into the structure of a completed proof. Thus we can have the typing rules of the pattern matching construct specifically disallow matching on proof objects.

In that case, we can define an alternate operational semantics for our language where all proof objects are *erased* before using the original small-step reduction rules. Because of type safety, these proof-erasure semantics are guaranteed to yield equivalent results: even if no proof objects are generated, they are still bound to exist.

Implicit arguments. Let us consider again the *instantiate* function defined earlier. This function expects five arguments. From its type alone, it is evident that only the last two arguments are strictly necessary. The last argument, corresponding to a proof expression for the proposition $\forall x : T, P$, can be used to reconstruct exactly the arguments ϕ, T and P. Furthermore, if we know what the resulting type of a call to the function needs to be, we can choose even the instantiation argument a appropriately. We employ a simple inferrence mechanism so that such arguments are omitted from our programs. This feature is also crucial in our development in order to implicitly maintain and utilize the current proof state within our proof scripts.

$$
\begin{array}{rrl}
\text{(sorts)} & s & ::= \mathsf{Type} \mid \mathsf{Type}' \\
\text{(kinds)} & \mathcal{K} & ::= \mathsf{Prop} \mid \mathsf{Nat} \mid \mathcal{K}_1 \to \mathcal{K}_2 \\
\text{(props.)} & P & ::= P_1 \to P_2 \mid \forall x : \mathcal{K}.P \mid x \mid \mathsf{True} \mid \mathsf{False} \mid P_1 \wedge P_2 \mid \cdots \\
\text{(dom.obj.)} & d & ::= \mathsf{Zero} \mid \mathsf{Succ}\ d \mid P \mid \cdots \\
\text{(proof objects)} & \pi & ::= x \mid \lambda x : P.\pi \mid \pi_1\ \pi_2 \mid \lambda x : \mathcal{K}.\pi \mid \pi\ d \mid \cdots \\
\text{(HOL terms)} & t & ::= s \mid \mathcal{K} \mid P \mid d \mid \pi
\end{array}
$$

Selected rules:

$$
\frac{\Psi;\ \Phi, x : P \vdash \pi : P'}{\Psi;\ \Phi \vdash \lambda x : P.\pi : P \to P'} \to \textsc{Intro}
\qquad
\frac{\Psi;\ \Phi \vdash \pi : P \to P' \qquad \Psi;\ \Phi \vdash \pi' : P}{\Psi;\ \Phi \vdash \pi\ \pi' : P'} \to \textsc{Elim}
$$

Figure 6. Syntax and selected rules of the logic language λHOL

$$
\textsc{Conversion}\ \frac{\Psi;\ \Phi \vdash_c \pi : P \qquad P =_{\beta\mathbb{N}} P'}{\Psi;\ \Phi \vdash_c \pi : P'}
$$

$d \to_{\beta\mathbb{N}} d'$:

$$
\begin{array}{l}
(\lambda x : \mathcal{K}.d)\ d' \to_{\beta\mathbb{N}} d[d'/x] \\
\mathsf{natElim}_{\mathcal{K}}\ d_z\ d_s\ \mathsf{zero} \to_{\beta\mathbb{N}} d_z \\
\mathsf{natElim}_{\mathcal{K}}\ d_z\ d_s\ (\mathsf{succ}\ d) \to_{\beta\mathbb{N}} d_s\ d\ (\mathsf{natElim}_{\mathcal{K}}\ d_z\ d_s\ d)
\end{array}
$$

$d =_{\beta\mathbb{N}} d'$: is the compatible, reflexive, symmetric and transitive closure of $d \to_{\beta\mathbb{N}} d'$

Figure 7. Extending λHOL with the conversion rule (λHOL_c)

Minimal staging support (new feature). Using the language we have seen so far we are able to write powerful tactics using a general-purpose programming model. But what if, inside our programs, we have calls to tactics where all of their arguments are constant? Presumably, those tactic calls could be evaluated to proof objects prior to tactic invocation. We could think of this as a form of generalized constant folding, which has one intriguing benefit: we can tell statically whether the tactic calls succeed or not.

This paper is exactly about exploring this possibility. Towards this effect, we introduce a rudimentary staging construct in our computational language. This takes the form of a letstatic construct, which binds a static expression to a variable. The static expression is evaluated during stage one (see Fig. 2), and can only depend on other static expressions. Details of this construct are presented in Fig. 11d and also in Sec. 6. After this addition, expressions in our language have a three-phase lifetime, that are also shown in Fig. 2.

- type-checking, where the well-formedness of expressions according to the rules of the language is checked, and inference of implicit arguments is performed
- static evaluation, where expressions inside letstatic are reduced to values, yielding a residual expression
- run-time, where the residual expression is evaluated

4. Extensible conversion rule

With these tools at hand, let us now return to the first issue that motivates us: the fact that proof checking is rigid and cannot be extended with user-defined procedures. As we have said in our introduction, many modern proof assistants are based on logics that include a *conversion rule*. This rule essentially identifies propositions up to some equivalence relation: usually this is equivalence up to partial evaluation of the functions contained within propositions.

The supported relation is decided when the logic is designed. Any extension to this relation requires a significant amount of work, both in terms of implementation, and in terms of metatheoretic proof required. This is evidenced by projects that extend the conversion rule in Coq, such as Blanqui et al. [1999] and Strub [2010]. Even if user extensions are supported, those only take the form of first-order theories. Can we do better than this, enabling arbitrarily complex user extensions, written with the full power of ML, yet maintaining soundness?

It turns out that we can: this is the subject of this section. The key idea is to recognize that the conversion rule is essentially a tactic, embedded within the type checker of the logic. Calls to this tactic are made implicitly as part of checking a given proof object for validity. So how can we support a flexible, extensible alternative? Instead of hardcoding a conversion tactic within the logic type checker, we can program a type-safe version of the same tactic within VeriML, with the requirement that it provides proof of the claimed equivalence. Instead of calling the conversion tactic as part of proof checking, we use staging to call the tactic statically – after (VeriML) type checking, but before runtime execution. This can be viewed as a second, potentially non-terminating proof checking stage. Users are now free to write their own conversion tactics, extending the static checking available for proof objects and proof scripts. Still, soundness is maintained, since full proof objects in the original logic can always be constructed. As an example, we have extended the conversion rule that we use by a congruence closure procedure, which makes use of mutable data structures, and by an arithmetic simplification procedure.

4.1 Introducing: the conversion rule

First, let us present what the conversion rule really is in more detail. We will base our discussion on a simple type-theoretic higher-order logic, based on the λHOL logic as described in Barendregt and Geuvers [1999], and used in our original work on VeriML [Stampoulis and Shao 2010]. We can think of such a logic composed by the following broad classes: the objects of the domain of discourse d, which are the objects that the logic reasons about, such as natural numbers and lists; their classifiers, the kinds $\mathcal{K}$ (classified in turn by sorts s); the propositions P; and the derivations, which prove that a certain proposition is true. We can represent derivations in a linear form as terms π in a typed lambda-calculus; we call such terms proof objects, and their types represent propositions in the logic. Checking whether a derivation is a valid proof of a certain proposition amounts to type-checking its corresponding proof object. Some details of this logic are presented in Fig. 6; the interested reader can find more information about it in the above references and in the appendix (Sec. A).

In Fig. 6, we show what the conversion rule looks like for this logic: it is a typing judgement that effectively identifies propositions up to an equivalence relation, with respect to checking proof objects. We call this version of the logic λHOL_c and use $\vdash_c$ to denote its entailment relation. The equivalence relation we consider in the conversion rule is evaluation up to β-reductions and uses of primitive recursion of natural numbers, denoted as $\mathsf{natElim}$. In this way, trivial arguments based on this notion of computation alone need not be witnessed, as for example is the fact that $(\mathsf{Succ}\ x) + y = \mathsf{Succ}\ (x + y)$ – when the addition function is defined by primitive recursion on the first argument. Of course, this is only a very basic use of the conversion rule. It is possible to omit larger proofs through much more sophisticated uses. This leads to simpler proofs and smaller proof objects.

Still, when using this approach, the choice of what relation is supported by the conversion rule needs to be made during the definition of the logic. This choice permeates all aspects of the metatheory of the logic. It is easy to see why, even with the tiny fragment of logic we have introduced. Most typing rules for proof objects in

$$\frac{\Psi;\ \Phi \vdash_e d_1 : \mathcal{K} \qquad \Psi;\ \Phi \vdash_c d_2 : \mathcal{K}}{\Psi;\ \Phi \vdash_e d_1 = d_2 : \mathsf{Prop}} \qquad \frac{\Psi;\ \Phi \vdash_e d : \mathcal{K}}{\Psi;\ \Phi \vdash_e \mathsf{refl}\ d : d = d}$$

$$\frac{\Psi;\ \Phi,\ x : \mathcal{K} \vdash_e P : \mathsf{Prop} \qquad \Psi;\ \Phi \vdash_e d_1 : \mathcal{K} \qquad \Psi;\ \Phi \vdash_e \pi : P[d_1/x] \qquad \Psi;\ \Phi \vdash_e \pi' : d_1 = d_2}{\Psi;\ \Phi \vdash_e \mathsf{leibniz}\ (\lambda x : \mathcal{K}.P)\ \pi\ \pi' : P[d_2/x]}$$

$$\frac{\Psi;\ \Phi,\ x : \mathcal{K} \vdash_e \pi : d_1 = d_2}{\Psi;\ \Phi \vdash_e \mathsf{lamEq}\ (\lambda x : \mathcal{K}.\pi) : (\lambda x : \mathcal{K}.d_1) = (\lambda x : \mathcal{K}.d_2)}$$

$$\frac{\Psi;\ \Phi,\ x : \mathcal{K} \vdash_e \pi : d_1 = d_2 \qquad \Psi;\ \Phi \vdash_e d_1 : \mathsf{Prop}}{\Psi;\ \Phi \vdash_e \mathsf{forallEq}\ (\lambda x : \mathcal{K}.\pi) : (\forall x : \mathcal{K}.d_1) = (\forall x : \mathcal{K}.d_2)}$$

$$\frac{\Psi;\ \Phi,\ x : \mathcal{K} \vdash_e d : \mathcal{K}' \qquad \Psi;\ \Phi \vdash_e d' : \mathcal{K}}{\Psi;\ \Phi \vdash_e \mathsf{betaEq}\ (\lambda x : \mathcal{K}.d)\ d' : (\lambda x : \mathcal{K}.d)\ d' = d[d'/x]}$$

Axioms assumed:

$$\begin{array}{lcl} \mathsf{natElimBase}_{\mathcal{K}} & : & \forall f_z.\forall f_s.\mathsf{natElim}_{\mathcal{K}}\ f_z\ f_s\ \mathsf{zero} = f_z \\ \mathsf{natElimStep}_{\mathcal{K}} & : & \forall f_z.\forall f_s.\forall n.\ \mathsf{natElim}_{\mathcal{K}}\ f_z\ f_s\ (\mathsf{succ}\ n) = \\ & & \qquad f_s\ n\ (\mathsf{natElim}_{\mathcal{K}}\ f_z\ f_s\ n) \end{array}$$

Figure 8. Extending λHOL with explicit equality ($\lambda\mathrm{HOL}_e$)

the logic are similar to the rules →INTRO and →ELIM: they are syntax-directed. This means that upon seeing the associated proof object constructor, like $\lambda x : P.\pi$ in the case of →INTRO, we can directly tell that it applies. If all rules were syntax directed, it would be entirely simple to prove that the logic is sound by an inductive argument: essentially, since no proof constructor for False exists, there is no valid derivation for False.

In this logic, the only rule that is not syntax directed is exactly the conversion rule. Therefore, in order to prove the soundness of the logic, we have to show that the conversion rule does not somehow introduce a proof of False. This means that proving the soundness of the logic passes essentially through the specific relation we have chosen for the conversion rule. Therefore, this approach is foundationally limited from supporting user extensions, since any new extension would require a new metatheoretic result in order to make sure that it does not violate logical soundness.

4.2 Throwing conversion away

Since having a fixed conversion rule is bound to fail if we want it to be extensible, what choice are we left with, but to throw it away? This radical sounding approach is what we will do here. We can replace the conversion rule by an explicit notion of equality, and provide explicit proof witnesses for rewriting based on that equality. Essentially, all the points where the conversion rule was alluded to and proofs were omitted, need now be replaced by proof objects witnessing the equivalence. Some details for the additions required to the base λHOL logic are shown in Fig. 8, yielding the $\lambda\mathrm{HOL}_e$ logic. There are good reasons for choosing this version: first, the proof checker is as simple as possible, and does not need to include the conversion checking routine. We could view this routine as performing proof search over the replacement rules, so it necessarily is more complicated, especially since it needs to be relatively efficient. Also, the metatheory of the logic itself can be simplified. Even when the conversion rule is supported, the metatheory for the associated logic is proved through the explicit

```
βℕequal : (ϕ : ctx, T : Type, t1 : T, t2 : T) → option LT(t1 = t2)
βℕequal ϕ T t1 t2 =
  holcase whnf ϕ T t1, whnf ϕ T t2 of
    ((ta : T' → T) tb), (tc td) ↦
      do ⟨pf1⟩ ← βℕequal ϕ (T' → T) ta tc
         ⟨pf1⟩ ← βℕequal ϕ T' tb td
         return ⟨··· proof of ta tb = tc td ···⟩
  | (ta → tb), (tc → td) ↦
      do ⟨pf1⟩ ← βℕequal ϕ Prop ta tc
         ⟨pf1⟩ ← βℕequal ϕ Prop tb td
         return ⟨··· proof of ta → tb = tc → td ···⟩
  | (λx : T.t1), (λx : T.t2) ↦
      do ⟨pf⟩ ← βℕequal [ϕ, x : T] Prop t1 t2
         return ⟨··· proof of λx : T.t1 = λx : T.t2 ···⟩
  | t1, t1 ↦ do return ⟨··· proof of t1 = t1 ···⟩
  | t1, t2 ↦ None

requireEqual : (ϕ : ctx, T : Type, t1 : T, t2 : T).LT(t1 = t2)
requireEqual ϕ T t1 t2 =
  match βℕequal ϕ T t1 t2 with Some x ↦ x | None ↦ error
```

Figure 9. VeriML tactic for checking equality up to β-conversion

equality approach; this is because model construction for a logic benefits from using explicit equality [Siles and Herbelin 2010].

Still, this approach has a big disadvantage: the proof objects soon become extremely large, since they include painstakingly detailed proofs for even the simplest of equivalences. This precludes their use as independently checkable proof certificates that can be sent to a third party. It is possible that this is one of the reasons why systems based on logics with explicit equalities, such as HOL4 [Slind and Norrish 2008] and Isabelle/HOL [Nipkow et al. 2002], do not generate proof objects by default.

4.3 Getting conversion back

We will now see how it is possible to reconcile the explicit equality based approach with the conversion rule: we will gain the conversion rule back, albeit it will remain completely outside the logic. Therefore we will be free to extend it, all the while without risking introducing unsoundness in the logic, since the logic remains fixed (λHOL_e as presented above).

We do this by revisiting the view of the conversion rule as a special "trusted" tactic, through the tools presented in the previous section. First, instead of hardcoding a conversion tactic in the type checker, we program a *type-safe conversion tactic*, utilizing the features of VeriML. Based on typing alone we require that it returns a valid proof of the claimed equivalences:

$$\beta\mathbb{N}\text{equal} : (\phi : \text{ctx},\ T : \text{Type},\ t : T,\ t' : T) \rightarrow \text{option}\ \text{LT}(t = t')$$

Second, we evaluate this tactic under *proof erasure semantics*. This means that no proof objects are produced, leading to the same space gains as the original conversion rule. Third, we use the staging construct in order to *check conversion statically*.

```
whnf : (ϕ : ctx, T : Type, t : T) → (t' : T) × LT(t = t')
whnf ϕ T t = holcase t of
  (t1 : T' → T)(t2 : T') ↦
    let ⟨t1', pf1⟩ = whnf ϕ (T' → T) t1 in
    holcase t1' of
      λx : T'.tf  ↦  ⟨[ϕ] tf/[id_Φ, t2], ···⟩
      | t1'       ↦  ⟨[ϕ] t1' t2, ···⟩
  | natElim_K fz fs n ↦
      let ⟨n', pf1⟩ = whnf ϕ Nat n in holcase n' of
            zero      ↦  ⟨[ϕ] fz, ···⟩
          | succ n'   ↦  ⟨[ϕ] fs n' (natElim_K fz fs n'), ···⟩
          | n'        ↦  ⟨[ϕ] natElim_K fz fs n', ···⟩
  | t ↦ ⟨t, ···⟩
```

Figure 10. VeriML tactic for rewriting to weak head-normal form

Details. We now present our approach in more detail. First, in Fig. 9, we show a sketch of the code behind the type-safe conversion check tactic. It works by first rewriting its input terms into weak head-normal form, via the whnf function in Fig. 10, and then recursively checking their subterms for equality. In the equivalence checking function, more cases are needed to deal with quantification; while in the rewriting procedure, a recursive call is missing, which would complicate our presentation here. We also define a version of the tactic that raises an error instead of returning an option type if we fail to prove the terms equal, which we call requireEqual. The full details can be found in our implementation.

The code of the $\beta\mathbb{N}$equal tactic is in fact entirely similar to the code one would write for the conversion check routine inside a logic type checker, save for the extra types and proof objects. It therefore follows trivially that everything that holds for the standard implementation of the conversion check also holds for this code: e.g. it corresponds exactly to the $=_{\beta\mathbb{N}}$ relation as defined in the logic; it is bound to terminate because of the strong normalization theorem for this relation; and its proof-erased version is at least as trustworthy as the standard implementation.

Furthermore, given this code, we can produce a form of *typed proof scripts* inside VeriML that correspond exactly to proof objects in the logic with the conversion rule, both in terms of their actual code, and in terms of the steps required to validate them. This is done by constructing a proof script in VeriML by induction on the derivation of the proof object in λHOL_c, replacing each proof object constructor by an equivalent VeriML tactic as follows:

constructor	to tactic	of type
$\lambda x : P.\pi$	Assume e	$\mathsf{LT}([\phi,\ H : P]\,P') \to \mathsf{LT}(P \to P')$
$\pi_1\ \pi_2$	Apply $e_1\ e_2$	$\mathsf{LT}(P \to P') \to \mathsf{LT}(P) \to \mathsf{LT}(P')$
$\lambda x : \mathcal{K}.\pi$	Intro e	$\mathsf{LT}([\phi,\ x : T]\,P') \to \mathsf{LT}(\forall x : T, P')$
$\pi\ d$	Inst $e\ a$	$\mathsf{LT}(\forall x : T, P) \to (a : T) \to \mathsf{LT}(P/[\mathsf{id},\ a])$
c	Lift c	$(H : P) \to \mathsf{LT}(P)$
(conversion)	Conversion	$\mathsf{LT}(P) \to \mathsf{LT}(P = P') \to \mathsf{LT}(P')$

Here we have omitted the current logical environment ϕ; it is maintained through syntactic means as discussed in Sec. 7 and through type inference. The only subtle case is conversion. Given the transformed proof e for the

proof object π contained within a use of the conversion rule, we call the conversion tactic as follows:

$$\mathsf{letstatic\ pf\ =\ requireEqual}\ P\ P'\ \mathsf{in\ Conversion\ e\ pf}$$

The arguments to requireEqual can be easily inferred, making crucial use of the rich type information available. Conversion could also be used implicitly in the other tactics. Thus the resulting expression looks entirely identical to the original proof object.

Correspondence with original proof object. In order to elucidate the correspondence between the resulting proof script expression and the original proof object, it is fruitful to view the proof script as a proof certificate, sent to a third party. The steps required to check whether it constitutes a valid proof are the following. First, the whole expression is checked using the type checker of the computational language. Then, the calls to the requireEqual function are evaluated during stage one, using proof erasure semantics. We expect them to be successful, just as we would expect the conversion rule to be applicable when it is used. Last, the rest of the tactics are evaluated; by a simple argument, based on the fact that they do not use pattern matching or side-effects, they are guaranteed to terminate and produce a proof object in $\lambda\mathrm{HOL}_e$. This validity check is entirely equivalent to the behavior of type-checking the $\lambda\mathrm{HOL}_c$ proof object, save for pushing all conversion checks towards the end.

4.4 Extending conversion at will

In our treatment of the conversion rule we have so far focused on regaining the $\beta\mathbb{N}$ conversion in our framework. Still, there is nothing confining us to supporting this conversion check only. As long as we can program a conversion tactic in VeriML that has the right type, it can safely be made part of our conversion rule.

For example, we have written an eufEqual function, which checks terms for equivalence based on the equality with uninterpreted functions decision procedure. It is adapted from our previous work on VeriML [Stampoulis and Shao 2010]. This equivalence checking tactic isolates hypotheses of the form $d_1 = d_2$ from the current context, using the newly-introduced context matching support. Then, it constructs a union-find data structure in order to form equivalence classes of terms. Based on this structure, and using code similar to $\beta\mathbb{N}$equal (recursive calls on subterms), we can decide whether two terms are equal up to simple uses of the equality hypotheses at hand. We have combined this tactic with the original $\beta\mathbb{N}$equal tactic, making the implicit equivalence supported similar to the one in the Calculus of Congruent Constructions [Blanqui et al. 2005]. This demonstrates the flexibility of this approach: equivalence checking is extended with a sophisticated decision procedure, which is programmed using its original, imperative formulation. We have programmed both the rewriting procedure and the equality checking procedure in an extensible manner, so that we can globally register further extensions.

4.5 Typed proof scripts as certificates

Earlier we discussed how we can validate the proof scripts resulting from turning the conversion rule into explicit tactic calls. This discussion shows an interesting aspect of typed proof scripts: they can be viewed as a proof witness that is a flexible compromise between untyped proof scripts and proof objects. When a typed proof script consists only of static calls to conversion tactics and uses of total tactics, it can be thought of as a proof object in a logic with the corresponding conversion rule. When it also contains other tactics, that perform potentially expensive proof search, it corresponds more closely to an untyped proof script, since it needs to be fully evaluated. Still, we are allowed to validate parts of it statically. This is especially useful when developing the proof script, because we can avoid the evaluation of expensive tactic calls while we focus on getting the skeleton of the proof correct.

Using proof erasure for evaluating requireEqual is only one of the choices the receiver of such a proof certificate can make. Another choice would be to have the function return an actual proof object, which we can check using the $\lambda\mathrm{HOL}_e$ type checker. In that case, the VeriML interpreter does not need to become part of

the trusted base of the system. Last, the 'safest possible' choice would be to avoid doing any evaluation of the function, and ask the proof certificate provider to do the evaluation of requireEqual themselves. In that case, no evaluation of computational code would need to happen at the proof certificate receiver's side. This mitigates any concerns one might have for code execution as part of proof validity checking, and guarantees that the small λHOL_e type checker is the trusted base in its entirety. Also, the receiver can decide on the above choices selectively for different conversion tactics – e.g. use proof erasure for $\beta\mathbb{N}$equal but not for eufEqual, leading to a trusted base identical to the λHOL_c case. This means that the choice of the conversion rule rests with the proof certificate receiver and not with the designer of the logic. Thus the proof certificate receiver can choose the level of trust they require at will.

5. Static proof scripts

In the previous section, we have demonstrated how proof checking for typed proof scripts can be made user-extensible, through a new treatment of the conversion rule. It makes use of user-defined, type-safe tactics, which are evaluated statically. The question that remains is what happens with respect to proofs within tactics. If a proof script is found within a tactic, must we wait until that evaluation point is reached to know whether the proof script is correct or not? Or is there a way to check this statically, as soon as the tactic is defined?

In this section we show how this is possible to do in VeriML using the staging construct we have introduced. Still, in this case matters are not as simple as evaluating certain expressions statically rather than dynamically. The reason is that proof scripts contained within tactics mention uninstantiated meta-variables, and thus cannot be evaluated through staging. We resolve this by showing the existence of a transformation, which "collapses" logical terms from an arbitrary meta-variables context into the empty one.

We will focus on the case of developing conversion routines, similar to the ones we saw earlier. The ideas we present are generally applicable when writing other types of tactics as well; we focus on conversion routines in order to demonstrate that the two main ideas we present in this paper can work in tandem.

A rewriter for plus. We will consider the case of writing a rewriter –similar to whnf– for simplifying expressions of the form $x+y$, depending on the second argument. The addition function is defined by induction on the first argument, as follows:

$$(+) = \lambda x.\lambda y.\mathsf{natElim}_{\mathsf{Nat}}\ y\ (\lambda p.\lambda r.\mathsf{Succ}\ r)\ x$$

In order for rewriters to be able to use existing as well as future rewriters to perform their recursive calls, we write them in the open recursion style – they receive a function of the same type that corresponds to the "current" rewriter. The code looks as follows:

$$
\begin{array}{l}
\mathsf{rewriterType} = (\phi : \mathsf{ctx}, T : \mathsf{Type}, t : T) \to (t' : T) \times \mathsf{LT}(t = t') \\
\mathsf{plusRewriter1} : \ \mathsf{rewriterType} \to \mathsf{rewriterType} \\
\mathsf{plusRewriter1}\ \mathsf{recursive}\ \phi\ T\ t = \mathsf{holcase}\ t\ \mathsf{with} \\
\quad x+y \mapsto \\
\qquad \mathsf{let}\ \langle y',\ \langle \mathsf{pfy}' \rangle \rangle = \mathsf{recursive}\ \phi\ y\ \mathsf{in} \\
\qquad \mathsf{let}\ \langle t',\ \langle \mathsf{pft}' \rangle \rangle\ = \\
\qquad\quad \mathsf{holcase}\ y'\ \mathsf{return}\ \Sigma t' : [\phi]\,\mathsf{Nat.LT}([\phi]\,x+y' = t')\ \mathsf{of} \\
\qquad\quad\ \ 0 \qquad\ \mapsto \langle x, \cdots\ \textit{proof of}\ \ x+0 = x\ \cdots \rangle \\
\qquad\quad \mid \mathsf{Succ}\ y' \mapsto \Big\langle \mathsf{Succ}(x+y'), \\
\qquad\qquad\qquad \cdots\ \textit{proof of}\ \ x+\mathsf{Succ}\ y' = \mathsf{Succ}\ (x+y')\ \cdots \Big\rangle \\
\qquad\quad \mid y' \qquad\ \mapsto \langle x+y', \cdots\ \textit{proof of}\ \ x+y' = x+y'\ \cdots \rangle \\
\qquad \mathsf{in}\ \langle t',\ \langle \cdots\ \textit{proof of}\ \ x+y = t'\ \cdots \rangle \rangle \\
\quad \mid t \qquad \mapsto \langle t, \cdots\ \textit{proof of}\ \ t = t\ \cdots \rangle
\end{array}
$$

While developing such a tactic, we can leverage the VeriML type checker to know the types of missing proofs. But how do we fill them in? For the interesting cases of $x+0=x$ and $x+\mathsf{Succ}\ y' = \mathsf{Succ}\ (x+y')$, we would certainly need to prove the corresponding lemmas. But for the rest of the cases, the corresponding lemmas would be uninteresting and tedious to state, such as the following for the $x+y=t'$ case:

$$\mathsf{lemma1} : \forall x,y,y',t',y=y' \rightarrow (x+y'=t') \rightarrow x+y=t$$

Stating and proving such lemmas soon becomes a hindrance when writing tactics. An alternative is to use the congruence closure conversion rule to solve this trivial obligation for us directly at the point where it is required. Our first attempt would be:

$$\begin{array}{l}\text{proof of } x+y=t' \ \equiv \\ \quad \mathsf{let}\ \langle \mathsf{pf} \rangle = \mathsf{requireEqual}\ \ [\phi, H_1 : y=y', H_2 : x+y'=t']\ \ (x+y)\ \ t' \\ \quad \mathsf{in}\ \left\langle [\phi]\,\mathsf{pf}/[\mathsf{id}_\phi,\ \mathsf{pfy}',\ \mathsf{pft}'] \right\rangle\end{array}$$

The benefit of this approach is evident when utilizing implicit arguments, since most of the details can be inferred and therefore omitted. Here we had to alter the environment passed to requireEqual, which includes several extra hypotheses. Once the resulting proof has been computed, the hypotheses are substituted by the actual proofs that we have.

The problem with this approach is two-fold: first, the call to the requireEqual tactic is recomputed every time we reach that point of our function. For such a simple tactic call, this does not impact the runtime significantly; still, if we could avoid it, we would be able use more sophisticated and expensive tactics. The second problem is that if for some reason the requireEqual is not able to prove what it is supposed to, we will not know until we actually reach that point in the function.

Moving to static proofs. This is where using the letstatic construct becomes essential. We can evaluate the call to requireEqual statically, during stage one interpretation. Thus we will know at the time that plusRewriter1 is defined whether the call succeeded; also, it will be replaced by a concrete value, so it will not affect the runtime behavior of each invocation of plusRewriter1 anymore. To do that, we need to avoid mentioning any of the metavariables that are bound during runtime, like x, y, and t'. This is done by specifying an appropriate environment in the call to requireEqual, similarly to the way we incorporated the extra knowledge above and substituted it later. Using this approach, we have:

$$\begin{array}{l}\text{proof of } x+y=t' \ \equiv \\ \quad \mathsf{letstatic}\ \langle \mathsf{pf} \rangle\ = \\ \quad\ \ \mathsf{let}\ \phi'\ = [x,y,y',t' : \mathsf{Nat}, H_1 : y=y', H_2 : x+y'=t']\ \mathsf{in} \\ \quad\ \ \mathsf{requireEqual}\ \phi'\ (x+y)\ t' \\ \quad \mathsf{in}\ \left\langle [\phi]\,\mathsf{pf}/[x/\mathsf{id}_\phi, y/\mathsf{id}_\phi, y'/\mathsf{id}_\phi, t'/\mathsf{id}_\phi, \mathsf{pfy}'/\mathsf{id}_\phi, \mathsf{pft}'/\mathsf{id}_\phi] \right\rangle\end{array}$$

What we are essentially doing here is replacing the meta-variables by normal logical variables, which our tactics can deal with. The meta-variable context is "collapsed" into a normal context; proofs are constructed using tactics in this environment; last, the resulting proofs are transported back into the desired context by substituting meta-variables for variables. We have explicitly stated the substitutions in order to distinguish between normal logical variables and meta-variables.

The reason why this transformation needs to be done is that functions in our computational language can only manipulate logical terms that are open with respect to a normal variables context; not logical terms that are open with respect to the meta-variables context too. A much more complicated, but also more flexible alternative to using this "collapsing" trick would be to support meta-n-variables within our computational language directly.

Overall, this approach is entirely similar to proving the auxiliary lemma mentioned above, prior to the tactic definition. The benefit is that by leveraging the type information together with type inference, we can avoid

stating such lemmas explicitly, while retaining the same runtime behavior. We thus end up with very concise proof expressions that are statically validated. We introduce syntactic sugar for binding a static proof script to a variable, and then performing a substitution to bring it into the current context, since this is a common operation.

$$\langle e \rangle_{\text{static}} \equiv \text{letstatic } \langle \text{pf} \rangle \ = \ e \text{ in } \langle [\phi]\, \text{pf} / \cdots \rangle$$

Based on these, the trivial proofs in the above tactic can be filled in using a simple $\langle \text{requireEqual} \rangle_{\text{static}}$ call; for the other two we use $\langle \text{Instantiate } (\text{NatInduction requireEqual requireEqual})\ x \rangle_{\text{static}}$.

After we define plusRewriter1, we can register it with the global equivalence checking procedure. Thus, all later calls to requireEqual will benefit from this simplification. It is then simple to prove commutativity for addition:

$$\begin{array}{lcl} \text{plusComm} & : & \text{LT}(\forall x, y. x + y = y + x) \\ \text{plusComm} & = & \text{NatInduction requireEqual requireEqual} \end{array}$$

Based on this proof, we can write a rewriter that takes commutativity into account and uses the hash values of logical terms to avoid infinite loops. We have worked on an arithmetic simplification rewriter that is built by layering such rewriters together, using previous ones to aid us in constructing the proofs required in later ones. It works by converting expressions into a list of monomials, sorting the list based on the hash values of the variables, and then factoring monomials on the same variable. Also, the eufEqual procedure mentioned earlier has all of its associated proofs automated through static proof scripts, using a naive, potentially non-terminating, equality rewriter.

Is collapsing always possible? A natural question to ask is whether collapsing the metavariables context into a normal context is always possible. In order to cast this as a more formal question, we notice that the essential step is replacing a proof object π of type $[\Phi]\, t$, typed under the meta-variables environment Ψ, by a proof object π' of type $[\Phi']\, t'$ typed under the empty meta-variables environment. There needs to be a substitution so that π' gets transported back to the Φ, Ψ environment, and has the appropriate type.

We have proved that this is possible under certain restrictions: the types of the metavariables in the current context need to depend on the same free variables context Φ_{max}, or prefixes of that context. Also the substitutions they are used with need to be prefixes of the identity substitution for Φ_{max}. Such terms are characterized as collapsible. We have proved that collapsible terms can be replaced using terms that do not make use of metavariables; more details can be found in Sec. 6 and in Sec. F of the appendix.

This restriction corresponds very well to the treatment of variable contexts in the Delphin language. This language assumes an ambient context of logical variables, instead of full, contextual modal terms. Constructs to extend this context and substitute a specific variable exist. If this last feature is not used, the ambient context grows monotonically and the mentioned restriction holds trivially. In our tests, this restriction has not turned out to be limiting.

6. Metatheory

We have completed an extensive reworking of the metatheory of VeriML, in order to incorporate the features that we have presented in this paper. Our new metatheory includes a number of technical advances compared to our earlier work [Stampoulis and Shao 2010]. We will present a technical overview of our metatheory in this section; full details can be found in the appendix.

Variable representation technique. Though our metatheory is done on paper, we have found that using a concrete variable representation technique elucidates some aspects of how different kinds of substitutions work in our language, compared to having normal named variables. For example, instantiating a context variable with

Syntax of the logic

$$\begin{array}{l}\textit{(terms) } t ::= s \mid c \mid f_i \mid b_i \mid \lambda(t_1).t_2 \mid t_1\ t_2 \mid \Pi(t_1).t_2 \mid t_1 = t_2 \mid \mathsf{refl}\ t \mid \mathsf{leibniz}\ t_1\ t_2 \mid \mathsf{lamEq}\ t \mid \mathsf{forallEq}\ t_1\ t_2 \mid \mathsf{betaEq}\ t_1\ t_2 \\ \textit{(sorts) } s ::= \mathsf{Prop} \mid \mathsf{Type} \mid \mathsf{Type}' \qquad \textit{(var. context)} \quad \Phi ::= \bullet \mid \Phi,\ t \qquad \textit{(substitutions)} \quad \sigma ::= \bullet \mid \sigma,\ t\end{array}$$

Example of representation: $a : \mathsf{Nat} \vdash \lambda x : \mathsf{Nat}.(\lambda y : \mathsf{Nat}.\mathsf{refl}\ (\mathsf{plus}\ a\ y))(\mathsf{plus}\ a\ x) \mapsto \mathsf{Nat} \vdash \lambda(\mathsf{Nat}).(\lambda(\mathsf{Nat}).\mathsf{refl}\ (\mathsf{plus}\ f_0\ b_0))\ (\mathsf{plus}\ f_0\ b_0)$

Freshen: $\lceil t \rceil^n_m$

$$\begin{array}{lcl}\lceil f_i \rceil^n_m & = & f_i \\ \lceil b_n \rceil^n_m & = & f_m \\ \lceil b_i \rceil^n_m & = & b_i \text{ when } i < n \\ \lceil (\lambda(t_1).t_2) \rceil^n & = & \lambda(\lceil t_1 \rceil^n).\ \lceil t_2 \rceil^{n+1} \\ \lceil t_1\ t_2 \rceil & = & \lceil t_1 \rceil\ \lceil t_2 \rceil\end{array}$$

Bind: $\lfloor t \rfloor^n_m$

$$\begin{array}{lcl}\lfloor f_{m-1} \rfloor^n_m & = & b_n \\ \lfloor f_i \rfloor^n_m & = & f_i \text{ when } i < m-1 \\ \lfloor b_i \rfloor & = & b_{i+1} \\ \lfloor (\lambda(t_1).t_2) \rfloor & = & \lambda(\lfloor t_1 \rfloor^n).\ \lfloor t_2 \rfloor^{n+1} \\ \lfloor t_1\ t_2 \rfloor & = & \lfloor t_1 \rfloor\ \lfloor t_2 \rfloor\end{array}$$

(a) Hybrid deBruijn levels-deBruijn indices representation technique

---

Syntax

$$\begin{array}{l}t ::= \cdots \mid f_{\mathbf{I}} \mid X_i/\sigma \qquad \Phi ::= \bullet \mid \Phi,\ t \mid \Phi,\ \phi_i \qquad \sigma ::= \bullet \mid \sigma,\ t \mid \sigma,\ \mathbf{id}(\phi_i) \qquad \textit{(indices) } \mathbf{I} ::= n \mid \mathbf{I} + |\phi_i| \qquad \textit{(ctx.terms) } T ::= [\Phi]t \mid [\Phi]\Phi' \\ \textit{(ctx.kinds) } K ::= [\Phi]t \mid [\Phi]\,\mathsf{ctx} \qquad \textit{(extension context) } \Psi ::= \bullet \mid \Psi,\ K \qquad \textit{(ext. subst.) } \sigma_\Psi ::= \bullet \mid \sigma_\Psi,\ T\end{array}$$

$\Psi;\ \Phi \vdash t : t'$ (sample)

$$\frac{\Phi.\mathbf{I} = t}{\Psi;\ \Phi \vdash f_{\mathbf{I}} : t} \qquad \frac{\Psi;\ \Phi \vdash t_1 : \Pi(t).t' \qquad \Psi;\ \Phi \vdash t_2 : t}{\Psi;\ \Phi \vdash t_1\ t_2 : \lceil t' \rceil \cdot (\mathsf{id}_\Phi, t_2)} \qquad \frac{\Psi.i = [\Phi']t' \qquad \Psi;\ \Phi \vdash \sigma : \Phi'}{\Psi;\ \Phi \vdash X_i/\sigma : t' \cdot \sigma}$$

$\Psi \vdash T : K$

$$\frac{\Psi;\ \Phi \vdash t : t'}{\Psi \vdash [\Phi]t : [\Phi]t'} \qquad \frac{\Psi \vdash \Phi,\ \Phi'\ \mathrm{wf}}{\Psi \vdash [\Phi]\Phi' : [\Phi]\,\mathsf{ctx}}$$

$\Psi \vdash \Phi$ wf (sample)

$$\frac{\Psi \vdash \Phi\ \mathrm{wf} \qquad \Psi.i = [\Phi]\,\mathsf{ctx}}{\Psi \vdash (\Phi,\ \phi_i)\ \mathrm{wf}}$$

(b) Extension variables: meta-variables and context variables

---

Subst. application: $t \cdot \sigma$

$$c \cdot \sigma = c \qquad f_{\mathbf{I}} \cdot \sigma = \sigma.\mathbf{I} \qquad b_i \cdot \sigma = b_i \qquad (\lambda(t_1).t_2) \cdot \sigma = \lambda(t_1 \cdot \sigma).(t_2 \cdot \sigma) \qquad (t_1\ t_2) \cdot \sigma = (t_1 \cdot \sigma)\ (t_2 \cdot \sigma)$$

Ext. subst. application (sample)

$$\begin{array}{ll}(\mathbf{I},\ |\phi_i|) \cdot \sigma_\Psi = (\mathbf{I} \cdot \sigma_\Psi),\ |\Phi'| \text{ when } \sigma_\Psi.i = [\_]\Phi' & (X_i/\sigma) \cdot \sigma_\Psi = t \cdot (\sigma \cdot \sigma_\Psi) \text{ when } \sigma_\Psi.i = [\_]t \\ (\sigma,\ \mathbf{id}(\phi_i)) \cdot \sigma_\Psi = \sigma \cdot \sigma_\Psi,\ \mathsf{id}_{\sigma_\Psi.i} & (\Phi,\ \phi_i) \cdot \sigma_\Psi = \Phi \cdot \sigma_\Psi,\ \Phi' \text{ when } \sigma_\Psi.i = [\_]\Phi'\end{array}$$

$\Psi;\ \Phi \vdash \sigma : \Phi'$

$$\frac{}{\Psi;\ \Phi \vdash \bullet : \bullet} \qquad \frac{\Psi;\ \Phi \vdash \sigma : \Phi' \qquad \Psi;\ \Phi \vdash t : t' \cdot \sigma}{\Psi;\ \Phi \vdash (\sigma,\ t) : (\Phi',\ t')} \qquad \frac{\Psi;\ \Phi \vdash \sigma : \Phi' \qquad \Psi.i = [\Phi']\,\mathsf{ctx} \qquad \Phi',\ \phi_i \subseteq \Phi}{\Psi;\ \Phi \vdash (\sigma,\ \mathbf{id}(\phi_i)) : (\Phi',\ \phi_i)}$$

$\Psi \vdash \sigma_\Psi : \Psi'$ (selected)

$$\frac{\Psi \vdash \sigma_\Psi : \Psi' \qquad \Psi \vdash T : K \cdot \sigma_\Psi}{\Psi \vdash (\sigma_\Psi,\ T) : (\Psi',\ K)}$$

Subst. lemmas:

$$\frac{\Psi;\ \Phi \vdash t : t' \qquad \Psi;\ \Phi' \vdash \sigma : \Phi}{\Psi;\ \Phi' \vdash t \cdot \sigma : t' \cdot \sigma} \qquad \frac{\Psi;\ \Phi' \vdash \sigma : \Phi \qquad \Psi;\ \Phi'' \vdash \sigma' : \Phi'}{\Psi;\ \Phi'' \vdash \sigma \cdot \sigma' : \Phi} \qquad \frac{\Psi \vdash T : K \qquad \Psi' \vdash \sigma_\Psi : \Psi}{\Psi' \vdash T \cdot \sigma_\Psi : K \cdot \sigma_\Psi}$$

(c) Substitutions over logical variables and extension variables

---

Syntax:

$$\Gamma ::= \bullet \mid \Gamma,\ x : \tau \mid \Gamma,\ x :_s \tau \mid \Gamma,\ \alpha : k \qquad e ::= \cdots \mid \mathsf{letstatic}\ x\ =\ e\ \mathsf{in}\ e'$$

Limit ctx:

$$\begin{array}{lcl}\bullet|_{\mathrm{static}} & = & \bullet \\ (\Gamma,\ x :_s t)|_{\mathrm{static}} & = & \Gamma|_{\mathrm{static}},\ x : t \\ (\Gamma,\ x : t)|_{\mathrm{static}} & = & \Gamma|_{\mathrm{static}} \\ (\Gamma,\ \alpha : k)|_{\mathrm{static}} & = & \Gamma|_{\mathrm{static}}\end{array}$$

$\Psi;\ \Sigma;\ \Gamma \vdash e : \tau$ (part)

$$\frac{\bullet;\ \Sigma;\ \Gamma|_{\mathrm{static}} \vdash e : \tau \qquad \Psi;\ \Sigma;\ \Gamma, x :_s \tau \vdash e' : \tau}{\Psi;\ \Sigma;\ \Gamma \vdash \mathsf{letstatic}\ x\ =\ e\ \mathsf{in}\ e' : \tau} \qquad \frac{x :_s \tau \in \Gamma}{\Psi;\ \Sigma;\ \Gamma \vdash x : \tau}$$

Evaluation:

$$\begin{array}{l}v ::= \Lambda(K).e_d \mid \mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ v \mid () \mid \lambda x : \tau.e_d \mid (v,\ v') \mid \mathsf{inj}_i\ v \mid \mathsf{fold}\ v \mid l \mid \Lambda\alpha : k.e_d \\ \mathcal{S} ::= \mathsf{letstatic}\ x\ =\ \bullet\ \mathsf{in}\ e' \mid \mathsf{letstatic}\ x\ =\ \mathcal{S}\ \mathsf{in}\ e' \mid \Lambda(K).\mathcal{S} \mid \lambda x : \tau.\mathcal{S} \mid \mathsf{unpack}\ e_d\ (.)x.(\mathcal{S}) \mid \mathsf{case}(e_d, x.\mathcal{S}, x.e_2) \\ \quad \mid \mathsf{case}(e_d, x.e_d, x.\mathcal{S}) \mid \Lambda\alpha : k.\mathcal{S} \mid \mathsf{fix}\ x : \tau.\mathcal{S} \mid \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi.T' \mapsto \mathcal{S}) \mid \mathcal{E}_s[\mathcal{S}] \\ \mathcal{E}_s ::= \mathcal{E}_s\ T \mid \mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ \mathcal{E}_s \mid \mathsf{unpack}\ \mathcal{E}_s\ (.)x.(e') \mid \mathcal{E}_s\ e' \mid e_d\ \mathcal{E}_s \mid (\mathcal{E}_s,\ e) \mid (e_d,\ \mathcal{E}_s) \mid \mathsf{proj}_i\ \mathcal{E}_s \mid \mathsf{inj}_i\ \mathcal{E}_s \\ \quad \mid \mathsf{case}(\mathcal{E}_s, x.e_1, x.e_2) \mid \mathsf{fold}\ \mathcal{E}_s \mid \mathsf{unfold}\ \mathcal{E}_s \mid \mathsf{ref}\ \mathcal{E}_s \mid \mathcal{E}_s := e' \mid e_d := \mathcal{E}_s \mid !\mathcal{E}_s \mid \mathcal{E}_s\ \tau \\ e_d ::= \text{all of } e \text{ except } \mathsf{letstatic}\ x\ =\ e\ \mathsf{in}\ e' \qquad \mathcal{E} ::= \text{exactly as } \mathcal{E}_s \text{ with } \mathcal{E}_s \to \mathcal{E} \text{ and } e \to e_d\end{array}$$

Stage 1 op.sem.:

$$\frac{(\ \mu,\ e_d\ ) \longrightarrow (\ \mu',\ e'_d\ )}{(\ \mu,\ \mathcal{S}[e_d]\ ) \longrightarrow_s (\ \mu',\ \mathcal{S}[e'_d]\ )} \qquad (\ \mu,\ \mathcal{S}[\mathsf{letstatic}\ x\ =\ v\ \mathsf{in}\ e]\ ) \longrightarrow_s (\ \mu,\ \mathcal{S}[e[v/x]]\ )$$

$$(\ \mu,\ \mathsf{letstatic}\ x\ =\ v\ \mathsf{in}\ e\ ) \longrightarrow_s (\ \mu,\ e[v/x]\ )$$

(d) Computational language: staging support

Figure 11. Main definitions in metatheory

a concrete context triggers a set of potentially complicated α-renamings, which a concrete representation makes explicit. We use a hybrid technique representing bound variables as deBruijn indices, and free variables as deBruijn levels. Our technique is a small departure from the named approach, requiring fewer extra annotations and lemmas than normal deBruijn indices. Also it identifies terms not only up to α-equivalence, but also up to extension of the context with new variables; this is why it is also used within the VeriML implementation.The two fundamental operations of this technique are freshening and binding, which are shown in Fig. 11a. Details can be found in section A of the appendix.

Extension variables. We extend the logic with support for meta-variables and context variables – we refer to both these sorts of variables as extension variables. A meta-variable X_i stands for a contextual term $T = [\Phi]t$, which packages a term together with the context it inhabits. Context variables ϕ_i stand for a context Φ, and are used to "weaken" parametric contexts in specific positions. Both kinds of variables are needed to support manipulation of open logical terms. Details of their definition and typing are shown in Fig. 11b. We use the same hybrid approach as above for representing these variables. A somewhat subtle aspect of this extension is that we generalize the deBruijn levels $\mathbf{I}$ used to index free variables, in order to deal effectively with parametric contexts.

Substitutions. The hybrid representation technique we use for variables renders simultaneous substitutions for all variables in scope as the most natural choice. In Fig. 11c, we show some example rules of how to apply a full simultaneous substitution σ to a term t, denoted as $t \cdot \sigma$. Similarly, we define full simultaneous substitutions σ_Ψ for extension contexts; defining their application has a very natural description, because of our variable representation technique. We prove a number of substitution lemmas which have simple statements, as shown in Fig. 11c. The proofs of these lemmas comprise the main effort required in proving the type-safety of a computational language such as the one we support, as they represent the point where computation specific to logical term manipulation takes place. Details can be found in section B of the appendix.

Computational language. We define an ML-style computational language that supports dependent functions and dependent pairs over contextual terms T, as well as pattern matching over them. Lack of space precludes us from including details here; full details can be found in section C of the appendix. A fairly complete ML calculus is supported, with mutable references and recursive types. Type safety is proved using standard techniques; its central point is extending the logic substitution lemmas to expressions and using them to prove progress and preservation of dependent functions and dependent pairs. This proof is modular with respect to the logic and other logics can easily be supported.

Pattern matching. Our metatheory includes many extensions in the pattern matching that is supported, as well as a new approach for dealing with typing patterns. We include support for pattern matching over contexts (e.g. to pick out hypotheses from the context) and for non-linear patterns. The allowed patterns are checked through a restriction of the usual typing rules $\Psi \vdash_p T : K$.

The essential idea behind our approach to pattern matching is to identify what the relevant variables in a typing derivation are. Since contexts are ordered, "removing" non-relevant variables amounts to replacing their definitions in the context with holes, which leads us to partial contexts $\widehat{\Psi}$. The corresponding notion of partial substitutions is denoted as $\widehat{\sigma_\Psi}$. Our main theorem about pattern matching can then be stated as:

Theorem 6.1 (Decidability of pattern matching) *If $\Psi \vdash_p T : K$, $\bullet \vdash_p T' : K$ and $\mathsf{relevant}(\Psi; \Phi \vdash T : K) = \widehat{\Psi}$, then either there exists a unique partial substitution $\widehat{\sigma_\Psi}$ such that $\bullet \vdash \widehat{\sigma_\Psi} : \widehat{\Psi}$ and $T \cdot \widehat{\sigma_\Psi} = T'$, or no such substitution exists.*

Details are found in section D of the appendix.

Staging. Our development in this paper critically depends on the letstatic construct we presented earlier. It can be seen as a dual of the traditional box construct of Davies and Pfenning [1996]. Details of its typing and semantics are shown in Fig. 11d. We define a notion of "static evaluation contexts" $\mathcal{S}$, which enclose a hole of the form $\mathsf{letstatic}\ x\ =\ \bullet\ \mathsf{in}\ e$. They include normal evaluation contexts, as well as evaluation contexts under binding structures. We evaluate expressions e that include staging constructs using the $\longrightarrow_s$ relation; internally, this uses the normal evaluation rules, that are used in the second stage as well, for evaluating expressions which do not include other staging constructs. If stage-one evaluation is successful, we are left with a residual dynamic configuration (μ', e_d) which is then evaluated normally. We prove type-safety for stage-one evaluation; its statement follows.

Theorem 6.2 (Stage-one Type Safety) *If* $\bullet; \Sigma; \bullet \vdash e : \tau$ *then: either* e *is a dynamic expression* e_d*; or, for every store* μ *such that* $\vdash \mu : \Sigma$*, we have: either* $\mu, e \longrightarrow_s \mathsf{error}$*, or, there exists an* e'*, a new store typing* $\Sigma' \supseteq \Sigma$ *and a new store* μ' *such that:* $(\mu, e) \longrightarrow (\mu', e')$*;* $\vdash \mu' : \Sigma'$*; and* $\bullet; \Sigma'; \bullet \vdash e' : \tau$.

Details are found in section E of the appendix.

Collapsing extension variables. Last, we have proved the fact that under the conditions described in Sec. 5, it is possible to collapse a term t into a term t' which is typed under the empty extension variables context; a substitution σ with which we can regain the original term t exists. This suggests that whenever a proof object t for a specific proposition is required, an equivalent proof object that does not mention uninstantiated extension variables exists. Therefore, we can write an equivalent proof script producing the collapsed proof object instead, and evaluate that script statically. The statement of this theorem is the following:

Theorem 6.3 *If* $\Psi \vdash [\Phi]t : [\Phi]t_T$ *and* $\mathsf{collapsible}(\Psi \vdash [\Phi]t : [\Phi]t_T)$*, then there exist* Φ'*,* t'*,* t'_T *and* σ *such that* $\bullet \vdash \Phi'$ *wf,* $\bullet \vdash [\Phi']t' : [\Phi']t'_T$*,* $\Psi; \Phi \vdash \sigma : \Phi'$*,* $t' \cdot \sigma = t$ *and* $t'_T \cdot \sigma = t_T$.

The main idea behind the proof is to maintain a number of substitutions and their inverses: one to go from a general Ψ extension context into an "equivalent" Ψ' context, which includes only definitions of the form $[\Phi]t$, for a constant Φ context that uses no extension variables. Then, another substitution and its inverse are maintained to go from that extension variables context into the empty one; this is simpler, since terms typed under Ψ' are already essentially free of metavariables. The computational content within the proof amounts to a procedure for transforming proof scripts inside tactics into static proof scripts. Details are found in section F of the appendix.

7. Implementation

We have completed a prototype implementation of the VeriML language, as described in this paper, that supports all of our claims. We have built on our existing prototype [Stampoulis and Shao 2010] and have added an extensive set of new features and improvements. The prototype is written in OCaml and is about 6k lines of code. Using the prototype we have implemented a number of examples, that are about 1.5k lines of code. Readers are encouraged to download and try the prototype from `http://flint.cs.yale.edu/publications/supc.html`.

New features. We have implemented the new features we have described so far: context matching, non-linear patterns, proof-erasure semantics, staging, and inferencing for logical and computational terms. Proof-erasure semantics are utilized only if requested by a per-function flag, enabling us to selectively "trust" tactics. The staging construct we support is more akin to the $\langle\cdot\rangle_{\text{static}}$ form described as syntactic sugar in Sec. 5, and it is able to infer the collapsing substitutions that are needed, following the approach used in our metatheory.

Changes. We have also changed quite a number of things in the prototype and improved many of its aspects. A central change, mediated by our new treatment of the conversion rule, was to modify the used logic in

order to use the explicit equality approach; the existing prototype used the λHOL_c logic. We also switched the variable representation to the hybrid deBruijn levels-deBruijn indices technique we described, which enabled us to implement subtyping based on context subsumption. Also, we have adapted the typing rules of the pattern matching construct in order to support refining the environment based on the current branch.

Examples implemented. We have implemented a number of examples to support our claims. First, we have written the type-safe conversion check routine for $\beta\mathbb{N}$, and extended it to support congruence closure based on equalities in the context. Proofs of this latter tactic are constructed automatically through static proof scripts, using a naive rewriter that is non-terminating in the general case. We have also completed proofs for theorems of arithmetic for the properties of addition and multiplication, and used them to write an arithmetic simplification tactic. All of the theorems are proved by making essential use of existing conversion rules, and are immediately added into new conversion rules, leading to a compact and clean development style. The resulting code does not need to make use of translation validation or proof by reflection, which are typically used to implement similar tactics in existing proof assistants.

Towards a practical proof assistant. In order to facilitate practical proof and program construction in VeriML, we introduced some features to support surface syntax, enabling users to omit most details about the environments of contextual terms and the substitutions used with meta-variables. This syntax follows the style of Delphin, assuming an ambient logical variable environment which is extended through a construct denoted as $\nu x : t.e$. Still, the full power of contextual modal type theory is available, which is crucial in order to change what the current ambient environment is, used, as we saw earlier, for static calls to tactics. In general the surface syntax leads to much more concise and readable code.

Last, we introduced syntax support for calls to tactics, enabling users to write proof expressions that look very similar to proof scripts in current proof assistants. We developed a rudimentary ProofGeneral mode for VeriML, that enables us to call the VeriML type-checker and interpreter for parts of source files. By adding holes to our sources, we can be informed by the type inference mechanism about their expected types. Those types correspond to what the current "proof state" is at that point. Therefore, a possible workflow for developing tactics or proofs, is writing the known parts, inserting holes in missing points to know what remains to be proved, and calling the typechecker to get the proof state information. This workflow corresponds closely to the interactive proof development support in proof assistants like Coq and Isabelle, but generalizes it to the case of tactics as well.

8. Related work

There is a large body of work that is related to the ideas we have presented here.

Techniques for robust proof development. There have been multiple proposals for making proof development inside existing proof assistants more robust. A well-known technique is *proof-by-reflection* [Boutin 1997]: writing total and certified decision procedures within the functional language contained in a logic like CIC. A recently introduced technique is *automation through canonical structures* [Gonthier et al. 2011]: the resolution mechanism for finding instances of canonical structures (a generalization of type classes) is cleverly utilized in order to program automation procedures for specific classes of propositions. We view both approaches as somewhat similar, as both are based in cleverly exploiting static "interpreters" that are available in a modern proof assistant: the partial evaluator within the conversion rule in the former case; the unification algorithm within instance discovery in the latter case.

Our approach can thus be seen as similar, but also as a generalization of these approaches, since a general-purpose programming model is supported. Therefore, users do not have to adapt to a specific programming style for writing automation code, but can rather use a familiar functional language. Proof-by-reflection could perhaps be used to support the same kind of extensions to the conversion rule; still, this would require reflecting

a large part of the logic in itself, through a prohibitively complicated encoding. Both techniques are applicable to our setting as well and could be used to provide benefits to large developments within our language.

The style advocated in Chlipala [2011] (and elsewhere) suggests that proper proof engineering entails developing sophisticated automation tactics in a modular style, and extending their power by adding proved lemmas as hints. We are largely inspired by this approach, and believe that our introduction of the extensible conversion rule and static checking of tactics can significantly benefit it. We demonstrate similar ideas in layering conversion tactics.

Traditional proof assistants. There are many parallels of our work with the *LCF family of proof assistants*, like HOL4 [Slind and Norrish 2008] and HOL-Light [Harrison 1996], which have served as inspiration. First, the foundational logic that we use is similar. Also, our use of a dedicated ML-like programming language to program tactics and proof scripts is similar to the approach taken by HOL4 and HOL-Light. Last, the fact that no proof objects need to be generated is shared. Still, checking a proof script in HOL requires evaluating it fully. Using our approach, we can selectively evaluate parts of proof scripts; we focus on conversion-like tactics, but we are not limited inherrently to those. This is only possible because our proof scripts carry proof state information within their types. Similarly, proof scripts contained within LCF tactics cannot be evaluated statically, so it is impossible to establish their validity upon tactic definition. It is possible to do a transformation similar to ours manually (lifting proof scripts into auxiliary lemmas that are proved prior to the tactic), but the lack of type information means that many more details need to be provided.

The Coq proof assistant [Barras et al. 2010] is another obvious point of reference for our work. We will focus on the conversion rule that CIC, its accompanying logic, supports – the same problems with respect to proof scripts and tactics that we described in the LCF case also apply for Coq. The conversion rule, which identifies computationally equivalent propositions, coupled with the rich type universe available, opens up many possibilities for constructing small and efficiently checkable proof objects. The implementation of the conversion rule needs to be part of the trusted base of the proof assistant. Also, the fact that the conversion check is built-in to the proof assistant makes the supported equivalence rigid and non-extensible by frequently used decision procedures.

There is a large body of work that aims to extend the conversion rule to arbitrary confluent rewrite systems (e.g. Blanqui et al. [1999]) and to include decision procedures [Strub 2010]. These approaches assume some small or larger addition to the trusted base, and extend the already complex metatheory of Coq. Furthermore, the NuPRL proof assistant [Constable et al. 1986] is based on extensional type theory which includes an extensional conversion rule. This enables complex decision procedures to be part of conversion; but it results in a very large trusted base. We show how, for a subset of these type theories, the conversion check can be recovered outside the trusted base. It can be extended with arbitrarily complex new tactics, written in a familiar programming style, without any metatheoretic additions and without hurting the soundness of the logic. The question of whether these type theories can be supported in full remains as future work, but as far as we know, there is no inherrent limitation to our approach.

Dependently-typed programming. The large body of work on dependently-typed languages has close parallels to our work. Out of the multitude of proposals, we consider the Russell framework [Sozeau 2006] as the current state-of-the-art, because of its high expressivity and automation in discharging proof obligations. In our setting, we can view dependently-typed programming as a specific case of tactics producing complex data types that include proof objects. Static proof scripts can be leveraged to support expressivity similar to the Russell framework. Furthermore, our approach opens up a new intriguing possibility: dependently-typed programs whose obligations are discharged statically and automatically, through code written within the same language.

Last, we have been largely inspired by the work on languages like Beluga [Pientka and Dunfield 2008] and Delphin [Poswolsky and Schürmann 2008], and build on our previous work on VeriML [Stampoulis and Shao

2010]. We investigate how to leverage type-safe tactics, as well as a number of new constructs we introduce, so as to offer an extensible notion of proof checking. Also, we address the issue of statically checking the proof scripts contained within tactics written in VeriML. As far as we know, our development is the first time languages such as these have been demonstrated to provide a workflow similar to interactive proof assistants.

Acknowledgments

We thank anonymous referees for their suggestions and comments on an earlier version of this paper. This research is based on work supported in part by DARPA CRASH grant FA8750-10-2-0254 and NSF grants CCF-0811665, CNS-0910670, and CNS 1065451. Any opinions, findings, and conclusions contained in this document are those of the authors and do not reflect the views of these agencies.

References

H.P. Barendregt and H. Geuvers. Proof-assistants using dependent type systems. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Sci. Pub. B.V., 1999.

B. Barras, S. Boutin, C. Cornes, J. Courant, Y. Coscoy, D. Delahaye, D. de Rauglaudre, J.C. Filliâtre, E. Giménez, H. Herbelin, et al. The Coq proof assistant reference manual (version 8.3), 2010.

F. Blanqui, J.P. Jouannaud, and M. Okada. The calculus of algebraic constructions. In *Rewriting Techniques and Applications*, pages 671–671. Springer, 1999.

F. Blanqui, J.P. Jouannaud, and P.Y. Strub. A calculus of congruent constructions. *Unpublished draft*, 2005.

S. Boutin. Using reflection to build efficient and certified decision procedures. *Lecture Notes in Computer Science*, 1281: 515–529, 1997.

A. Chlipala. Mostly-automated verification of low-level programs in computational separation logic. In *Proceedings of the 2011 ACM SIGPLAN conference on Programming Language Design and Implementation*. ACM, 2011.

R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P. Panangaden, et al. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, NJ, 1986.

R. Davies and F. Pfenning. A modal analysis of staged computation. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 258–270. ACM, 1996.

G. Gonthier. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

G. Gonthier, B. Ziliani, A. Nanevski, and D. Dreyer. How to make ad hoc proof automation less ad hoc. In *Proceeding of the 16th ACM SIGPLAN International Conference on Functional Programming*, pages 163–175. ACM, 2011.

J. Harrison. HOL Light: A tutorial introduction. *Lecture Notes in Computer Science*, pages 265–269, 1996.

G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, et al. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, pages 207–220. ACM, 2009.

X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

T. Nipkow, L.C. Paulson, and M. Wenzel. Isabelle/HOL : A Proof Assistant for Higher-Order Logic, volume 2283 of LNCS, 2002.

B. Pientka and J. Dunfield. Programming with proofs and explicit contexts. In *Proceedings of the 10th international ACM SIGPLAN conference on Principles and Practice of Declarative Programming*, pages 163–173. ACM New York, NY, USA, 2008.

A. Poswolsky and C. Schürmann. Practical programming with higher-order encodings and dependent types. *Lecture Notes in Computer Science*, 4960:93, 2008.

V. Siles and H. Herbelin. Equality is typable in semi-full pure type systems. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 21–30. IEEE, 2010.

K. Slind and M. Norrish. A brief overview of HOL4. *Theorem Proving in Higher Order Logics*, pages 28–32, 2008.

M. Sozeau. Subset coercions in coq. In *Proceedings of the 2006 International Conference on Types for Proofs and Programs*, pages 237–252. Springer-Verlag, 2006.

A. Stampoulis and Z. Shao. VeriML: Typed computation of logical terms inside a language with effects. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming*, pages 333–344. ACM, 2010.

P.Y. Strub. Coq modulo theory. In *Proceedings of the 24th International Conference on Computer Science Logic*, pages 529–543. Springer-Verlag, 2010.

Appendices

A. The logic λHOL_c

Definition A.1 (Syntax of the language) *The syntax of the logic language is given below.*

$$
\begin{array}{rl}
t ::= & s \mid c \mid f_i \mid b_i \mid \lambda(t_1).t_2 \mid t_1\ t_2 \mid \Pi(t_1).t_2 \mid t_1 = t_2 \mid \mathsf{conv}\ t\ t \mid \mathsf{refl}\ t \mid \mathsf{symm}\ t \mid \mathsf{trans}\ t_1\ t_2 \mid \mathsf{congapp}\ t_1\ t_2 \\
& \mid \mathsf{congimpl}\ t_1\ t_2 \mid \mathsf{conglam}\ t \mid \mathsf{congpi}\ t \mid \mathsf{beta}\ t_1\ t_2 \\
s ::= & \mathsf{Prop} \mid \mathsf{Type} \mid \mathsf{Type}' \\
\Phi ::= & \bullet \mid \Phi,\ t \\
\sigma ::= & \bullet \mid t \\
\Sigma ::= & \bullet \mid \Sigma,\ c : t
\end{array}
$$

We use f_i to denote the i-th free variable in the current environment and b_i for the bound variable with deBruijn index i. The benefit of this approach is that the representation of terms is unique both up to α-equivalence and up to extensions of the current free variables context.

Definition A.2 (Context length and access) *Getting the length of a context, and an element out of a context, are defined as follows. In the case of element access, we assume that $i < |\Phi|$.*

$\boxed{|\Phi|}$

$$
\begin{array}{lcl}
|\bullet| & = & 0 \\
|\Phi,t| & = & |\Phi| + 1
\end{array}
$$

$\boxed{\Phi.i}$

$$
\begin{array}{lcl}
(\Phi,t).|\Phi| & = & t \\
(\Phi,t).i & = & \Phi.i
\end{array}
$$

Definition A.3 (Substitution length) *Getting the length of a substitution is defined as follows.*

$$
\begin{array}{lcl}
|\bullet| & = & 0 \\
|\sigma,t| & = & |\sigma| + 1
\end{array}
$$

Definition A.4 (Substitution access) *The operation of accessing the i-th term out of a substitution is defined as follows. We assume that $i < |\sigma|$.*

$$
\begin{array}{lcl}
(\sigma,t).|\sigma| & = & t \\
(\sigma,t).i & = & \sigma.i
\end{array}
$$

Definition A.5 (Substitution application) *The operation of applying a substitution is defined as follows.*

$\boxed{t \cdot \sigma}$

$$
\begin{array}{lcl}
s \cdot \sigma & = & s \\
c \cdot \sigma & = & c \\
f_i \cdot \sigma & = & \sigma.i \\
b_i \cdot \sigma & = & b_i \\
(\lambda(t_1).t_2) \cdot \sigma & = & \lambda(t_1 \cdot \sigma).(t_2 \cdot \sigma) \\
(t_1\ t_2) \cdot \sigma & = & (t_1 \cdot \sigma)\ (t_2 \cdot \sigma) \\
(\Pi(t_1).t_2) \cdot \sigma & = & \Pi(t_1 \cdot \sigma).(t_2 \cdot \sigma) \\
(t_1 = t_2) \cdot \sigma & = & (t_1 \cdot \sigma) = (t_2 \cdot \sigma) \\
(\mathsf{conv}\ t_1\ t_2) \cdot \sigma & = & \mathsf{conv}\ (t_1 \cdot \sigma)\ (t_2 \cdot \sigma) \\
(\mathsf{refl}\ t) \cdot \sigma & = & \mathsf{refl}\ (t \cdot \sigma) \\
(\mathsf{symm}\ t) \cdot \sigma & = & \mathsf{symm}\ (t \cdot \sigma) \\
(\mathsf{trans}\ t_1\ t_2) \cdot \sigma & = & \mathsf{trans}\ (t_1 \cdot \sigma)\ (t_2 \cdot \sigma) \\
(\mathsf{congapp}\ t_1\ t_2) \cdot \sigma & = & \mathsf{congapp}\ (t_1 \cdot \sigma)\ (t_2 \cdot \sigma) \\
(\mathsf{congimpl}\ t_1\ t_2) \cdot \sigma & = & \mathsf{congimpl}\ (t_1 \cdot \sigma)\ (t_2 \cdot \sigma) \\
(\mathsf{conglam}\ t) \cdot \sigma & = & \mathsf{conglam}\ (t \cdot \sigma) \\
(\mathsf{congpi}\ t) \cdot \sigma & = & \mathsf{congpi}\ (t \cdot \sigma) \\
(\mathsf{beta}\ t_1\ t_2) \cdot \sigma & = & \mathsf{beta}\ (t_1 \cdot \sigma)\ (t_2 \cdot \sigma)
\end{array}
$$

$\boxed{\sigma' \cdot \sigma}$

$$
\begin{array}{lcl}
\bullet \cdot \sigma & = & \bullet \\
(\sigma', t) \cdot \sigma & = & (\sigma' \cdot \sigma), (t \cdot \sigma)
\end{array}
$$

Definition A.6 (Identity substitution) *The identity substitution is defined as follows.*

$$
\begin{array}{lcl}
\mathsf{id}_\bullet & = & \bullet \\
\mathsf{id}_{\Phi,\, t} & = & \mathsf{id}_\Phi,\ f_{|\Phi|}
\end{array}
$$

Definition A.7 *Free and bound variable limits for terms are defined as follows.*

$\boxed{t <^f n}$

$$
\begin{array}{lcl}
s <^f n & & \\
c <^f n & & \\
f_i <^f n & \Leftarrow & n > i \\
b_i <^f n & & \\
(\lambda(t_1).t_2) <^f n & \Leftarrow & t_1 <^f n \wedge t_2 <^f n \\
t_1\ t_2 <^f n & \Leftarrow & t_1 <^f n \wedge t_2 <^f n \\
 & \cdots &
\end{array}
$$

$\boxed{t <^b n}$

$$
\begin{array}{lcl}
s <^b n & & \\
c <^b n & & \\
f_i <^b n & & \\
b_i <^b n & \Leftarrow & n > i \\
(\lambda(t_1).t_2) <^b n & \Leftarrow & t_1 <^b n \wedge t_2 <^b n + 1 \\
t_1\ t_2 <^b n & \Leftarrow & t_1 <^b n \wedge t_2 <^b n \\
 & \cdots &
\end{array}
$$

Definition A.8 *Free and bound variable limits for substitutions are defined as follows.*

$\boxed{\sigma <^f n}$

$$\begin{array}{l} \bullet <^f n \\ (\sigma, t) <^f n \quad \Leftarrow \quad \sigma <^f n \wedge t <^f n \end{array}$$

$\boxed{\sigma <^b n}$

$$\begin{array}{l} \bullet <^b n \\ (\sigma, t) <^b n \quad \Leftarrow \quad \sigma <^b n \wedge t <^b n \end{array}$$

Definition A.9 (Freshening) *Freshening a term is defined as follows. We assume that $t <^f m$ and $t <^b n+1$.*

$\boxed{\lceil t \rceil^n_m}$

$$\begin{array}{lcl}
\lceil s \rceil & = & s \\
\lceil c \rceil & = & c \\
\lceil f_i \rceil & = & f_i \\
\lceil b_n \rceil^n_m & = & f_m \\
\lceil b_i \rceil^n & = & b_i \text{ when } i < n \\
\lceil (\lambda(t_1).t_2) \rceil^n & = & \lambda(\lceil t_1 \rceil^n).\lceil t_2 \rceil^{n+1} \\
\lceil t_1\ t_2 \rceil & = & \lceil t_1 \rceil\ \lceil t_2 \rceil \\
\lceil \Pi(t_1).t_2) \rceil^n & = & \Pi(\lceil t_1 \rceil^n).(\lceil t_2 \rceil^{n+1}) \\
\lceil t_1 = t_2 \rceil & = & \lceil t_1 \rceil = \lceil t_2 \rceil \\
\lceil \mathsf{conv}\ t_1\ t_2 \rceil & = & \mathsf{conv}\ \lceil t_1 \rceil\ \lceil t_2 \rceil \\
\lceil \mathsf{refl}\ t \rceil & = & \mathsf{refl}\ \lceil t \rceil \\
\lceil \mathsf{symm}\ t \rceil & = & \mathsf{symm}\ \lceil t \rceil \\
\lceil \mathsf{trans}\ t_1\ t_2 \rceil & = & \mathsf{trans}\ \lceil t_1 \rceil\ \lceil t_2 \rceil \\
\lceil \mathsf{congapp}\ t_1\ t_2 \rceil & = & \mathsf{congapp}\ \lceil t_1 \rceil\ \lceil t_2 \rceil \\
\lceil \mathsf{congimpl}\ t_1\ t_2 \rceil & = & \mathsf{congimpl}\ \lceil t_1 \rceil\ \lceil t_2 \rceil \\
\lceil \mathsf{conglam}\ t \rceil & = & \mathsf{conglam}\ \lceil t \rceil \\
\lceil \mathsf{congpi}\ t \rceil & = & \mathsf{congpi}\ \lceil t \rceil \\
\lceil \mathsf{beta}\ t_1\ t_2 \rceil & = & \mathsf{beta}\ \lceil t_1 \rceil\ \lceil t_2 \rceil
\end{array}$$

Definition A.10 (Binding) *Binding a term is defined as follows. We assume that $t <^f m$ and $t <^b n$.*

$\boxed{\lfloor t \rfloor_m^n}$

$$
\begin{array}{lcl}
\lfloor s \rfloor & = & s \\
\lfloor c \rfloor & = & c \\
\lfloor f_{m-1} \rfloor_m^n & = & b_n \\
\lfloor f_i \rfloor_m^n & = & f_i \text{ when } i < m-1 \\
\lfloor b_i \rfloor & = & b_i \\
\lfloor (\lambda(t_1).t_2) \rfloor & = & \lambda(\lfloor t_1 \rfloor^n).\, \lfloor t_2 \rfloor^{n+1} \\
\lfloor t_1\ t_2 \rfloor & = & \lfloor t_1 \rfloor^n\ \lfloor t_2 \rfloor^n \\
\lfloor \Pi(t_1).t_2) \rfloor & = & \Pi(\lfloor t_1 \rfloor^n).\, \lfloor t_2 \rfloor^{n+1} \\
\lfloor t_1 = t_2 \rfloor & = & \lfloor t_1 \rfloor = \lfloor t_2 \rfloor \\
\lfloor \mathsf{conv}\ t_1\ t_2 \rfloor & = & \mathsf{conv}\ \lfloor t_1 \rfloor\ \lfloor t_2 \rfloor \\
\lfloor \mathsf{refl}\ t \rfloor & = & \mathsf{refl}\ \lfloor t \rfloor \\
\lfloor \mathsf{symm}\ t \rfloor & = & \mathsf{symm}\ \lfloor t \rfloor \\
\lfloor \mathsf{trans}\ t_1\ t_2 \rfloor & = & \mathsf{trans}\ \lfloor t_1 \rfloor\ \lfloor t_2 \rfloor \\
\lfloor \mathsf{congapp}\ t_1\ t_2 \rfloor & = & \mathsf{congapp}\ \lfloor t_1 \rfloor\ \lfloor t_2 \rfloor \\
\lfloor \mathsf{congimpl}\ t_1\ t_2 \rfloor & = & \mathsf{congimpl}\ \lfloor t_1 \rfloor\ \lfloor t_2 \rfloor \\
\lfloor \mathsf{conglam}\ t \rfloor & = & \mathsf{conglam}\ \lfloor t \rfloor \\
\lfloor \mathsf{congpi}\ t \rfloor & = & \mathsf{congpi}\ \lfloor t \rfloor \\
\lfloor \mathsf{beta}\ t_1\ t_2 \rfloor & = & \mathsf{beta}\ \lfloor t_1 \rfloor\ \lfloor t_2 \rfloor
\end{array}
$$

Definition A.11 (Typing) *The typing rules are defined as follows.*

$\boxed{\vdash \Sigma\ \mathrm{wf}}$

$$\vdash \bullet\ \mathrm{wf} \qquad \frac{\vdash \Sigma\ \mathrm{wf} \qquad \bullet \vdash_\Sigma t : s \qquad (c : \_) \notin \Sigma}{\vdash \Sigma,\, c : t\ \mathrm{wf}}$$

$\boxed{\vdash_\Sigma \Phi\ \mathrm{wf}}$

$$\vdash \bullet\ \mathrm{wf} \qquad \frac{\vdash \Phi\ \mathrm{wf} \qquad \Phi \vdash t : s}{\vdash \Phi,\, t\ \mathrm{wf}}$$

$\boxed{\Phi \vdash_\Sigma t : t'}$

$$\frac{c : t \in \Sigma}{\Phi \vdash_\Sigma c : t} \qquad \frac{\Phi.i = t}{\Phi \vdash f_i : t} \qquad \frac{(s,s') \in \mathcal{A}}{\Phi \vdash s : s'} \qquad \frac{\Phi \vdash t_1 : s \qquad \Phi,\, t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : s' \qquad (s,s',s'') \in \mathcal{R}}{\Phi \vdash \Pi(t_1).t_2 : s''}$$

$$\frac{\Phi \vdash t_1 : s \qquad \Phi,\, t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : t' \qquad \Phi \vdash \Pi(t_1).\, \lfloor t' \rfloor_{|\Phi|+1} : s'}{\Phi \vdash \lambda(t_1).t_2 : \Pi(t_1).\, \lfloor t' \rfloor_{|\Phi|+1}} \qquad \frac{\Phi \vdash t_1 : \Pi(t).t' \qquad \Phi \vdash t_2 : t}{\Phi \vdash t_1\ t_2 : \lceil t' \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2)}$$

$$\frac{\Phi \vdash t_1 : t \qquad \Phi \vdash t_2 : t \qquad \Phi \vdash t : \mathsf{Type}}{\Phi \vdash t_1 = t_2 : \mathsf{Prop}}$$

$$\frac{\Phi \vdash t : t_1 \qquad \Phi \vdash t_1 : \mathsf{Prop} \qquad \Phi \vdash t' : t_1 = t_2}{\Phi \vdash \mathsf{conv}\ t\ t' : t_2} \qquad \frac{\Phi \vdash t_1 : t \qquad \Phi \vdash t_1 = t_1 : \mathsf{Prop}}{\Phi \vdash \mathsf{refl}\ t_1 : t_1 = t_1} \qquad \frac{\Phi \vdash t_a : t_1 = t_2}{\Phi \vdash \mathsf{symm}\ t_a : t_2 = t_1}$$

$$\frac{\Phi \vdash t_a : t_1 = t_2 \qquad \Phi \vdash t_b : t_2 = t_3}{\Phi \vdash \mathsf{trans}\ t_a\ t_b : t_1 = t_3}$$

$$\frac{\Phi \vdash t_a : M_1 = M_2 \qquad \Phi \vdash M_1 : A \to B \qquad \Phi \vdash t_b : N_1 = N_2 \qquad \Phi \vdash N_1 : A}{\Phi \vdash \mathsf{congapp}\ t_a\ t_b : M_1\ N_1 = M_2\ N_2}$$

$$\frac{\Phi \vdash t_a : A_1 = A_2 \qquad \Phi, A_1 \vdash \lceil t_b \rceil : B_1 = B_2 \qquad \Phi \vdash A_1 : \mathsf{Prop} \qquad \Phi, A_1 \vdash \lceil B_1 \rceil : \mathsf{Prop}}{\Phi \vdash \mathsf{congimpl}\ t_a\ (\lambda(A_1).t_b) : \Pi(A_1).\lfloor B_1 \rfloor = \Pi(A_2).\lfloor B_2 \rfloor}$$

$$\frac{\Phi,\ A \vdash \lceil t_b \rceil : B = B' \qquad \Phi \vdash \Pi(A).\lfloor B \rfloor = \Pi(A).\lfloor B' \rfloor : \mathsf{Prop}}{\Phi \vdash \mathsf{congpi}\ (\lambda(A).t_b) : \Pi(A).\lfloor B \rfloor = \Pi(A).\lfloor B' \rfloor}$$

$$\frac{\Phi,\ A \vdash \lceil t_b \rceil : B_1 = B_2 \qquad \Phi \vdash \lambda(A).\lfloor B_1 \rfloor = \lambda(A).\lfloor B_2 \rfloor : \mathsf{Prop}}{\Phi \vdash \mathsf{conglam}\ (\lambda(A).t_b) : \lambda(A).\lfloor B_1 \rfloor = \lambda(A).\lfloor B_2 \rfloor}$$

$$\frac{\Phi \vdash \lambda(A).M : A \to B \qquad \Phi \vdash N : A \qquad \Phi \vdash A \to B : \mathsf{Type}}{\Phi \vdash \mathsf{beta}\ (\lambda(A).M)\ N : (\lambda(A).M)\ N = \lceil M \rceil \cdot (\mathsf{id}_\Phi, N)}$$

$\boxed{\Phi \vdash \sigma : \Phi'}$

$$\frac{\vdash \Phi\ \mathrm{wf}}{\Phi \vdash \bullet : \bullet} \qquad \frac{\Phi \vdash \sigma : \Phi' \qquad \Phi \vdash t : t' \cdot \sigma}{\Phi \vdash \sigma, t : (\Phi', t')}$$

Lemma A.12 *If $t <^f m$ and $|\Phi| = m$ then $t \cdot id_\Phi = t$.*

Trivial by induction on $t <^f m$. The interesting case is $f_i \cdot \mathsf{id}_\Phi = f_i$. This is simple to prove by induction on Φ.

Lemma A.13 *If $\sigma <^f m$ then $\sigma \cdot id_m = \sigma$.*

By induction on σ and use of lemma A.12.

Lemma A.14 *If $\Phi \vdash t : t'$ then $t <^f |\Phi|$ and $t <^b 0$.*

Trivial by induction on the typing derivation $\Phi \vdash t : t'$ (and use of implicit assumptions for $\lceil t \rceil$).

Lemma A.15 *If $\vdash \Phi$ wf then for any Φ' and $t_{1\cdots n}$ such that $\Phi' = \Phi,\ t_1,\ t_2, \cdots, t_n$ and $\vdash \Phi'$ wf, we have that $\Phi' \vdash id_\Phi : \Phi$.*

By induction on Φ.
In case $\Phi = \bullet$, trivial.
In case $\Phi = \Phi'',\ t'$, then by induction hypothesis we have for all proper extensions of Φ'' $\Phi'', t_1, \cdots, t_n \vdash \mathsf{id}_{\Phi''} : \Phi''$.

We now need to prove that for all proper extensions of Φ'', t' we have Φ'', t', t_1, $\cdots$, $t_n \vdash \mathsf{id}_{\Phi'', t'} : (\Phi'', t')$.
From the inductive hypothesis we get that Φ'', t', t_1, $\cdots$, $t_n \vdash \mathsf{id}_{\Phi''} : \Phi''$. We also have that $\Phi'' \vdash t' : s$ by inversion of the well-formedness of Φ.
Thus by A.14, we get that $t' <^f |\Phi''|$.
Furthermore by A.12 we get that $t' \cdot \mathsf{id}_{\Phi''} = t'$.
Thus we have $\Phi'', t', t_1, \cdots, t_n \vdash f_{|\Phi''|} : t' \cdot \mathsf{id}_{\Phi''}$.
Thus by applying the appropriate substitution typing rule, we get that $\Phi'', t', t_1, \cdots, t_n \vdash (\mathsf{id}_{\Phi''}, f_{|\Phi''|}) : (\Phi'', t')$, which is exactly the desired result.

Lemma A.16 *If* $\Phi \vdash \sigma : \Phi'$ *then* $\sigma <^f |\Phi|$, $\sigma <^b 0$ *and* $|\sigma| = |\Phi'|$.

Trivial by induction on the typing derivation for σ, and use of lemma A.14.

Lemma A.17 *If* $\vdash \Phi$ *wf and* $|\Phi| = n$ *then for all* $i < n$, $\Phi.i <^f i$.

Trivial by induction on the well-formedness derivation for Φ and use of lemma A.14.

Lemma A.18 *If* $t <^f m$, $|\sigma| = m$ *and* $t \cdot \sigma = t'$ *then* $t \cdot (\sigma, t_1, t_2, \cdots, t_n) = t'$.

Trivial by induction on $t <^f m$.

Lemma A.19 *If* $\sigma <^f m$, $|\sigma'| = m$ *and* $\sigma \cdot \sigma' = \sigma_r$ *then* $\sigma \cdot (\sigma', t_1, t_2, \cdots, t_n) = \sigma_r$.

Trivial by induction on σ, and use of the lemma A.18.

Lemma A.20 *If* $\vdash \Phi$ *wf,* $\Phi.i = t$ *and* $\Phi' \vdash \sigma : \Phi$, *then* $\Phi' \vdash \sigma.i : t \cdot \sigma$.

Induction on the derivation of typing for σ.
In the case where $\sigma = \bullet$, the (implicit) assumption that $i < |\Phi|$ obviously does not hold, so the case is impossible.
In the case where $\sigma = \sigma', t'$, we split cases on whether $i = |\Phi| - 1$ or not.
If it is, then the typing rule gives us the desired directly.
If it is not, the inductive hypothesis gives us the result $\Phi' \vdash \sigma'.i : t \cdot \sigma'$. Now from lemma A.17 we have that $\Phi.i <^f i$. We can now apply lemma A.18 to get $t \cdot \sigma' = t \cdot (\sigma', t') = t \cdot \sigma$, proving the desired.

Lemma A.21 *If* $t <^f m$, $t <^b n+1$, $\sigma <^f m'$ *and* $|\sigma| = m$ *then* $\lceil t \cdot \sigma \rceil^n_{m'} = \lceil t \rceil^n_m \cdot (\sigma, f_{m'})$.

By structural induction on t.
Cases $t = s$ and $t = c$ are trivial.
When $t = f_i$, we have $i < m$ thus both sides will be equal to $\sigma.i$.
When $t = b_i$, we split cases on whether $i = n$ or $i < n$.
If $i = n$, then the left-hand side becomes $\lceil b_n \cdot \sigma \rceil^n_{m'} = \lceil b_n \rceil^n_{m'} = f_{m'}$.
The right-hand side becomes $\lceil b_n \rceil^n_m \cdot (\sigma, f_{m'}) = f_m \cdot (\sigma, f_{m'}) = f_{m'}$.
When $i < n$ it is trivial to see that both sides are equal to bi.
In the case where $t = \lambda(t_1).(t_2)$, we prove the result trivially using the induction hypothesis.
The subtlety for t_2 is that the inductive hypothesis is applied for $n = n+1$, which is possible because from the definition of $\cdot <^b \cdot$ we have that $t_2 <^b (n+1)+1$.

Lemma A.22 *If* $t <^f m+1$, $t <^b n$, $\sigma <^f m'$ *and* $|\sigma| = m$ *then* $\lfloor t \cdot (\sigma, f_{m'}) \rfloor^n_{m'+1} = \lfloor t \rfloor^n_{m+1} \cdot \sigma$.

By structural induction on t. Cases $t = s$ and $t = c$ are trivial. When $t = f_i$, we split cases on whether $i = m$ or $i < m$. If $i = m$, then the left hand side becomes: $\lfloor f_m \cdot (\sigma, f_{m'}) \rfloor_{m'+1}^{n} = \lfloor f_{m'} \rfloor_{m'+1}^{n} = b_n$. The right hand side becomes: $\lfloor f_m \rfloor_{m+1}^{n} \cdot \sigma = b_n \cdot \sigma = b_n$. In case $i < m$, both sides are trivially equal to $\sigma.i$. When $t = b_i$, both sides are trivially equal to b_i. When $t = \lambda(t_1).t_2$, the result follows directly from the inductive hypothesis for t_1 and t_2, and the definitions of $\cdot$ and $\lfloor \cdot \rfloor$.

Lemma A.23 *If $t <^f m$, $|\sigma| = m$, $\sigma <^f m'$ and $|\sigma'| = m'$ then $(t \cdot \sigma) \cdot \sigma' = t \cdot (\sigma \cdot \sigma')$.*

Trivial induction, with the only interesting case where $t = f_i$. The left hand side becomes $(f_i \cdot \sigma) \cdot \sigma' = (\sigma.i) \cdot \sigma'$. The right hand side becomes $f_i \cdot (\sigma \cdot \sigma') = (\sigma \cdot \sigma').i = (\sigma.i) \cdot \sigma'$.

Lemma A.24 *If $|\sigma| = m$ and $|\Phi| = m$ then $id_\Phi \cdot \sigma = \sigma$.*

Trivial by induction on Φ.

Lemma A.25 *If $\lceil t \rceil_m^n = \lceil t' \rceil_m^n$ then $t = t'$.*

By induction on the structure of t. In each case we perform induction on t' as well. The only interesting case is when $t = f_i$ and $t' = b_n$. We have that $\lceil t' \rceil = f_m$; so it could be that $i = m$. This is avoided from the implicit assumption that $t <^f m$ (that is required to apply freshening).

The main substitution theorem that we are proving is the following.

Theorem A.26 (Substitution)
If $\Phi \vdash t : t'$ and $\Phi' \vdash \sigma : \Phi$ then $\Phi' \vdash t \cdot \sigma : t' \cdot \sigma'$.

By structural induction on the typing derivation for t.

Case $\dfrac{c : t \in \Sigma}{\Phi \vdash_\Sigma c : t}$ $\triangleright$
By applying the same typing rule we get that $\Phi' \vdash c : t$. By inversion of the well-formedness of Σ, we get that $\bullet \vdash t : t'$. Thus from lemma A.14 we get that $t <^f 0$ and from lemma A.18 we get that $t \cdot \sigma = t$. Considering also that $c \cdot \sigma = c$, the derivation $\Phi' \vdash c : t$ proves the desired.

Case $\dfrac{\Phi.i = t}{\Phi \vdash f_i : t}$ $\triangleright$
We have that $f_i \cdot \sigma = \sigma.i$. Directly using lemma A.20 we get that $\Phi' \vdash \sigma.i : t \cdot \sigma$.

Case $\dfrac{(s, s') \in \mathcal{A}}{\Phi \vdash s : s'}$ $\triangleright$
Trivial by application of the same rule and the definition of $\cdot$.

Case $\dfrac{\Phi \vdash t_1 : s \qquad \Phi,\, t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : s' \qquad (s, s', s'') \in \mathcal{R}}{\Phi \vdash \Pi(t_1).t_2 : s''}$ $\triangleright$
By induction hypothesis for t_1 we get: $\Phi' \vdash t_1 \cdot \sigma : s$.
By induction hypothesis for $\Phi, t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : s'$ and $\Phi', t_1 \cdot \sigma \vdash (\sigma, f_{|\Phi'|}) : (\Phi, t_1)$ we get: $\Phi', t_1 \cdot \sigma \vdash \lceil t_2 \rceil_{|\Phi|} \cdot (\sigma, f_{|\Phi'|}) :$

$s' \cdot (\sigma, f_{|\Phi'|})$.
We have $s' = s' \cdot (\sigma, f_{|\Phi'|})$ trivially.
Also by the lemma A.21, $\lceil t_2 \rceil_{|\Phi|} \cdot (\sigma, f_{|\Phi'|}) = \lceil t_2 \cdot \sigma \rceil_{|\Phi'|}$.
Thus by application of the same typing rule we get $\Phi' \vdash \Pi(t_1 \cdot \sigma).(t_2 \cdot \sigma) : s''$ which is the desired, since $(\Pi(t_1).t_2) \cdot \sigma = \Pi(t_1 \cdot \sigma).(t_2 \cdot \sigma)$.

Case $\dfrac{\Phi \vdash t_1 : s \qquad \Phi,\, t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : t' \qquad \Phi \vdash \Pi(t_1).\, \lfloor t' \rfloor_{|\Phi|+1} : s'}{\Phi \vdash \lambda(t_1).t_2 : \Pi(t_1).\, \lfloor t' \rfloor_{|\Phi|+1}}$ $\triangleright$
Similarly to the above, from the inductive hypothesis for t_1 and t_2 we get:
$\Phi' \vdash t_1 \cdot \sigma : s$
$\Phi', t_1 \cdot \sigma \vdash \lceil t_2 \cdot \sigma \rceil_{|\Phi'|} : t' \cdot (\sigma, f_{|\Phi'|})$
From the inductive hypothesis for $\Pi(t_1).\, \lfloor t' \rfloor$ we get: $\Phi' \vdash (\Pi(t_1).\, \lfloor t' \rfloor_{|\Phi|+1}) \cdot \sigma : s'$.
By the definition of $\cdot$ we get: $\Phi' \vdash \Pi(t_1 \cdot \sigma).(\lfloor t' \rfloor_{|\Phi+1|} \cdot \sigma) : s'$.
By the lemma A.22, we have that $(\lfloor t' \rfloor_{|\Phi+1|} \cdot \sigma) = \lfloor t' \cdot (\sigma, f_{|\Phi'|}) \rfloor_{|\Phi'|+1}$.
Thus we get $\Phi' \vdash \Pi(t_1 \cdot \sigma).\, \lfloor t' \cdot (\sigma, f_{|\Phi'|}) \rfloor_{|\Phi'|+1} : s'$.
We can now apply the same typing rule to get: $\Phi' \vdash \lambda(t_1 \cdot \sigma).(t_2 \cdot \sigma) : \Pi(t_1 \cdot \sigma).\, \lfloor t' \cdot (\sigma, f_{|\Phi'|}) \rfloor_{|\Phi'|+1}$.
We have $\Pi(t_1 \cdot \sigma).\, \lfloor t' \cdot (\sigma, f_{|\Phi'|}) \rfloor_{|\Phi'|+1} = \Pi(t_1 \cdot \sigma).((\lfloor t' \rfloor_{|\Phi|+1}) \cdot \sigma) = (\Pi(t_1).\, \lfloor t' \rfloor_{|\Phi|+1}) \cdot \sigma$, thus this is the desired result.

Case $\dfrac{\Phi \vdash t_1 : \Pi(t).t' \qquad \Phi \vdash t_2 : t}{\Phi \vdash t_1\ t_2 : \lceil t' \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2)}$ $\triangleright$
By induction hypothesis for t_1 we get $\Phi' \vdash t_1 \cdot \sigma : \Pi(t \cdot \sigma).(t' \cdot \sigma)$.
By induction hypothesis for t_2 we get $\Phi' \vdash t_2 \cdot \sigma : t \cdot \sigma$.
By application of the same typing rule we get $\Phi' \vdash (t_1\ t_2) \cdot \sigma : \lceil t' \cdot \sigma \rceil_{|\Phi'|} \cdot (\mathsf{id}_{\Phi'}, t_2 \cdot \sigma)$.
We have that $\lceil t' \cdot \sigma \rceil_{|\Phi'|} \cdot (\mathsf{id}_{\Phi'}, t_2 \cdot \sigma) = (\lceil t' \rceil_{|\Phi|} \cdot (\sigma, f_{|\Phi'|})) \cdot (\mathsf{id}_{\Phi'},\, t_2 \cdot \sigma)$ due to lemma A.21
From lemma A.23 $(t \cdot \sigma) \cdot \sigma' = t \cdot (\sigma \cdot \sigma')$, we further have that the above is equal to $\lceil t' \rceil_{|\Phi|} \cdot ((\sigma, f_{|\Phi'|}) \cdot (\mathsf{id}_{\Phi'}, t_2 \cdot \sigma))$.
We will now prove that $((\sigma,\, f_{|\Phi'|}) \cdot (\mathsf{id}_{\Phi'},\, t_2 \cdot \sigma) = \sigma,\, (t_2 \cdot \sigma)$.
By definition we have $(\sigma, f_{|\Phi'|}) \cdot (\mathsf{id}_{\Phi'},\, t_2 \cdot \sigma) = (\sigma \cdot (\mathsf{id}_{\Phi'},\, t_2 \cdot \sigma)),\, (f_{|\Phi'|} \cdot (\mathsf{id}_{\Phi'},\, t_2 \cdot \sigma)) = (\sigma \cdot (\mathsf{id}_{\Phi'},\, t_2 \cdot \sigma)),\, t_2 \cdot \sigma$.
Due to lemma A.16, we have that $\sigma <^f |\Phi'|$. Thus from lemma A.19, we get that $\sigma \cdot (\mathsf{id}_{\Phi'},\, t_2) = \sigma \cdot \mathsf{id}_{\Phi'}$.
Last from lemma A.13 we get that $\sigma \cdot \mathsf{id}_{\Phi'} = \sigma$.
Thus we only need to show that $\lceil t' \rceil_{|\Phi|} \cdot (\sigma,\, (t_2 \cdot \sigma))$ is equal to $(\lceil t' \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2)) \cdot \sigma$.
As above, per lemma A.23, this is equal to $\lceil t' \rceil_{|\Phi|} \cdot ((\mathsf{id}_\Phi, t_2) \cdot \sigma)$.
From definition we have $((\mathsf{id}_\Phi, t_2) \cdot \sigma) = (\mathsf{id}_\Phi \cdot \sigma), (t_2 \cdot \sigma)$.
Furthermore, from lemma A.24 we get that $(\mathsf{id}_\Phi \cdot \sigma), (t_2 \cdot \sigma) = \sigma, (t_2 \cdot \sigma)$.
Thus we have the desired result.

Case (otherwise) $\triangleright$
Simple to prove based on the methods we have shown above.

Corollary A.27 *If* $\Phi' \vdash \sigma : \Phi$ *and* $\Phi'' \vdash \sigma' : \Phi'$ *then* $\Phi'' \vdash \sigma \cdot \sigma' : \Phi$.

Induction on the typing derivation for σ, with use of the substitution theorem A.26.

Lemma A.28 (Types are well-typed) *If* $\Phi \vdash t : t'$ *then either* $t' = \mathsf{Type}'$ *or* $\Phi \vdash t' : s$.

By structural induction on the typing derivation for t.

Case $\dfrac{c : t \in \Sigma}{\Phi \vdash_\Sigma c : t}$ $\triangleright$ Trivial by inversion of the well-formedness of Σ.

Case $\dfrac{\Phi.i = t}{\Phi \vdash f_i : t}$ $\triangleright$ Trivial by inversion of the well-formedness of Φ.

Case $\dfrac{(s,s') \in \mathcal{A}}{\Phi \vdash s : s'}$ $\triangleright$ By splitting cases for (s,s') and application of the same typing rule.

Case $\dfrac{\Phi \vdash t_1 : s \qquad \Phi,\, t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : s' \qquad (s,s',s'') \in \mathcal{R}}{\Phi \vdash \Pi(t_1).t_2 : s''}$ $\triangleright$ By splitting cases for (s,s',s'') and use of sort typing rule.

Case $\dfrac{\Phi \vdash t_1 : \Pi(t).t' \qquad \Phi \vdash t_2 : t}{\Phi \vdash t_1\ t_2 : \lceil t' \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2)}$ $\triangleright$
By induction hypothesis we get that $\Phi \vdash \Pi(t).t' : s$. By inversion of this judgement, we get that $\Phi,\, t \vdash \lceil t' \rceil : s'$.
Furthermore we have by lemma A.15 that $\Phi \vdash \mathsf{id}_{|\Phi|} : \Phi$, and using the typing for t_2 and lemma A.12, we get that $\Phi \vdash \mathsf{id}_{|\Phi|},\, t_2 : (\Phi,\, t)$.
Thus by application of the substitution lemma A.26 for $\lceil t' \rceil$ we get the desired result.

Case (otherwise) $\triangleright$ Simple to prove based on the methods we have shown above.

Lemma A.29 (Weakening) *If $\Phi \vdash t : t'$, then $\Phi, t_1, t_2, \cdots, t_n \vdash t : t'$.*

Using lemma A.15 we have that $\Phi, t_1, t_2, \cdots, t_n \vdash \mathsf{id}_\Phi : \Phi$.
Using the substitution lemma A.26 we get that $\Phi, t_1, t_2, \cdots, t_n \vdash t \cdot \mathsf{id}_\Phi : t' \cdot \mathsf{id}_\Phi$.
From lemma A.18 and A.14, we get that $t \cdot \mathsf{id}_\Phi = t$.
From lemma A.28 we further get $\Phi \vdash t' : s$ and applying the same lemmas as for t we get $t' \cdot \mathsf{id}_\Phi = t'$.

B. Extension with metavariables and polymorphic contexts

B.1 Extending with metavariables

First, we extend the previous definition of terms to account for metavariables.

Definition B.1 (Syntax of the language) *The syntax of the logic language is extended below. We furthermore add new syntactic classes for modal terms and environments of metavariables.*

$$
\begin{aligned}
t &::= \cdots \mid X_i/\sigma \\
T &::= [\Phi]t \\
\mathcal{M} &::= \bullet \mid \mathcal{M},\, T
\end{aligned}
$$

Now we gather all the places from the above section where something was defined through induction on terms, and redefine/extend them here. Things that are identical are noted.

Definition B.2 (Context length and access) *Identical to A.2. We furthermore define metavariables environment length and access here.*

$\boxed{|\mathcal{M}|}$

$$\begin{array}{lcl} |\bullet| & = & 0 \\ |\mathcal{M},\, T| & = & |\mathcal{M}|+1 \end{array}$$

$\boxed{\mathcal{M}.i}$

$$\begin{array}{lcl} (\mathcal{M},\, T).|\mathcal{M}| & = & T \\ (\mathcal{M},\, T).i & = & \mathcal{M}.i \end{array}$$

Definition B.3 (Substitution length) *Identical to A.3.*

Definition B.4 (Substitution access) *Identical to A.4.*

Definition B.5 (Substitution application) *This is the extension of definition A.5. We lift it to modal terms.*

$\boxed{t \cdot \sigma}$

$$(X_i/\sigma') \cdot \sigma \quad = \quad X_i/(\sigma' \cdot \sigma)$$

$\boxed{T \cdot \sigma}$

$$([\Phi]t) \cdot \sigma \quad = \quad t \cdot \sigma$$

Definition B.6 (Identity substitution) *Identical to A.6.*

Definition B.7 (Variable limits for terms and substitutions) *This is the extension of definition A.7 and definition A.8 (who are now mutually dependent). The definition for substitutions is identical.*

$\boxed{t <^f n}$

$$X_i/\sigma <^f n \quad \Leftarrow \quad \sigma <^f n$$

$\boxed{t <^b n}$

$$X_i/\sigma <^b n \quad \Leftarrow \quad \sigma <^b n$$

Definition B.8 (Freshening) *This is the extension of definition A.9. Furthermore we need to lift the freshening operation to substitutions.*

$\boxed{\lceil t \rceil^n_m}$

$$\lceil X_i/\sigma \rceil^n_m \quad = \quad X_i/(\lceil \sigma^n_m \rceil)$$

$\boxed{\lceil \sigma \rceil^n_m}$

$$\begin{array}{lcl} \lceil \bullet \rceil^n_m & = & \bullet \\ \lceil \sigma,\, t \rceil^n_m & = & (\lceil \sigma \rceil^n_m),\, \lceil t \rceil^n_m \end{array}$$

Definition B.9 (Binding) *This is the extension of definition A.10. As above, we need to lift binding to substitutions.*

$\boxed{\lfloor t \rfloor^n_m}$

$$\lfloor X_i/\sigma \rfloor^n_m \quad = \quad X_i/(\lfloor \sigma^n_m \rfloor)$$

$\boxed{\lfloor \sigma \rfloor^n_m}$

$$\begin{array}{lcl} \lfloor \bullet \rfloor^n_m & = & \bullet \\ \lfloor \sigma, t \rfloor^n_m & = & (\lfloor \sigma \rfloor^n_m),\ \lfloor t \rfloor^n_m \end{array}$$

Definition B.10 (Typing judgements) *The typing judgements defined in A.11 are adjusted as follows.*
First, the judgement $\Phi \vdash t : t'$ is replaced by the judgement $\mathcal{M};\ \Phi \vdash t : t'$ and the existing rules are adjusted as needed. Also we include a new rule shown below.
Second, the judgement $\vdash \Phi$ wf is replaced by the judgement $\mathcal{M} \vdash \Phi$ wf.
Third, the judgement $\mathcal{M};\ \Phi \vdash \sigma : \Phi'$ replaces the original judgement for substitutions.
The $\vdash \Sigma$ wf judgement stays as is, with the adjustment shown below.
Last, we introduce a new judgement $\vdash \mathcal{M}$ wf for meta-environments and a judgement $\mathcal{M} \vdash T : T'$ for modal terms.

$\boxed{\vdash \Sigma \text{ wf}}$

$$\frac{}{\vdash \Sigma \text{ wf}} \qquad\qquad \frac{\vdash \Sigma \text{ wf} \qquad \bullet;\bullet \vdash_\Sigma t : s \qquad (c :) \notin \Sigma}{\vdash (\Sigma,\ c : t) \text{ wf}}$$

$\boxed{\mathcal{M};\ \Phi \vdash t : t'}$

$$\frac{\mathcal{M}.i = T \qquad T = [\Phi']t' \qquad \mathcal{M};\ \Phi \vdash \sigma : \Phi'}{\mathcal{M};\ \Phi \vdash X_i/\sigma : t' \cdot \sigma}$$

$\boxed{\vdash \mathcal{M} \text{ wf}}$

$$\vdash \bullet \text{ wf} \qquad\qquad \frac{\vdash \mathcal{M} \text{ wf} \qquad \mathcal{M} \vdash [\Phi]t : [\Phi]s}{\vdash (\mathcal{M},\ [\Phi]t) \text{ wf}}$$

$\boxed{\mathcal{M} \vdash T : T'}$

$$\frac{\mathcal{M};\ \Phi \vdash t : t'}{\mathcal{M} \vdash [\Phi]t : [\Phi]t'}$$

We can now proceed to adjust the proofs from above in order to handle the additional cases of the extension.

Lemma B.11 (Extension of lemmas A.12 and A.13) *1. If $t <^f m$ and $|\Phi| = m$ then $t \cdot id_\Phi = t$.*
2. If $\sigma <^f m$ and $|\Phi| = m$ then $\sigma \cdot id_\Phi = \sigma$.

The two lemmas become mutually dependent. For the first part, we proceed as previously by induction on t, and the only additional case we need to take into account is for the extension[1]:
We have that $(X_i/\sigma) \cdot \mathsf{id}_m = X_i/(\sigma \cdot \mathsf{id}_m)$. Using the second part, we have that $X_i/(\sigma \cdot \mathsf{id}_m) = X_i/\sigma$. The second part is proved as previously.

[1] We will not note this any more below; all the proofs mimic the inductive structure of the base proofs

Lemma B.12 (Extension of lemmas A.14 and A.16) *1. If* $\mathcal{M};\ \Phi \vdash t : t'$ *then* $t <^f |\Phi|$ *and* $t <^b 0$.

2. If $\mathcal{M};\ \Phi \vdash \sigma : \Phi'$ *then* $\sigma <^f |\Phi|$, $\sigma <^b 0$ *and* $|\sigma| = |\Phi'|$.

Again the two lemmas become mutually dependent when they weren't before. For the first one, we have that $\mathcal{M};\ \Phi \vdash X_i/\sigma : t'$; using the second part, we have that $\sigma <^f |\Phi|$ and $\sigma <^b 0$. By definition we thus have $X_i/\sigma <^f |\Phi|$ and $X_i/\sigma <^b 0$. The second part is proved as previously.

Lemma B.13 (Extension of lemma A.15) *If* $\mathcal{M} \vdash \Phi$ *wf then for any* Φ' *and* $t_{1\cdots n}$ *such that* $\Phi' = \Phi,\ t_1,\ t_2, \cdots, t_n$ *and* $\mathcal{M} \vdash \Phi'$ *wf, we have that* $\mathcal{M};\ \Phi' \vdash id_\Phi : \Phi$.

Identical as before.

Lemma B.14 (Extension of lemma A.17) *If* $\mathcal{M} \vdash \Phi$ *wf and* $|\Phi| = n$ *then for all* $i < n$, $\Phi.i <^f i$.

Identical as before.

Lemma B.15 (Extension of lemmas A.18 and A.19) *1. If* $t <^f m$, $|\sigma| = m$ *and* $t \cdot \sigma = t'$ *then* $t \cdot (\sigma, t_1, t_2, \cdots, t_n) = t'$.

2. If $\sigma <^f m$, $|\sigma'| = m$ *and* $\sigma \cdot \sigma' = \sigma_r$ *then* $\sigma \cdot (\sigma', t_1, t_2, \cdots, t_n) = \sigma_r$.

For the first part, taking $t = X_i/\sigma'$, we have that $X/\sigma' <^f m$ and thus $\sigma' <^f m$.
Furthermore we have $(X_i/\sigma') \cdot \sigma = X_i/(\sigma' \cdot \sigma) = X_i/\sigma_r$, assuming $\sigma_r = \sigma' \cdot \sigma$.
Using the second lemma we have that $\sigma' \cdot (\sigma, t_1, t_2, \cdots, t_n) = \sigma_r$.
Thus we also have that $(X_i/\sigma') \cdot (\sigma, t_1, t_2, \cdots, t_n) = X_i/(\sigma' \cdot (\sigma, t_1, t_2, \cdots, t_n)) = X_i/\sigma_r$.
For the second part, the proof proceeds as previously.

Lemma B.16 (Extension of lemma A.20) *If* $\mathcal{M} \vdash \Phi$ *wf,* $\Phi.i = t$ *and* $\mathcal{M};\ \Phi' \vdash \sigma : \Phi$, *then* $\mathcal{M};\ \Phi' \vdash \sigma.i : t \cdot \sigma$.

Identical as before.

Lemma B.17 (Extension of lemma A.21 and new lemma for substitutions) *1. If* $t <^f m$, $t <^b n+1$, $\sigma <^f m'$ *and* $|\sigma| = m$ *then* $\lceil t \cdot \sigma \rceil^n_{m'} = \lceil t \rceil^n_m \cdot (\sigma, f_{m'})$.

2. If $\sigma' <^f m$, $\sigma' <^b n+1$, $\sigma <^f m'$ *and* $|\sigma| = m$ *then* $\lceil \sigma' \cdot \sigma \rceil^n_{m'} = \lceil \sigma' \rceil^n_m \cdot (\sigma, f_{m'})$.

The second part of this lemma is a new lemma; it corresponds to the lifting of the first part to substitutions.
For the first part, we have: $\lceil (X_i/\sigma') \cdot \sigma \rceil^n_{m'} = \lceil X_i/(\sigma' \cdot \sigma) \rceil^n_{m'} = X_i/\lceil \sigma' \cdot \sigma \rceil^n_{m'}$.
Using the second part, we have that this is equal to $X_i/(\lceil \sigma' \rceil^n_m \cdot (\sigma,\ f_{m'}))$.
Furthermore, this is equal to $(X_i/\lceil \sigma' \rceil^n_m) \cdot (\sigma,\ f_{m'})$.
Last, this is equal to $(\lceil X_i/\sigma' \rceil^n_m) \cdot (\sigma,\ f_{m'})$, which is the desired.

For the second part, we proceed by induction on σ'.
If $\sigma' = \bullet$, the result is trivial.
If $\sigma' = \sigma'', t$ then $\lceil (\sigma'',\ t) \cdot \sigma \rceil^n_{m'} = \lceil (\sigma'' \cdot \sigma),\ t \cdot \sigma \rceil^n_{m'} = \lceil \sigma'' \cdot \sigma \rceil^n_{m'},\ \lceil t \cdot \sigma \rceil^n_{m'}$.
Using the induction hypothesis and the first part, we have that this is equal to $\lceil \sigma'' \rceil^n_m \cdot (\sigma,\ f_{m'}),\ \lceil t \rceil^n_m \cdot (\sigma,\ f_{m'}) = \lceil \sigma'',\ t \rceil^n_m \cdot (\sigma,\ f_{m'})$, which is the desired.

Lemma B.18 (Extension of lemma A.22 and new lemma for substitutions) *1. If* $t <^f m+1$, $t <^b n$, $\sigma <^f m'$ *and* $|\sigma| = m$ *then* $\lfloor t \cdot (\sigma, f_{m'}) \rfloor^n_{m'+1} = \lfloor t \rfloor^n_{m+1} \cdot \sigma$.

2. If $\sigma' <^f m+1$, $\sigma' <^b n$, $\sigma <^f m'$ *and* $|\sigma| = m$ *then* $\lfloor \sigma' \cdot (\sigma, f_{m'}) \rfloor^n_{m'+1} = \lfloor \sigma' \rfloor^n_{m+1} \cdot \sigma$.

This proof is entirely similar to the above for both parts.

Lemma B.19 (Extension of lemma A.23 and new lemma for substitutions) *1. If $t <^f m$, $|\sigma| = m$, $\sigma <^f m'$ and $|\sigma'| = m'$ then $(t \cdot \sigma) \cdot \sigma' = t \cdot (\sigma \cdot \sigma')$.*

2. If $\sigma_1 <^f m$, $|\sigma| = m$, $\sigma <^f m'$ and $|\sigma'| = m'$ then $(\sigma_1 \cdot \sigma) \cdot \sigma' = \sigma_1 \cdot (\sigma \cdot \sigma')$.

Entirely similar to the above.

Lemma B.20 (Extension of lemma A.24) *If $|\sigma| = m$ and $|\Phi| = m$ then $id_\Phi \cdot \sigma = \sigma$.*

Identical as before.

Lemma B.21 (Extension of lemma A.25) *1. If $\lceil t \rceil^n_m = \lceil t' \rceil^n_m$ then $t = t'$.*

2. If $\lceil \sigma \rceil^n_m = \lceil \sigma' \rceil^n_m$ then $\sigma = \sigma'$.

Part 1 is identical as before, with the additional case $t = X_i/\sigma$ and $t' = X_i/\sigma'$ handled using the second part. Part 2 is proved by induction on the structure of σ.

Theorem B.22 (Extension of main substitution theorem A.26 and corollary A.27) *1. If $\mathcal{M};\ \Phi \vdash t : t'$ and $\mathcal{M};\ \Phi' \vdash \sigma : \Phi$ then $\mathcal{M};\ \Phi' \vdash t \cdot \sigma : t' \cdot \sigma$.*

2. If $\mathcal{M};\ \Phi' \vdash \sigma : \Phi$ and $\mathcal{M};\ \Phi'' \vdash \sigma' : \Phi'$ then $\mathcal{M};\ \Phi'' \vdash \sigma \cdot \sigma' : \Phi$.

3. If $\mathcal{M} \vdash [\Phi']t : [\Phi']t'$ and $\mathcal{M};\ \Phi \vdash \sigma : \Phi'$ then $\mathcal{M} \vdash [\Phi]t \cdot \sigma : [\Phi]t' \cdot \sigma$.

For the first part we have, when $t = X_i/\sigma_0$:
From $\mathcal{M};\ \Phi \vdash X_i/\sigma_0 : t'$ we get that $\mathcal{M}.i = [\Phi_0]t_0$, $\mathcal{M};\ \Phi \vdash \sigma_0 : \Phi_0$ and $t' = t_0 \cdot \sigma_0$.
Applying the second part of the lemma for $\sigma = \sigma_0$ and $\sigma' = \sigma$ we get that $\mathcal{M};\ \Phi' \vdash \sigma_0 \cdot \sigma' : \Phi_0$.
Thus applying the same typing rule for $t = X_i/(\sigma_0 \cdot \sigma)$ we get that $\mathcal{M};\ \Phi' \vdash X_i/(\sigma_0 \cdot \sigma') : t_0 \cdot (\sigma_0 \cdot \sigma')$.
Taking into account the definition of $\cdot$ and also lemma B.19, we have that this is the desired result.

For the second part, the proof is identical to the proof done earlier.
For the third part, by typing inversion for $[\Phi']t$ we get that $\mathcal{M};\ \Phi' \vdash t : t'$.
Using the first part we get that $\mathcal{M};\ \Phi \vdash t \cdot \sigma : t' \cdot \sigma$.
Using the typing rule for modal terms we get $\mathcal{M} \vdash [\Phi]t \cdot \sigma : [\Phi]t' \cdot \sigma$.

Lemma B.23 (Meta-variables context weakening) *1. If $\mathcal{M};\Phi \vdash t : t'$ then $\mathcal{M}, T_1, \cdots, T_n;\ \Phi \vdash t : t'$.*

2. If $\mathcal{M};\Phi \vdash \sigma : \Phi'$ then $\mathcal{M}, T_1, \cdots, T_n;\ \Phi \vdash \sigma : \Phi'$.

3. If $\mathcal{M} \vdash \Phi$ wf then $\mathcal{M}, T_1, \cdots, T_n \vdash \Phi$ wf.

4. If $\mathcal{M} \vdash T : T'$ then $\mathcal{M}, T_1, \cdots, T_n \vdash T : T'$.

All are trivial by induction on the typing derivations.

Lemma B.24 (Extension of lemma A.28) *If $\mathcal{M};\ \Phi \vdash t : t'$ then either $t' = \mathsf{Type}'$ or $\mathcal{M};\ \Phi \vdash t' : s$.*

When $t = X_i/\sigma$, by inversion of typing we get $\mathcal{M}.i = [\Phi']t''$, $\mathcal{M};\ \Phi \vdash \sigma : \Phi'$ and $t' = t'' \cdot \sigma$.
By inversion of well-formedness for $\mathcal{M}$ and lemma 4, we get that $\mathcal{M} \vdash \mathcal{M}.i : [\Phi']s$.
Furthermore by inversion of that we get $\mathcal{M};\ \Phi' \vdash t'' : s$.
By application of the substitution lemma B.22 for t'' and σ we get $\mathcal{M};\ \Phi \vdash t'' \cdot \sigma : s$, which is the desired result.

Lemma B.25 (Extension of the lemma A.29 and new lemma for substitutions) *1. If* $\mathcal{M};\ \Phi \vdash t : t'$ *then* $\mathcal{M};\ \Phi, t_1, t_2, \cdots, t_n \vdash t : t'$.

2. If $\mathcal{M};\ \Phi \vdash \sigma : \Phi'$ *then* $\mathcal{M};\ \Phi, t_1, t_2, \cdots, t_n \vdash \sigma : \Phi'$.

For the first part, proceed identically as before.
For the second part, the proof is entirely similar to the first part (construct and prove well-typedness of identity substitution, and then allude to substitution theorem).

Now we know that everything that all the theorems we had proved for the non-extended version still hold. We can now prove a new meta-substitution theorem. Before doing that we need some new definitions.

Definition B.26 (Substitutions of meta-variables) *The syntax of substitutions of meta-variables is defined as follows.*

$$\sigma_{\mathcal{M}} ::= \bullet \mid \sigma_{\mathcal{M}},\ T$$

Definition B.27 (Meta-substitution length and access) *We define the length of meta-substitutions and accessing the i-th element as follows.*

$\boxed{|\sigma_{\mathcal{M}}|}$

$$\begin{array}{lcl} |\bullet| & = & 0 \\ |\sigma_{\mathcal{M}},\ T| & = & |\sigma_{\mathcal{M}}| + 1 \end{array}$$

$\boxed{\sigma_{\mathcal{M}}.i}$

$$\begin{array}{lcl} (\sigma_{\mathcal{M}},\ T).|\sigma_{\mathcal{M}}| & = & T \\ (\sigma_{\mathcal{M}},\ T).i & = & \sigma_{\mathcal{M}}.i \end{array}$$

Definition B.28 (Meta-substitution application) *The application of meta-substitutions is defined as follows. We mark the interesting cases with a star.*

$\boxed{t \cdot \sigma_{\mathcal{M}}}$

$$\begin{array}{lcl} s \cdot \sigma_{\mathcal{M}} & = & s \\ c \cdot \sigma_{\mathcal{M}} & = & c \\ f_i \cdot \sigma_{\mathcal{M}} & = & f_i \\ b_i \cdot \sigma_{\mathcal{M}} & = & b_i \\ (\lambda(t_1).t_2) \cdot \sigma_{\mathcal{M}} & = & \lambda(t_1 \cdot \sigma_{\mathcal{M}}).(t_2 \cdot \sigma_{\mathcal{M}}) \\ (t_1\ t_2) \cdot \sigma_{\mathcal{M}} & = & (t_1 \cdot \sigma_{\mathcal{M}})\ (t_2 \cdot \sigma_{\mathcal{M}}) \\ (\Pi(t_1).t_2) \cdot \sigma_{\mathcal{M}} & = & \Pi(t_1 \cdot \sigma_{\mathcal{M}}).(t_2 \cdot \sigma_{\mathcal{M}}) \end{array}$$

$\boxed{t \cdot \sigma_{\mathcal{M}} \text{ (continued)}}$

$$
\begin{array}{lll}
(t_1 = t_2) \cdot \sigma_{\mathcal{M}} & = & (t_1 \cdot \sigma_{\mathcal{M}}) = (t_2 \cdot \sigma_{\mathcal{M}}) \\
(\mathsf{conv}\ t_1\ t_2) \cdot \sigma_{\mathcal{M}} & = & \mathsf{conv}\ (t_1 \cdot \sigma_{\mathcal{M}})\ (t_2 \cdot \sigma_{\mathcal{M}}) \\
(\mathsf{refl}\ t) \cdot \sigma_{\mathcal{M}} & = & \mathsf{refl}\ (t \cdot \sigma_{\mathcal{M}}) \\
(\mathsf{symm}\ t) \cdot \sigma_{\mathcal{M}} & = & \mathsf{symm}\ (t \cdot \sigma_{\mathcal{M}}) \\
(\mathsf{trans}\ t_1\ t_2) \cdot \sigma_{\mathcal{M}} & = & \mathsf{trans}\ (t_1 \cdot \sigma_{\mathcal{M}})\ (t_2 \cdot \sigma_{\mathcal{M}}) \\
(\mathsf{congapp}\ t_1\ t_2) \cdot \sigma_{\mathcal{M}} & = & \mathsf{congapp}\ (t_1 \cdot \sigma_{\mathcal{M}})\ (t_2 \cdot \sigma_{\mathcal{M}}) \\
(\mathsf{congimpl}\ t_1\ t_2) \cdot \sigma_{\mathcal{M}} & = & \mathsf{congimpl}\ (t_1 \cdot \sigma_{\mathcal{M}})\ (t_2 \cdot \sigma_{\mathcal{M}}) \\
(\mathsf{conglam}\ t) \cdot \sigma_{\mathcal{M}} & = & \mathsf{conglam}\ (t \cdot \sigma_{\mathcal{M}}) \\
(\mathsf{congpi}\ t) \cdot \sigma_{\mathcal{M}} & = & \mathsf{congpi}\ (t \cdot \sigma_{\mathcal{M}}) \\
(\mathsf{beta}\ t_1\ t_2) \cdot \sigma_{\mathcal{M}} & = & \mathsf{beta}\ (t_1 \cdot \sigma_{\mathcal{M}})\ (t_2 \cdot \sigma_{\mathcal{M}}) \\
* \quad (X_i/\sigma) \cdot \sigma_{\mathcal{M}} & = & (\sigma_{\mathcal{M}}.i) \cdot (\sigma \cdot \sigma_{\mathcal{M}})
\end{array}
$$

$\boxed{\sigma \cdot \sigma_{\mathcal{M}}}$

$$
\begin{array}{lll}
\bullet \cdot \sigma_{\mathcal{M}} & = & \bullet \\
(\sigma,\ t) \cdot \sigma_{\mathcal{M}} & = & \sigma \cdot \sigma_{\mathcal{M}},\ t \cdot \sigma_{\mathcal{M}}
\end{array}
$$

$\boxed{\Phi \cdot \sigma_{\mathcal{M}}}$

$$
\begin{array}{lll}
\bullet \cdot \sigma_{\mathcal{M}} & = & \bullet \\
(\Phi,\ t) \cdot \sigma_{\mathcal{M}} & = & \Phi \cdot \sigma_{\mathcal{M}},\ t \cdot \sigma_{\mathcal{M}}
\end{array}
$$

$\boxed{T \cdot \sigma_{\mathcal{M}}}$

$$
* \quad ([\Phi]\, t) \cdot \sigma_{\mathcal{M}} \quad = \quad [\Phi \cdot \sigma_{\mathcal{M}}]\, (t \cdot \sigma_{\mathcal{M}})
$$

Definition B.29 (Meta-substitution typing) *The typing judgement for meta-substitutions is as follows.*

$\boxed{\mathcal{M} \vdash \sigma_{\mathcal{M}} : \mathcal{M}'}$

$$
\frac{}{\mathcal{M} \vdash \bullet : \bullet}
\qquad\qquad
\frac{\mathcal{M} \vdash \sigma_{\mathcal{M}} : \mathcal{M}' \qquad \mathcal{M} \vdash T : T' \cdot \sigma_{\mathcal{M}}}{\mathcal{M} \vdash (\sigma_{\mathcal{M}},\ T) : (\mathcal{M}',\ T')}
$$

We proceed to prove the meta-substitution theorem.

The lemmas that we need are the following:

Lemma B.30 (Limits for elements of metasubstitutions) *If $\mathcal{M} \vdash \sigma_{\mathcal{M}} : \mathcal{M}'$ and $\sigma_{\mathcal{M}}.i = [\Phi]\, t$ then $t <^f |\Phi|$ and $t <^b 0$.*

By repeated inversion of typing for $\sigma_{\mathcal{M}}$ we get that $\mathcal{M}' \vdash \sigma_{\mathcal{M}}.i : T'$ for some $\mathcal{M}'$ and T'. By inversion we get that $\mathcal{M}';\ \Phi \vdash t : t'$. By use of lemma 2 we get the desired.

Lemma B.31 (Freshen on closed term) *If $t <^b n$ then $\lceil t \cdot \sigma \rceil^n_m = t \cdot \lceil \sigma \rceil^n_m$.*

Easy by induction on t.

Lemma B.32 (Interaction of freshen and metasubstitution application) *1. If* $\mathcal{M} \vdash \sigma_{\mathcal{M}} : \mathcal{M}'$ *then* $\lceil t \rceil^n_m \cdot \sigma_{\mathcal{M}} = \lceil t \cdot \sigma_{\mathcal{M}} \rceil^n_m$

2. If $\mathcal{M} \vdash \sigma_{\mathcal{M}} : \mathcal{M}'$ *then* $\lceil \sigma \rceil^n_m \cdot \sigma_{\mathcal{M}} = \lceil \sigma \cdot \sigma_{\mathcal{M}} \rceil^n_m$

The first part is proved by induction on t. The interesting case is the metavariables case, where we have the following.
$\lceil X_i/\sigma \rceil^n_m \cdot \sigma_{\mathcal{M}} = (X_i/\lceil \sigma \rceil^n_m) \cdot \sigma_{\mathcal{M}} = \sigma_{\mathcal{M}}.i \cdot (\lceil \sigma \rceil^n_m \cdot \sigma_{\mathcal{M}}) = \sigma_{\mathcal{M}}.i \cdot \lceil \sigma \cdot \sigma_{\mathcal{M}} \rceil^n_m$ based on the second part.
Now $\sigma_{\mathcal{M}}.i = [\Phi]t$ and the above is further equal to: $t \cdot \lceil \sigma \cdot \sigma_{\mathcal{M}} \rceil^n_m$. The right-hand side is rewritten as follows:
$\lceil X_i/\sigma \cdot \sigma_{\mathcal{M}} \rceil^n_m = \lceil \sigma_{\mathcal{M}}.i \cdot (\sigma \cdot \sigma_{\mathcal{M}}) \rceil^n_m = \lceil t \cdot (\sigma \cdot \sigma_{\mathcal{M}}) \rceil^n_m = t \cdot \lceil \sigma \cdot \sigma_{\mathcal{M}} \rceil^n_m$ using lemma B.31 and also B.30.
The second part is proved trivially using induction.

Lemma B.33 (Bind on closed term) *If* $t <^b n$ *then* $\lfloor t \cdot \sigma \rfloor^n_m = t \cdot \lfloor \sigma \rfloor^n_m$.

Easy by induction on t.

Lemma B.34 (Interaction of bind and metasubstitution application) *1. If* $\mathcal{M} \vdash \sigma_{\mathcal{M}} : \mathcal{M}'$ *then* $\lfloor t \rfloor^n_m \cdot \sigma_{\mathcal{M}} = \lfloor t \cdot \sigma_{\mathcal{M}} \rfloor^n_m$

2. If $\mathcal{M} \vdash \sigma_{\mathcal{M}} : \mathcal{M}'$ *then* $\lfloor \sigma \rfloor^n_m \cdot \sigma_{\mathcal{M}} = \lfloor \sigma \cdot \sigma_{\mathcal{M}} \rfloor^n_m$

Similar to the equivalent lemma for freshen.

Lemma B.35 (Interaction of substitution application and metasubstitution application) *1.* $(t \cdot \sigma) \cdot \sigma_{\mathcal{M}} = (t \cdot \sigma_{\mathcal{M}}) \cdot (\sigma \cdot \sigma_{\mathcal{M}})$

2. $(\sigma \cdot \sigma') \cdot \sigma_{\mathcal{M}} = (\sigma \cdot \sigma_{\mathcal{M}}) \cdot (\sigma' \cdot \sigma_{\mathcal{M}})$

In the first part, we perform induction on t. The interesting case is the metavariables case. We have:
$((X_i/\sigma') \cdot \sigma) \cdot \sigma_{\mathcal{M}} = (X_i/(\sigma' \cdot \sigma)) \cdot \sigma_{\mathcal{M}} = \sigma_{\mathcal{M}}.i \cdot ((\sigma' \cdot \sigma) \cdot \sigma_{\mathcal{M}})$.
From the second part, this is equal to: $\sigma_{\mathcal{M}}.i \cdot ((\sigma' \cdot \sigma_{\mathcal{M}}) \cdot (\sigma \cdot \sigma_{\mathcal{M}}))$.
There exists a t such that $\sigma_{\mathcal{M}}.i = [\Phi]t$ and thus the above is further equal to:
$t \cdot ((\sigma' \cdot \sigma_{\mathcal{M}}) \cdot (\sigma \cdot \sigma_{\mathcal{M}})) = (t \cdot (\sigma' \cdot \sigma_{\mathcal{M}})) \cdot (\sigma \cdot \sigma_{\mathcal{M}})$ based on lemma B.19.
The right-hand side is written as: $((X_i/\sigma') \cdot \sigma_{\mathcal{M}}) \cdot (\sigma \cdot \sigma_{\mathcal{M}}) = (t \cdot (\sigma' \cdot \sigma_{\mathcal{M}})) \cdot (\sigma \cdot \sigma_{\mathcal{M}})$. Thus the desired.
The second part is trivially proved by induction and use of the first part.

Lemma B.36 (Application of metasubstitution to identity substitution) $id_\Phi \cdot \sigma_{\mathcal{M}} = id_{\Phi \cdot \sigma_{\mathcal{M}}}$

Trivial by induction on Φ.

Lemma B.37 (Redundant elements in metasubstitutions) *1. If* $\mathcal{M}; \Phi \vdash t : t'$ *and* $|\sigma_{\mathcal{M}}| = |\mathcal{M}|$ *then* $t \cdot (\sigma_{\mathcal{M}}, T_1, T_2, \cdots, T_n) = t \cdot \sigma_{\mathcal{M}}$.

2. If $\mathcal{M}; \Phi \vdash \sigma : \Phi'$ *and* $|\sigma_{\mathcal{M}}| = |\mathcal{M}|$ *then* $\sigma \cdot (\sigma_{\mathcal{M}}, T_1, T_2, \cdots, T_n) = \sigma \cdot \sigma_{\mathcal{M}}$.

3. If $\mathcal{M} \vdash \Phi$ *wf and* $|\sigma_{\mathcal{M}}| = |\mathcal{M}|$ *then* $\Phi \cdot (\sigma_{\mathcal{M}}, T_1, T_2, \cdots, T_n) = \Phi \cdot \sigma_{\mathcal{M}}$.

4. If $\mathcal{M} \vdash T : T'$ *and* $|\sigma_{\mathcal{M}}| = |\mathcal{M}|$ *then* $T \cdot (\sigma_{\mathcal{M}}, T_1, T_2, \cdots, T_n) = T \cdot \sigma_{\mathcal{M}}$.

By induction on the typing derivations.

Lemma B.38 (Type of i-th metasubstitution element) *If* $\vdash \mathcal{M}$ *wf and* $\mathcal{M} \vdash \sigma_{\mathcal{M}} : \mathcal{M}'$ *then* $\mathcal{M} \vdash \sigma_{\mathcal{M}}.i : (\mathcal{M}'.i) \cdot \sigma_{\mathcal{M}}$.

By induction and use of lemma B.37; furthermore using inversion of the well-formedness relation for $\mathcal{M}$. Similar to lemma A.20.

Theorem B.39 (Substitution over metavariables) *1. If* $\mathcal{M};\ \Phi \vdash t : t'$ *and* $\mathcal{M}' \vdash \sigma_{\mathcal{M}} : \mathcal{M}$ *then* $\mathcal{M}';\ \Phi \cdot \sigma_{\mathcal{M}} : t \cdot \sigma_{\mathcal{M}} : t' \cdot \sigma_{\mathcal{M}}$.

2. If $\mathcal{M};\ \Phi \vdash \sigma : \Phi'$ *and* $\mathcal{M}' \vdash \sigma_{\mathcal{M}} : \mathcal{M}$ *then* $\mathcal{M}';\ \Phi \cdot \sigma_{\mathcal{M}} \vdash \sigma \cdot \sigma_{\mathcal{M}} : \Phi' \cdot \sigma_{\mathcal{M}}$.

3. If $\mathcal{M} \vdash \Phi$ *wf and* $\mathcal{M}' \vdash \sigma_{\mathcal{M}} : \mathcal{M}$ *then* $\mathcal{M}' \vdash \Phi \cdot \sigma_{\mathcal{M}}$ *wf.*

4. If $\mathcal{M} \vdash T : T'$ *and* $\mathcal{M}' \vdash \sigma_{\mathcal{M}} : \mathcal{M}$ *then* $\mathcal{M}' \vdash T \cdot \sigma_{\mathcal{M}} : T' \cdot \sigma_{\mathcal{M}}$.

Part 1 Proceed by structural induction on the typing of t.

Case $\dfrac{c : t \in \Sigma}{\mathcal{M};\ \Phi \vdash_\Sigma c : t}\ \triangleright$
From inversion of the well-formedness of Σ we have that $\bullet;\bullet \vdash t : s$.
From lemma B.37 we have that $t \cdot \sigma_{\mathcal{M}} = t$.
So the result follows from application of the same typing rule for $\Phi \cdot \sigma_{\mathcal{M}}$.

Case $\dfrac{\Phi.i = t}{\mathcal{M};\ \Phi \vdash f_i : t}\ \triangleright$
We have $t \cdot \sigma_{\mathcal{M}} = (\Phi \cdot \sigma_{\mathcal{M}}).i$, so using the same typing rule we get $\mathcal{M}';\ \Phi \cdot \sigma_{\mathcal{M}} \vdash f_i : t \cdot \sigma_{\mathcal{M}}$.

Case $\dfrac{(s,s') \in \mathcal{A}}{\mathcal{M};\ \Phi \vdash s : s'}\ \triangleright$
Trivial by application of the same rule and the definition of $\cdot$.

Case $\dfrac{\mathcal{M};\ \Phi \vdash t_1 : s \qquad \mathcal{M};\ \Phi,\ t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : s' \qquad (s,s',s'') \in \mathcal{R}}{\mathcal{M};\ \Phi \vdash \Pi(t_1).t_2 : s''}\ \triangleright$
By induction hypothesis for t_1 we get: $\mathcal{M}';\ \Phi \cdot \sigma_{\mathcal{M}} \vdash t_1 \cdot \sigma_{\mathcal{M}} : s$.
By induction hypothesis for $\Phi, t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : s'$ we get:
$\mathcal{M}';\ \Phi \cdot \sigma_{\mathcal{M}}, t_1 \cdot \sigma_{\mathcal{M}} \vdash \lceil t_2 \rceil_{|\Phi|} \cdot \sigma_{\mathcal{M}} : s' \cdot \sigma_{\mathcal{M}}$.
We have $s' = s' \cdot \sigma_{\mathcal{M}}$ trivially.
Also by the lemma B.32, $\lceil t_2 \rceil_{|\Phi|} \cdot \sigma_{\mathcal{M}} = \lceil t_2 \cdot \sigma_{\mathcal{M}} \rceil_{|\Phi|}$.
Thus by application of the same typing rule we get $\mathcal{M}';\ \Phi \cdot \sigma_{\mathcal{M}} \vdash \Pi(t_1 \cdot \sigma_{\mathcal{M}}).(t_2 \cdot \sigma_{\mathcal{M}}) : s''$ which is the desired.

Case $\dfrac{\mathcal{M};\ \Phi \vdash t_1 : s \qquad \mathcal{M};\ \Phi,\ t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : t' \qquad \mathcal{M};\ \Phi \vdash \Pi(t_1).\ \lfloor t' \rfloor_{|\Phi|+1} : s'}{\mathcal{M};\ \Phi \vdash \lambda(t_1).t_2 : \Pi(t_1).\ \lfloor t' \rfloor_{|\Phi|+1}}\ \triangleright$
Similarly to the above, from the inductive hypothesis for t_1 and t_2 (and use of lemma B.32) we get:
$\sigma_{\mathcal{M}};\ \Phi \vdash t_1 \cdot \sigma_{\mathcal{M}} : s$
$\sigma_{\mathcal{M}};\ \Phi \cdot \sigma_{\mathcal{M}}, t_1 \cdot \sigma_{\mathcal{M}} \vdash \lceil t_2 \cdot \sigma_{\mathcal{M}} \rceil_{|\Phi|} : t' \cdot \sigma_{\mathcal{M}}$
From the inductive hypothesis for $\Pi(t_1).\ \lfloor t' \rfloor$ we get: $\mathcal{M}';\ \Phi \cdot \sigma_{\mathcal{M}} \vdash (\Pi(t_1).\ \lfloor t' \rfloor_{|\Phi|+1}) \cdot \sigma_{\mathcal{M}} : s'$.
By the definition of $\cdot$ we get: $\mathcal{M}';\ \Phi \cdot \sigma_{\mathcal{M}} \vdash \Pi(t_1 \cdot \sigma_{\mathcal{M}}).(\lfloor t' \rfloor_{|\Phi+1|} \cdot \sigma_{\mathcal{M}}) : s'$.
By the lemma B.34, we have that $(\lfloor t' \rfloor_{|\Phi+1|} \cdot \sigma_{\mathcal{M}}) = \lfloor t' \cdot \sigma_{\mathcal{M}} \rfloor_{|\Phi|+1}$.
Thus we get $\mathcal{M};\ \Phi \cdot \sigma_{\mathcal{M}} \vdash \Pi(t_1 \cdot \sigma_{\mathcal{M}}).\ \lfloor t' \cdot \sigma_{\mathcal{M}} \rfloor_{|\Phi|+1} : s'$.
We can now apply the same typing rule to get: $\mathcal{M};\ \Phi \cdot \sigma_{\mathcal{M}} \vdash \lambda(t_1 \cdot \sigma_{\mathcal{M}}).(t_2 \cdot \sigma_{\mathcal{M}}) : \Pi(t_1 \cdot \sigma_{\mathcal{M}}).\ \lfloor t' \cdot \sigma_{\mathcal{M}} \rfloor_{|\Phi|+1}$.
We have $\Pi(t_1 \cdot \sigma_{\mathcal{M}}).\ \lfloor t' \cdot \sigma_{\mathcal{M}} \rfloor_{|\Phi|+1} = \Pi(t_1 \cdot \sigma_{\mathcal{M}}).((\lfloor t' \rfloor_{|\Phi|+1}) \cdot \sigma_{\mathcal{M}}) = (\Pi(t_1).\ \lfloor t' \rfloor_{|\Phi|+1}) \cdot \sigma_{\mathcal{M}}$, thus this is the desired result.

Case $\dfrac{\mathcal{M};\ \Phi \vdash t_1 : \Pi(t).t' \qquad \mathcal{M};\ \Phi \vdash t_2 : t}{\mathcal{M};\ \Phi \vdash t_1\ t_2 : \lceil t' \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2)}\ \triangleright$

By induction hypothesis for t_1 we get $\mathcal{M}';\ \Phi \cdot \sigma_\mathcal{M} \vdash t_1 \cdot \sigma_\mathcal{M} : \Pi(t \cdot \sigma_\mathcal{M}).(t' \cdot \sigma_\mathcal{M})$.
By induction hypothesis for t_2 we get $\mathcal{M}';\ \Phi \cdot \sigma_\mathcal{M} \vdash t_2 \cdot \sigma_\mathcal{M} : t \cdot \sigma_\mathcal{M}$.
By application of the same typing rule we get $\mathcal{M}';\ \Phi \cdot \sigma_\mathcal{M} \vdash (t_1\ t_2) \cdot \sigma_\mathcal{M} : \lceil t' \cdot \sigma_\mathcal{M} \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2 \cdot \sigma_\mathcal{M})$.
We need to prove that $(\lceil t' \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2)) \cdot \sigma_\mathcal{M} = \lceil t' \cdot \sigma_\mathcal{M} \rceil_{|\Phi|} \cdot (\mathsf{id}_{\Phi \cdot \sigma_\mathcal{M}}, t_2 \cdot \sigma_\mathcal{M})$.
From lemma B.35 we have that $(\lceil t' \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2)) \cdot \sigma_\mathcal{M} = (\lceil t' \rceil_{|\Phi|} \cdot \sigma_\mathcal{M}) \cdot ((\mathsf{id}_\Phi, t_2) \cdot \sigma_\mathcal{M})$.
From lemma B.32 we get that this is further equal to: $(\lceil t' \cdot \sigma_\mathcal{M} \rceil_{|\Phi|}) \cdot ((\mathsf{id}_\Phi, t_2) \cdot \sigma_\mathcal{M})$.
From definition of $\cdot$ we get that this is equal to $(\lceil t' \cdot \sigma_\mathcal{M} \rceil_{|\Phi|}) \cdot (\mathsf{id}_\Phi \cdot \sigma_\mathcal{M},\ t_2 \cdot \sigma_\mathcal{M})$.
Last from B.36 we get the desired result.

Case $\dfrac{\mathcal{M}.i = T \qquad T = [\Phi']t' \qquad \mathcal{M};\ \Phi \vdash \sigma : \Phi'}{\mathcal{M};\ \Phi \vdash X_i/\sigma : t' \cdot \sigma}\ \triangleright$

Assuming that $\sigma_\mathcal{M}.i = [\Phi'']t$, we need to show that $\mathcal{M}';\ \Phi \cdot \sigma_\mathcal{M} \vdash t \cdot (\sigma \cdot \sigma_\mathcal{M}) : (t' \cdot \sigma) \cdot \sigma_\mathcal{M}$.
From lemma B.35, we have that $(t' \cdot \sigma) \cdot \sigma_\mathcal{M} = (t' \cdot \sigma_\mathcal{M}) \cdot (\sigma \cdot \sigma_\mathcal{M})$.
So equivalently we need to show $\mathcal{M}';\ \Phi \cdot \sigma_\mathcal{M} \vdash t \cdot (\sigma \cdot \sigma_\mathcal{M}) : (t' \cdot \sigma_\mathcal{M}) \cdot (\sigma \cdot \sigma_\mathcal{M})$.
Using the second part of the lemma for σ we get: $\mathcal{M}';\ \Phi \cdot \sigma_\mathcal{M} \vdash \sigma \cdot \sigma_\mathcal{M} : \Phi' \cdot \sigma_\mathcal{M}$.
From lemma B.38 we get that $\mathcal{M}' \vdash \sigma_\mathcal{M}.i : \mathcal{M}.i \cdot \sigma_\mathcal{M}$.
From hypothesis we have that $\mathcal{M}.i = [\Phi']t'$.
Thus the above typing judgement is rewritten as $\mathcal{M}' \vdash \sigma_\mathcal{M}.i : [\Phi' \cdot \sigma_\mathcal{M}]t \cdot \sigma_\mathcal{M}$.
By inversion we get that $\sigma_\mathcal{M}.i = [\Phi' \cdot \sigma_\mathcal{M}]t$ and that $\mathcal{M}';\ \Phi' \cdot \sigma_\mathcal{M} \vdash t : t' \cdot \sigma_\mathcal{M}$.
** Now we use the main substitution theorem B.22 for t and $\sigma \cdot \sigma_\mathcal{M}$ and get:
$\mathcal{M}';\ \Phi \cdot \sigma_\mathcal{M} \vdash t \cdot (\sigma \cdot \sigma_\mathcal{M}) : (t' \cdot \sigma_\mathcal{M}) \cdot (\sigma \cdot \sigma_\mathcal{M})$.

Case (otherwise) $\triangleright$
Simple to prove based on the methods we have shown above.

Part 2 By induction on the typing derivation of σ.

Case $\dfrac{\mathcal{M} \vdash \Phi\ \mathrm{wf}}{\mathcal{M};\ \Phi \vdash \bullet : \bullet}\ \triangleright$ Use of the same typing rule, for $\Phi \cdot \sigma_\mathcal{M}$ which is well formed based on part 3.

Case $\dfrac{\mathcal{M};\ \Phi \vdash \sigma : \Phi' \qquad \mathcal{M};\ \Phi \vdash t : t' \cdot \sigma}{\mathcal{M};\ \Phi \vdash \sigma, t : (\Phi', t')}\ \triangleright$ By induction hypothesis and use of part 1 we get:
$\mathcal{M}';\ \Phi \cdot \sigma_\mathcal{M} \vdash \sigma \cdot \sigma_\mathcal{M} : \Phi' \cdot \sigma_\mathcal{M}$
$\mathcal{M}';\ \Phi \cdot \sigma_\mathcal{M} \vdash t \cdot \sigma_\mathcal{M} : (t' \cdot \sigma) \cdot \sigma_\mathcal{M}$
By use of lemma B.35 in the typing for $t \cdot \sigma_\mathcal{M}$ we get that:
$\mathcal{M}';\ \Phi \cdot \sigma_\mathcal{M} \vdash t \cdot \sigma_\mathcal{M} : (t' \cdot \sigma_\mathcal{M}) \cdot (\sigma \cdot \sigma_\mathcal{M})$
By use of the same typing rule we get: $\mathcal{M}';\ \Phi \cdot \sigma_\mathcal{M} \vdash (\sigma \cdot \sigma_\mathcal{M},\ t \cdot \sigma_\mathcal{M}) : (\Phi' \cdot \sigma_\mathcal{M}, t' \cdot \sigma_\mathcal{M})$

Part 3 By induction on the well-formedness derivation of Φ.

Case $\mathcal{M} \vdash \bullet\ \mathrm{wf}\ \triangleright$
Trivial use of the same typing rule.

Case $\dfrac{\mathcal{M} \vdash \Phi\ \mathrm{wf} \qquad \mathcal{M};\ \Phi \vdash t : s}{\mathcal{M} \vdash \Phi,\ t\ \mathrm{wf}}\ \triangleright$
Use of induction hypothesis, part 2, and the same typing rule.

Part 4 By induction on the typing derivation for T.

Case $\dfrac{\mathcal{M};\ \Phi \vdash t : t'}{\mathcal{M} \vdash [\Phi]t : [\Phi]t'}\ \triangleright$

Using part 1 we get $\mathcal{M}'$; $\Phi \cdot \sigma_{\mathcal{M}} \vdash t \cdot \sigma_{\mathcal{M}} : t' \cdot \sigma_{\mathcal{M}}$. Thus using the same typing rule we get $\mathcal{M}' \vdash [\Phi \cdot \sigma_{\mathcal{M}}]t \cdot \sigma_{\mathcal{M}} : [\Phi \cdot \sigma_{\mathcal{M}}]t' \cdot \sigma_{\mathcal{M}}$, which is the desired result.

B.2 Extension with metavariables and polymorphic contexts

In order to incorporate polymorphic contexts, we change the representation of free variables from a deBruijn level to an index into a parametric context. We thus need to redefine the notions of length of a context, variable limits etc. in order to be compatible with the new definition of free variables.

Definition B.40 (Syntax of the language) *The syntax of the logic language is extended below. We use the syntactic class T for modal terms and modal contexts, and the syntactic class K for their classifiers (modal terms and context prefixes). Furthermore, we use a single context* Ψ *for both extensions.*

$$
\begin{array}{rl}
\Phi ::= & \cdots \mid \Phi, X_i \\
\sigma ::= & \cdots \mid \sigma,\ \mathbf{id}(X_i) \\
\Psi ::= & \bullet \mid \Psi,\ K \\
t ::= & s \mid c \mid f_{\mathbf{I}} \mid b_i \mid \lambda(t_1).t_2 \mid t_1\ t_2 \mid \Pi(t_1).t_2 \mid t_1 = t_2 \mid \mathsf{conv}\ t\ t \mid \mathsf{refl}\ t \mid \mathsf{symm}\ t \mid \mathsf{trans}\ t_1\ t_2 \mid \mathsf{congapp}\ t_1\ t_2 \\
& \mid \mathsf{congimpl}\ t_1\ t_2 \mid \mathsf{conglam}\ t \mid \mathsf{congpi}\ t \mid \mathsf{beta}\ t_1\ t_2 \mid X_i/\sigma \\
T ::= & [\Phi]t \mid [\Phi]\Phi' \\
K ::= & [\Phi]t \mid [\Phi]\,\mathsf{ctx} \\
\mathbf{I} ::= & \bullet \mid \mathbf{I},\ \cdot \mid \mathbf{I},\ |X_i|
\end{array}
$$

Definition B.41 (Substitution length) *Redefinition of B.3.*

$\boxed{|\sigma| = \mathbf{I}}$

$$
\begin{array}{lcl}
|\bullet| & = & \bullet \\
|\sigma,\ t| & = & |\sigma|,\ \cdot \\
|\sigma,\ \mathbf{id}(X_i)| & = & |\sigma|,\ |X_i|
\end{array}
$$

Definition B.42 (Ordering of indexes) *We define what it means for an index to be less than another index.*

$\boxed{\mathbf{I} < \mathbf{I}'}$

$$
\begin{array}{lcl}
\mathbf{I} & < & \mathbf{I}',\ \cdot \text{ when } \mathbf{I} = \mathbf{I}' \text{ or } \mathbf{I} < \mathbf{I}' \\
\mathbf{I} & < & \mathbf{I}',\ |X_i| \text{ when } \mathbf{I} = \mathbf{I}' \text{ or } \mathbf{I} < \mathbf{I}'
\end{array}
$$

$\boxed{\mathbf{I} \leq \mathbf{I}'}$

$$
\mathbf{I} \ \leq\ \mathbf{I}' \text{ when } \mathbf{I} = \mathbf{I}' \text{ or } \mathbf{I} < \mathbf{I}'
$$

Definition B.43 (Substitution access) *Redefinition of B.4. We assume* $\mathbf{I} < |\sigma|$.

$\boxed{\sigma.\mathbf{I}}$

$$\begin{array}{lcl}(\sigma,\, t).\mathbf{I} & = & t \text{ when } |\sigma| = \mathbf{I} \\ (\sigma,\, t).\mathbf{I} & = & \sigma.\mathbf{I} \text{ otherwise} \\ (\sigma,\, \mathbf{id}(X_i)).\mathbf{I} & = & t \text{ when } |\sigma| = \mathbf{I} \\ (\sigma,\, \mathbf{id}(X_i)).\mathbf{I} & = & \sigma.\mathbf{I} \text{ otherwise}\end{array}$$

Definition B.44 (Context length and access) *Redefinition of context length and context access, from definition B.2. Furthermore we define length and element access for environments of contexts. Element access assumes* $\mathbf{I} < |\Phi|$.

$\boxed{|\Phi| = \mathbf{I}}$

$$\begin{array}{lcl}|\bullet| & = & \bullet \\ |\Phi,\, t| & = & |\Phi|,\, \cdot \\ |\Phi,\, X_i| & = & |\Phi|,\, |X_i|\end{array}$$

$\boxed{\Phi.\mathbf{I}}$

$$\begin{array}{lcl}(\Phi,\, t).\mathbf{I} & = & t \text{ when } |\Phi| = \mathbf{I} \\ (\Phi,\, t).\mathbf{I} & = & \Phi.\mathbf{I} \text{ otherwise} \\ (\Phi,\, X_i).\mathbf{I} & = & X_i \text{ when } |\Phi| = \mathbf{I} \\ (\Phi,\, X_i).\mathbf{I} & = & \Phi.\mathbf{I} \text{ otherwise}\end{array}$$

Definition B.45 (Extensions context length and access) *New definition.*

$\boxed{|\Psi|}$

$$\begin{array}{lcl}|\bullet| & = & 0 \\ |\Psi,\, K| & = & |\Psi| + 1\end{array}$$

$\boxed{\Psi.i}$

$$\begin{array}{lcl}(\Psi,\, K).|\Psi| & = & K \\ (\Psi,\, K).i & = & \Psi.i \text{ when } i < |\Psi|\end{array}$$

Definition B.46 (Substitution application) *Extension of substitution application from definition B.5. The application of a substitution to a term is entirely identical as before, with a slight adjustment for the new definitions of variable indexes.*

$\boxed{t \cdot \sigma}$

$$f_{\mathbf{I}} \cdot \sigma \;=\; \sigma.\mathbf{I}$$

$\boxed{\sigma' \cdot \sigma}$

$$(\sigma',\, \mathbf{id}(X_i)) \cdot \sigma \;=\; \sigma' \cdot \sigma,\, \mathbf{id}(X_i)$$

Definition B.47 (Identity substitution) *Redefinition of identity substitution from B.6.*

$$\begin{array}{lcl} \mathsf{id}_{\bullet} & = & \bullet \\ \mathsf{id}_{\Phi,\, t} & = & \mathsf{id}_{\Phi},\, f_{|\Phi|} \\ \mathsf{id}_{\Phi,\, X_i} & = & \mathsf{id}_{\Phi},\, \mathbf{id}(X_i) \end{array}$$

Definition B.48 (Variable limits for terms and substitutions) *Redefinition of the definition B.7.*

$\boxed{t <^f \mathbf{I}}$

$$\begin{array}{lcl} s <^f \mathbf{I} & & \\ c <^f \mathbf{I} & & \\ f_{\mathbf{I}} <^f \mathbf{I}' & \Leftarrow & \mathbf{I} < \mathbf{I}' \\ b_i <^f \mathbf{I} & & \\ (\lambda(t_1).t_2) <^f \mathbf{I} & \Leftarrow & t_1 <^f \mathbf{I} \wedge t_2 <^f \mathbf{I} \\ t_1\, t_2 <^f \mathbf{I} & \Leftarrow & t_1 <^f \mathbf{I} \wedge t_2 <^f \mathbf{I} \\ & & \cdots \end{array}$$

$\boxed{\sigma <^f \mathbf{I}}$

$$\begin{array}{lcl} \bullet <^f \mathbf{I} & & \\ \sigma,\, t <^f \mathbf{I} & \Leftarrow & \sigma <^f \mathbf{I} \wedge t <^f \mathbf{I} \\ \sigma,\, \mathbf{id}(X_i) <^f \mathbf{I} & \Leftarrow & \sigma <^f \mathbf{I} \wedge \exists \mathbf{I}' : (\mathbf{I}',\, |X_i|) \leq \mathbf{I} \end{array}$$

$\boxed{\sigma <^b n}$

$$\sigma,\, \mathbf{id}(\phi_i) <^b n \quad \Leftarrow \quad \sigma <^b n$$

Definition B.49 (Extension of freshening) *This is an extension of definition B.8 and adjustment for indexes. We assume $t <^f \mathbf{I}$ and $\sigma <^f \mathbf{I}$. Also $t <^b n+1$ and $\sigma <^b n+1$.*

$\boxed{\lceil t \rceil^n_{\mathbf{I}}}$

$$\begin{array}{lcl} \lceil b_n \rceil^n_{\mathbf{I}} & = & f_{\mathbf{I}} \\ \lceil b_i \rceil^n_{\mathbf{I}} & = & b_i \end{array}$$

$\boxed{\lceil \sigma \rceil^n_{\mathbf{I}}}$

$$\begin{array}{lcl} \lceil \bullet \rceil^n_{\mathbf{I}} & = & \bullet \\ \lceil \sigma,\, t \rceil^n_{\mathbf{I}} & = & \lceil \sigma \rceil^n_{\mathbf{I}},\, \lceil t \rceil^n_{\mathbf{I}} \\ \lceil \sigma,\, \mathbf{id}(X_i) \rceil^n_{\mathbf{I}} & = & \lceil \sigma \rceil^n_{\mathbf{I}},\, \mathbf{id}(X_i) \end{array}$$

Definition B.50 (Extension of binding) *This is an extension of definition B.9 and adjustment for indexes. We assume $t <^f \mathbf{I}$ and $\sigma <^f \mathbf{I}$. Also $t <^b n$ and $\sigma <^b n$.*

$\boxed{\lfloor t \rfloor^n_{\mathbf{I}}}$

$$\begin{array}{lcl} \lfloor f_{\mathbf{I}'} \rfloor^n_{\mathbf{I}} & = & b_n \text{ when } \mathbf{I} = \mathbf{I}', \cdot \\ \lfloor f_{\mathbf{I}'} \rfloor^n_{\mathbf{I}} & = & f_{\mathbf{I}'} \text{ otherwise} \end{array}$$

$\boxed{\lfloor \sigma \rfloor^n_{\mathbf{I}}}$

$$\begin{array}{lcl} \lfloor \bullet \rfloor^n_{\mathbf{I}} & = & \bullet \\ \lfloor \sigma, t \rfloor^n_{\mathbf{I}} & = & \lfloor \sigma \rfloor^n_{\mathbf{I}}, \lfloor t \rfloor^n_{\mathbf{I}} \\ \lfloor \sigma, \mathbf{id}(X_i) \rfloor^n_{\mathbf{I}} & = & \lfloor \sigma \rfloor^n_{\mathbf{I}}, \mathbf{id}(X_i) \end{array}$$

Definition B.51 (Environment subsumption) *We define what it means for an environment to be a subenvironment (be a prefix of; or be subsumed by) another one.*

$\boxed{\Phi \subseteq \Phi'}$

$$\begin{array}{lcl} \Phi \subseteq \Phi & & \\ \Phi \subseteq \Phi', t & \Leftarrow & \Phi \subseteq \Phi' \\ \Phi \subseteq \Phi', X_i & \Leftarrow & \Phi \subseteq \Phi' \end{array}$$

$\boxed{\Psi \subseteq \Psi'}$

$$\begin{array}{lcl} \Psi \subseteq \Psi & & \\ \Psi \subseteq \Psi', K & \Leftarrow & \Psi \subseteq \Psi' \end{array}$$

Definition B.52 (Substitution subsumption) *We define what it means for an substitution to be a prefix of another one.*

$\boxed{\sigma \subseteq \sigma'}$

$$\begin{array}{lcl} \sigma \subseteq \sigma & & \\ \sigma \subseteq \sigma', t & \Leftarrow & \sigma \subseteq \sigma' \\ \sigma \subseteq \sigma', \mathbf{id}(X_i) & \Leftarrow & \sigma \subseteq \sigma' \end{array}$$

Definition B.53 *The typing judgements defined in B.10 and are redefined as follows.*

1. $\vdash \Sigma$ *wf is adjusted as shown below.*
2. $\vdash_\Sigma \Phi$ *wf is redefined as* $\Psi \vdash_\Sigma \Phi$ *wf, and the rules below are added.*
3. $\Phi \vdash t : t'$ *is redefined as* $\Psi;\ \Phi \vdash t : t'$*, and adjusted as shown below.*
4. $\Phi \vdash \sigma : \Phi'$ *is redefined as* $\Psi;\ \Phi \vdash \sigma : \Phi'$ *and the rules below are added.*
5. $\vdash \Psi$ *wf is defined below.*
6. $\Psi \vdash T : K$ *is defined below.*

$\boxed{\vdash \Sigma \text{ wf}}$

$$\frac{\vdash \Sigma \text{ wf} \qquad \bullet;\, \bullet \vdash t : s \qquad (c :) \notin \Sigma}{\vdash (\Sigma,\, c : t) \text{ wf}}$$

$\boxed{\Psi \vdash_\Sigma \Phi \text{ wf}}$

$$\frac{}{\Psi \vdash \bullet \text{ wf}} \qquad \frac{\Psi \vdash \Phi \text{ wf} \qquad \Psi;\, \Phi \vdash t : s}{\Psi \vdash (\Phi,\, t) \text{ wf}} \qquad \frac{\Psi \vdash \Phi \text{ wf} \qquad \Psi.i = [\Phi]\,\mathsf{ctx}}{\Psi \vdash (\Phi,\, X_i) \text{ wf}}$$

$\boxed{\Psi;\, \Phi \vdash t : t'}$

$$\frac{c : t \in \Sigma}{\Psi;\, \Phi \vdash_\Sigma c : t} \qquad \frac{\Phi.\mathbf{I} = t}{\Psi;\, \Phi \vdash f_{\mathbf{I}} : t} \qquad \frac{\Psi;\, \Phi \vdash t_1 : s \qquad \Psi;\, \Phi,\, t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : s' \qquad (s, s', s'') \in \mathcal{R}}{\Psi;\, \Phi \vdash \Pi(t_1).t_2 : s''}$$

$$\frac{\Psi;\, \Phi \vdash t_1 : s \qquad \Psi;\, \Phi,\, t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : t' \qquad \Psi;\, \Phi \vdash \Pi(t_1).\, \lfloor t' \rfloor_{|\Phi|,\,.} : s'}{\Psi;\, \Phi \vdash \lambda(t_1).t_2 : \Pi(t_1).\, \lfloor t' \rfloor_{|\Phi|,\,.}} \qquad \frac{\Psi;\, \Phi \vdash t_1 : \Pi(t).t' \qquad \Psi;\, \Phi \vdash t_2 : t}{\Psi;\, \Phi \vdash t_1\, t_2 : \lceil t' \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2)}$$

$$\frac{\Psi.i = T \qquad T = [\Phi']\,t' \qquad \Psi;\, \Phi \vdash \sigma : \Phi'}{\Psi;\, \Phi \vdash X_i/\sigma : t' \cdot \sigma}$$

$\boxed{\Psi;\, \Phi \vdash \sigma : \Phi'}$

$$\frac{}{\Psi;\, \Phi \vdash \bullet : \bullet} \qquad \frac{\Psi;\, \Phi \vdash \sigma : \Phi' \qquad \Psi;\, \Phi \vdash t : t' \cdot \sigma}{\Psi;\, \Phi \vdash (\sigma,\, t) : (\Phi',\, t')} \qquad \frac{\Psi;\, \Phi \vdash \sigma : \Phi' \qquad \Psi.i = [\Phi']\,\mathsf{ctx} \qquad \Phi',\, X_i \subseteq \Phi}{\Psi;\, \Phi \vdash (\sigma,\, \mathbf{id}(X_i)) : (\Phi',\, X_i)}$$

$\boxed{\vdash \Psi \text{ wf}}$

$$\frac{}{\vdash \Psi \text{ wf}} \qquad \frac{\vdash \Psi \text{ wf} \qquad \Psi \vdash \Phi \text{ wf}}{\vdash (\Psi,\, [\Phi]\,\mathsf{ctx}) \text{ wf}} \qquad \frac{\vdash \Psi \text{ wf} \qquad \Psi \vdash [\Phi]\,t : [\Phi]\,s}{\vdash (\Psi,\, [\Phi]\,t) \text{ wf}}$$

$\boxed{\Psi \vdash T : K}$

$$\frac{\Psi;\, \Phi \vdash t : t'}{\Psi \vdash [\Phi]\,t : [\Phi]\,t'} \qquad \frac{\Psi \vdash \Phi,\, \Phi' \text{ wf}}{\Psi \vdash [\Phi]\Phi' : [\Phi]\,\mathsf{ctx}}$$

Lemma B.54 (Extension of lemma 2) *1. If $t <^f \mathbf{I}$ and $|\Phi| = \mathbf{I}$ then $t \cdot id_\Phi = t$.*

2. If $\sigma <^f \mathbf{I}$ and $|\Phi| = \mathbf{I}$ then $\sigma \cdot id_\Phi = \sigma$.

Part 1 is proved by induction on $t <^f \mathbf{I}$. The interesting case is $f_{\mathbf{I}'}$, with $\mathbf{I}' < \mathbf{I}$. In this case we have to prove $\mathsf{id}_\Phi.\mathbf{I}' = f_{\mathbf{I}'}$. This is done by induction on $\mathbf{I}' < \mathbf{I}$.
When $\mathbf{I} = \mathbf{I}'$, $\cdot$ we have by inversion of $|\Phi| = \mathbf{I}$ that $\Phi = \Phi'$, t and $|\Phi'| = \mathbf{I}'$. Thus $\mathsf{id}_\Phi = \mathsf{id}\Phi'$, $f_{\mathbf{I}'}$ and thus the desired result.
When $\mathbf{I} = \mathbf{I}'$, $|X_i|$, exactly as above.
When $\mathbf{I} = \mathbf{I}^*$, $\cdot$ and $\mathbf{I}' < \mathbf{I}^*$, we have that $\Phi = \Phi^*$, t and $|\Phi^*| = \mathbf{I}^*$. By (inner) induction hypothesis we get that $\mathsf{id}_{\Phi^*}.\mathbf{I}' = f_{\mathbf{I}'}$. From this directly we get that $\mathsf{id}_\Phi.\mathbf{I}' = f_{\mathbf{I}'}$.
When $\mathbf{I} = \mathbf{I}^*$, $|X_i|$ and $\mathbf{I}' < \mathbf{I}^*$, entirely as the previous case.
Part 2 is trivial to prove by induction and use of part 1 in cases $\sigma = \bullet$ or $\sigma = \sigma'$, t. In the case $\sigma = \sigma'$, $\mathbf{id}(X_i)$ we have: $\sigma' <^f \mathbf{I}$ thus by induction $\sigma' \cdot \mathsf{id}_\Phi = \sigma'$, and furthermore $(\sigma',\, \mathbf{id}(X_i)) \cdot \mathsf{id}_\Phi = \sigma$.

Lemma B.55 (Length of subcontexts) *If* $\Phi \subseteq \Phi'$ *then* $|\Phi| \leq |\Phi'|$.

Trivial by induction on $\Phi \subseteq \Phi'$.

Lemma B.56 (Variable limits can be increased) *1. If* $t <^f \mathbf{I}$ *and* $\mathbf{I} < \mathbf{I}'$ *then* $t <^f \mathbf{I}'$

2. If $t <^b n$ *and* $n < n'$ *then* $t <^b n'$

3. If $\sigma <^f \mathbf{I}$ *and* $\mathbf{I} < \mathbf{I}'$ *then* $\sigma <^f \mathbf{I}'$

4. If $\sigma <^b n$ *and* $n < n'$ *then* $\sigma <^b n'$

Trivial by induction on t or σ.

Lemma B.57 (Extension of lemma 2) *1. If* $\Psi;\ \Phi \vdash t : t'$ *then* $t <^f |\Phi|$ *and* $t <^b 0$.

2. If $\Psi;\ \Phi \vdash \sigma : \Phi'$ *then* $\sigma <^f |\Phi|$, $\sigma <^b 0$ *and* $|\sigma| = |\Phi'|$.

Part 1 is proved similarly as before.
Part 2 needs to account for the new case $\sigma = \sigma^*,\ \mathbf{id}(X_i)$.
By inversion of typing for σ we get that $\Phi' = \Phi^*,\ X_i$ with $\sigma^* : \Phi^*$. By induction we get that $\sigma^* <^f |\Phi^*|$. Again by inversion of typing for σ we get that $\Phi^*,\ X_i \subseteq \Phi$. Thus $\sigma^* <^f |\Phi|$ by use of lemma B.56. Furthermore from $\Phi^*,\ X_i \subseteq \Phi$ and lemma B.55 we get that $|\Phi^*|,\ |X_i| \leq |\Phi|$. Thus for $\mathbf{I}' = |\Phi^*|$ we have $\mathbf{I}',\ |X_i| < |\Phi|$ thus we overall get $\sigma <^f |\Phi|$.
Furthermore the other two parts of the theorem are trivial from induction hypothesis.

Lemma B.58 (Extension of lemma B.13) *If* $\Psi \vdash \Phi$ *wf then for any* Φ' *such that* $\Phi \subseteq \Phi'$ *and* $\Psi \vdash \Phi'$ *wf, we have that* $\Psi;\ \Phi' \vdash id_\Phi : \Phi$.

Similar to the original proof. The new case for $\Phi = \Phi',\ X_i$ works as follows. By induction hypothesis for Φ' we get that $\Psi;\ \Phi',\ X_i \vdash \mathsf{id}_{\Phi'} : \Phi'$. Now for any environment Φ^* such that $\Phi',\ X_i \subseteq \Phi^*$, by using the typing rule for $\mathbf{id}(X_i)$, we get the desired.

Lemma B.59 (Extension of lemma B.14) *If* $\Psi \vdash \Phi$ *wf and* $|\Phi| = \mathbf{I}$ *then for all* $\mathbf{I}' < \mathbf{I}$ *with* $\Phi.\mathbf{I}' = t$, *we have* $\Phi.\mathbf{I}' <^f \mathbf{I}$.

Identical as before.

Lemma B.60 (Extension of lemmas B.15 and B.15) *1. If* $t <^f \mathbf{I}$, $|\sigma| = \mathbf{I}$, $t \cdot \sigma = t'$ *and* $\sigma \subseteq \sigma'$ *then* $t \cdot \sigma' = t'$.

2. If $\sigma <^f \mathbf{I}$, $|\sigma'| = \mathbf{I}$, $\sigma \cdot \sigma' = \sigma_r$ *and* $\sigma \subseteq \sigma''$ *then* $\sigma \cdot \sigma'' = \sigma_r$.

Part 1 is identical as before. In part 2, in case $\sigma = \sigma,\ \mathbf{id}(X_i)$, proved trivially by definition of substitution application.

Lemma B.61 (Extension of lemma B.16) *If* $\Psi \vdash \Phi$ *wf,* $\Phi.\mathbf{I} = t$ *and* $\Psi;\ \Phi' \vdash \sigma : \Phi$, *then* $\Psi;\ \Phi' \vdash \sigma.\mathbf{I} : t \cdot \sigma$.

The proof proceeds by structural induction on the typing derivation for σ as before. In case $\sigma = \sigma^*,\ \mathbf{id}(X_i)$, we have that $(\Phi^*,\ X_i) \subseteq \Phi'$. We have that $\Phi^*.\mathbf{I} = \Phi.\mathbf{I} = t$ (since $\mathbf{I} \not= |\Phi^*|$, because $(\Phi^*,\ X_i).|\Phi| \not= t$). Thus from induction hypothesis for σ^* we get that $\Psi;\ \Phi' \vdash \sigma^*.\mathbf{I} : t \cdot \sigma^*$. Using lemma B.60 and also the fact that $\sigma.\mathbf{I} = \sigma^*.\mathbf{I}$, we get that $\Psi;\ \Phi' \vdash \sigma.\mathbf{I} : t \cdot \sigma$.

Lemma B.62 (Extension of lemma B.17) *1. If* $t <^f \mathbf{I}$, $t <^b n+1$, $\sigma <^f \mathbf{I}'$ *and* $|\sigma| = \mathbf{I}$ *then* $\lceil t \cdot \sigma \rceil^n_{\mathbf{I}'} = \lceil t \rceil^n_{\mathbf{I}} \cdot (\sigma, f_{\mathbf{I}'})$.

2. If $\sigma' <^f \mathbf{I}$, $\sigma' <^b n+1$, $\sigma <^f \mathbf{I}'$ *and* $|\sigma| = \mathbf{I}$ *then* $\lceil \sigma' \cdot \sigma \rceil^n_{\mathbf{I}'} = \lceil \sigma' \rceil^n_{\mathbf{I}} \cdot (\sigma, f_{\mathbf{I}'})$.

Part 1 is entirely similar as before, with slight adjustments to account for the new type of indices. Part 2 needs to account for the new case of $\sigma' = \sigma'', \mathbf{id}(X_i)$, which is entirely trivial based on the definition.

Lemma B.63 (Extension of lemma B.18) *1. If* $t <^f \mathbf{I}$, $\cdot$, $t <^b n$, $\sigma <^f \mathbf{I}'$ *and* $|\sigma| = \mathbf{I}$ *then* $\lfloor t \cdot (\sigma, f_{\mathbf{I}'}) \rfloor^n_{\mathbf{I}', \cdot} = \lfloor t \rfloor^n_{\mathbf{I}', \cdot} \cdot \sigma$.

2. If $\sigma' <^f \mathbf{I}$, $\cdot$, $\sigma' <^b n$, $\sigma <^f \mathbf{I}'$ *and* $|\sigma| = m$ *then* $\lfloor \sigma' \cdot (\sigma, f_{\mathbf{I}'}) \rfloor^n_{\mathbf{I}', \cdot} = \lfloor \sigma' \rfloor^n_{\mathbf{I}', \cdot} \cdot \sigma$.

Similarly to the above.

Lemma B.64 (Extension of lemma B.19) *1. If* $t <^f \mathbf{I}$, $|\sigma| = \mathbf{I}$, $\sigma <^f \mathbf{I}'$ *and* $|\sigma'| = \mathbf{I}'$ *then* $(t \cdot \sigma) \cdot \sigma' = t \cdot (\sigma \cdot \sigma')$.

2. If $\sigma_1 <^f \mathbf{I}$, $|\sigma| = \mathbf{I}$, $\sigma <^f \mathbf{I}'$ *and* $|\sigma'| = \mathbf{I}'$ *then* $(\sigma_1 \cdot \sigma) \cdot \sigma' = \sigma_1 \cdot (\sigma \cdot \sigma')$.

Part 1 is identical as before. Part 2 needs to account for the case where $\sigma_1 = \sigma'_1, \mathbf{id}(X_i)$, which is entirely trivial.

Lemma B.65 (Extension of lemma B.20) *If* $|\sigma| = \mathbf{I}$ *and* $|\Phi| = \mathbf{I}$ *then* $id_\Phi \cdot \sigma = \sigma$.

We need to account for the new case of $\Phi = \Phi', X_i$. In that case, $\mathsf{id}_{\Phi', X_i} = \mathsf{id}_{\Phi'}, \mathbf{id}(X_i)$. By inversion of $|\sigma| = \mathbf{I} = |\Phi'|, |X_i|$ we get that $\sigma = \sigma', \mathbf{id}(X_i)$. By induction hypothesis we get $\mathsf{id}_{\Phi'} \cdot \sigma' = \sigma'$. By lemma B.60 we get $\mathsf{id}_{\Phi'} \cdot \sigma = \sigma'$. Last it is trivial to see that $(\mathsf{id}_{\Phi'}, \mathbf{id}(X_i)) \cdot \sigma = \sigma', \mathbf{id}(X_i) = \sigma$.

Lemma B.66 (Extension of lemma B.21) *1. If* $\lceil t \rceil^n_{\mathbf{I}} = \lceil t' \rceil^n_{\mathbf{I}}$ *then* $t = t'$.

2. If $\lceil \sigma \rceil^n_{\mathbf{I}} = \lceil \sigma' \rceil^n_{\mathbf{I}}$ *then* $\sigma = \sigma'$.

Part 1 is identical as before; part 2 holds trivially for the new case of σ.

Theorem B.67 (Extension of main substitution theorem B.22) *1. If* $\Psi;\ \Phi \vdash t : t'$ *and* $\Psi;\ \Phi' \vdash \sigma : \Phi$ *then* $\Psi;\ \Phi' \vdash t \cdot \sigma : t' \cdot \sigma$.

2. If $\Psi;\ \Phi' \vdash \sigma : \Phi$ *and* $\Psi;\ \Phi'' \vdash \sigma' : \Phi'$ *then* $\Psi;\ \Phi'' \vdash \sigma \cdot \sigma' : \Phi$.

3. If $\Psi \vdash [\Phi']t : [\Phi']t'$ *and* $\Psi;\ \Phi \vdash \sigma : \Phi'$ *then* $\Psi \vdash [\Phi]t \cdot \sigma : [\Phi]t' \cdot \sigma$.

Part 1 is identical as before; all the needed theorems were adjusted above, so the new form of indexes does not change the proof at all. The only case that needs adjustment is the metavariables case.

$$\textbf{Case}\ \frac{\Psi.i = T \qquad T = [\Phi']t' \qquad \Psi;\ \Phi \vdash \sigma : \Phi'}{\Psi;\ \Phi \vdash X_i/\sigma_0 : t'}\ \triangleright$$

From $\Psi;\ \Phi \vdash X_i/\sigma_0 : t'$ we get that $\Psi.i = [\Phi_0]t_0$, $\Psi;\ \Phi \vdash \sigma_0 : \Phi_0$ and $t' = t_0 \cdot \sigma_0$.
Applying the second part of the lemma for $\sigma = \sigma_0$ and $\sigma' = \sigma$ we get that $\Psi;\ \Phi' \vdash \sigma_0 \cdot \sigma' : \Phi_0$.
Thus applying the same typing rule for $t = X_i/(\sigma_0 \cdot \sigma)$ we get that $\Psi;\ \Phi' \vdash X_i/(\sigma_0 \cdot \sigma') : t_0 \cdot (\sigma_0 \cdot \sigma')$.
Taking into account the definition of $\cdot$ and also lemma B.64, we have that this is the desired result.

For the second part, we need to account for the new case of substitutions.

$$\textbf{Case}\ \frac{\Psi;\ \Phi' \vdash \sigma : \Phi_0 \qquad \Psi.i = [\Phi_0]\,\mathsf{ctx} \qquad \Phi_0,\ X_i \subseteq \Phi'}{\Psi;\ \Phi' \vdash (\sigma,\ \mathbf{id}(X_i)) : (\Phi_0,\ X_i)}\ \triangleright$$

By induction hypothesis for σ, we get: $\Psi;\ \Phi'' \vdash \sigma \cdot \sigma' : \Phi_0$.
We need to prove that $(\Phi_0,\ X_i) \subseteq \Phi''$.

We have that $\Psi;\ \Phi'' \vdash \sigma' : \Phi'$.
By induction on $(\Phi_0,\ X_i) \subseteq \Phi'$ and repeated inversions of σ' we arrive at a $\sigma'' \subseteq \sigma'$ such that:
$\Psi;\ \Phi'' \vdash \sigma'' : \Phi_0,\ X_i$
By inversion of this we get that $(\Phi_0,\ X_i) \subseteq \Phi''$.
Thus, using the same typing rule, we get $\Psi;\ \Phi'' \vdash (\sigma \cdot \sigma', \mathbf{id}(X_i)) : (\Phi_0,\ X_i)$, which is the desired.

For the third part, the proof is identical as before.

Lemma B.68 (Extension of lemma B.24) *If $\Psi;\ \Phi \vdash t : t'$ then either $t' = \mathsf{Type}'$ or $\Psi;\ \Phi \vdash t' : s$.*

Identical as before.

Lemma B.69 (Extension of the lemma B.25) *1. If $\Psi;\ \Phi \vdash t : t'$ and $\Phi \subseteq \Phi'$ then $\Psi;\ \Phi' \vdash t : t'$.*
2. If $\Psi;\ \Phi \vdash \sigma : \Phi''$ and $\Phi \subseteq \Phi'$ then $\Psi;\ \Phi' \vdash \sigma : \Phi''$.

Identical as before.

Lemma B.70 (Adaptation of lemma 4) *1. If $\Psi;\Phi \vdash t : t'$ and $\Psi \subseteq \Psi'$ then $\Psi';\ \Phi \vdash t : t'$.*
2. If $\Psi;\Phi \vdash \sigma : \Phi'$ and $\Psi \subseteq \Psi'$ then $\Psi';\ \Phi \vdash \sigma : \Phi'$.
3. If $\Psi \vdash \Phi$ wf and $\Psi \subseteq \Psi'$ then $\Psi' \vdash \Phi$ wf.
4. If $\Psi \vdash T : K$ and $\Psi \subseteq \Psi'$ then $\Psi' \vdash T : K$.

Parts 2 and 3 are trivial for the new cases; otherwise identical as before.

Now we have proved the fundamentals. We proceed to define substitutions for the extension variables (meta- and context-variables), typing for such substitutions, and prove an extensions substitution theorem.

Definition B.71 (Substitutions of extension variables) *The syntax of substitutions for meta- and context-variables is given below.*

$$\sigma_\Psi ::= \bullet \mid \sigma_\Psi,\ T$$

Definition B.72 (Context, substitution, index concatenation) *We define what it means to concatenate one context (substitution, index) to another.*

$\boxed{\Phi,\ \Phi'}$

$$\begin{array}{lcl} \Phi,\ (\bullet) & = & \Phi \\ \Phi,\ (\Phi',\ t) & = & (\Phi,\ \Phi'),\ t \\ \Phi,\ (\Phi',\ X_i) & = & (\Phi,\ \Phi'),\ X_i \end{array}$$

$\boxed{\sigma,\ \sigma'}$

$$\begin{array}{lcl} \sigma,\ (\bullet) & = & \sigma \\ \sigma,\ (\sigma',\ t) & = & (\sigma,\ \sigma'),\ t \\ \sigma,\ (\sigma',\ \mathbf{id}(X_i)) & = & (\sigma,\ \sigma'),\ \mathbf{id}(X_i) \end{array}$$

$\boxed{I,\, I'}$

$$\begin{array}{lcl} \mathbf{I},\, (\bullet) & = & \mathbf{I} \\ \mathbf{I},\, (\mathbf{I}',\, \cdot) & = & (\mathbf{I},\, \mathbf{I}'),\, \cdot \\ \mathbf{I},\, (\mathbf{I}',\, |X_i|) & = & (\mathbf{I},\, \mathbf{I}'),\, |X_i| \end{array}$$

Definition B.73 (Partial identity substitution) *We define what partial identity substitutions (for a suffix of a context) are.*

$\boxed{\mathsf{id}_{[\Phi]\,\Phi'}}$

$$\begin{array}{lcl} \mathsf{id}_{[\Phi]\,\bullet} & = & \bullet \\ \mathsf{id}_{[\Phi]\,\Phi',\,t} & = & \mathsf{id}_{[\Phi]\,\Phi'},\, f_{|\Phi|+|\Phi'|} \\ \mathsf{id}_{[\Phi]\,\Phi',\,X_i} & = & \mathsf{id}_{[\Phi]\,\Phi'},\, \mathbf{id}(X_i) \end{array}$$

Definition B.74 (Extensions substitution length and access) *Defined below.*

$\boxed{|\sigma_\Psi|}$

$$\begin{array}{lcl} |\bullet| & = & 0 \\ |\sigma_\Psi,\, T| & = & 1+|\sigma_\Psi| \end{array}$$

Definition B.75 (Extension substitution and context concatenation) *We define concatenation of extension substitutions and extensions contexts below.*

$\boxed{\Psi,\, \Psi'}$

$$\begin{array}{lcl} \Psi,\, (\bullet) & = & \Psi \\ \Psi,\, (\Psi',\, K) & = & (\Psi,\, \Psi'),\, K \end{array}$$

$\boxed{\sigma_\Psi,\, \sigma'_\Psi}$

$$\begin{array}{lcl} \sigma_\Psi,\, (\bullet) & = & \sigma_\Psi \\ \sigma_\Psi,\, (\sigma'_\Psi,\, T) & = & (\sigma_\Psi,\, \sigma'_\Psi),\, T \end{array}$$

Definition B.76 (Extensions substitution subsumption) *Defined below.*

$\boxed{\sigma_\Psi \subseteq \sigma'_\Psi}$

$$\begin{array}{l} \sigma_\Psi \subseteq \sigma_\Psi \\ \sigma_\Psi \subseteq \sigma'_\Psi,\, T \quad \Leftarrow \quad \sigma_\Psi \subseteq \sigma'_\Psi \end{array}$$

Definition B.77 (Application of extensions substitution) *This is an adaptation of definition B.28.*

$\boxed{\mathbf{I}\cdot\sigma\Psi}$

$$
\begin{array}{rlcl}
 & \bullet\cdot\sigma\Psi & = & \bullet \\
 & (\mathbf{I},\,\cdot)\cdot\sigma\Psi & = & (\mathbf{I}\cdot\sigma\Psi),\,\cdot \\
* & (\mathbf{I},\,|X_i|)\cdot\sigma\Psi & = & (\mathbf{I}\cdot\sigma\Psi),\,|\Phi'| \text{ when } \sigma\Psi.i = [\Phi]\Phi'
\end{array}
$$

$\boxed{t\cdot\sigma\Psi}$

$$
\begin{array}{rcl}
f_{\mathbf{I}}\cdot\sigma\Psi & = & f_{\mathbf{I}\cdot\sigma\Psi} \\
(X_i/\sigma)\cdot\sigma\Psi & = & t\cdot(\sigma\cdot\sigma\Psi) \text{ when } \sigma\Psi.i = [\Phi]t
\end{array}
$$

$\boxed{\sigma\cdot\sigma\Psi}$

$$
\begin{array}{rlcl}
 & \bullet\cdot\sigma\Psi & = & \bullet \\
 & (\sigma,\,t)\cdot\sigma\Psi & = & \sigma\cdot\sigma\Psi,\,t\cdot\sigma\Psi \\
* & (\sigma,\,\mathbf{id}(X_i))\cdot\sigma\Psi & = & \sigma\cdot\sigma\Psi,\,\mathsf{id}_{\sigma\Psi.i} \text{ when } \sigma\Psi.i = [\Phi]\Phi'
\end{array}
$$

$\boxed{\Phi\cdot\sigma\Psi}$

$$
\begin{array}{rcl}
\bullet\cdot\sigma\Psi & = & \bullet \\
(\Phi,\,t)\cdot\sigma\Psi & = & \Phi\cdot\sigma\Psi,\,t\cdot\sigma\Psi \\
(\Phi,\,X_i)\cdot\sigma\Psi & = & \Phi\cdot\sigma\Psi,\,\Phi' \text{ when } \sigma\Psi.i = [\Phi]\Phi'
\end{array}
$$

$\boxed{T\cdot\sigma\Psi}$

$$
\begin{array}{rcl}
([\Phi]t)\cdot\sigma\Psi & = & [\Phi\cdot\sigma\Psi]\,(t\cdot\sigma\Psi) \\
([\Phi]\Phi')\cdot\sigma\Psi & = & [\Phi\cdot\sigma\Psi](\Phi'\cdot\sigma\Psi)
\end{array}
$$

$\boxed{K\cdot\sigma\Psi}$

$$
\begin{array}{rcl}
([\Phi]t)\cdot\sigma\Psi & = & [\Phi\cdot\sigma\Psi]\,(t\cdot\sigma\Psi) \\
([\Phi]\,\mathsf{ctx})\cdot\sigma\Psi & = & [\Phi\cdot\sigma\Psi]\,\mathsf{ctx}
\end{array}
$$

$\boxed{\sigma\Psi\cdot\sigma'_\Psi}$

$$
\begin{array}{rcl}
\bullet\cdot\sigma'_\Psi & = & \bullet \\
(\sigma\Psi,\,T)\cdot\sigma'_\Psi & = & \sigma\Psi\cdot\sigma'_\Psi,\,T\cdot\sigma'_\Psi
\end{array}
$$

Definition B.78 (Application of extended substitution to open extended context) *Assuming that Ψ' does not include variables bigger than $X_{|\Psi|}$, we have:*

$\boxed{\Psi'\cdot\sigma\Psi}$

$$
\begin{array}{rcl}
\bullet\cdot\sigma\Psi & = & \bullet \\
(\Psi',\,K)\cdot\sigma\Psi & = & \Psi'\cdot\sigma\Psi,\,K\cdot(\sigma\Psi,\,X_{|\Psi|},\,\cdots,X_{|\Psi|+|\Psi'|})
\end{array}
$$

Definition B.79 (Identity extension substitution) *The identity substitution for extension contexts is defined below.*

$\boxed{\mathsf{id}_\Psi}$

$$\begin{array}{lcl} \mathsf{id}_\bullet & = & \bullet \\ \mathsf{id}_{\Psi,\,K} & = & \mathsf{id}_\Psi,\, X_{|\Psi|} \end{array}$$

Definition B.80 (Extensions substitution typing) *The typing judgement for extensions substitutions is redefined as* $\Psi \vdash \sigma_\Psi : \Psi'$. *The rules are given below. We also define typing for open extension contexts.*

$\boxed{\Psi \vdash \sigma_\Psi : \Psi'}$

$$\frac{}{\Psi \vdash \bullet : \bullet} \qquad \frac{\Psi \vdash \sigma_\Psi : \Psi' \qquad \Psi \vdash T : K \cdot \sigma_\Psi}{\Psi \vdash (\sigma_\Psi,\, T) : (\Psi',\, K)}$$

$\boxed{\Psi \vdash \Psi' \text{ wf}}$

$$\frac{\vdash \Psi,\, \Psi' \text{ wf}}{\Psi \vdash \Psi' \text{ wf}}$$

Lemma B.81 (Interaction of extensions substitution and length) *1.* $|\sigma| \cdot \sigma_\Psi = |\sigma \cdot \sigma_\Psi|$
2. $|\Phi| \cdot \sigma_\Psi = |\Phi \cdot \sigma_\Psi|$

By induction on σ and Φ.

Lemma B.82 (Interaction of environment subsumption and length) *If* $\Phi \subseteq \Phi'$ *then* $|\Phi| \leq |\Phi'|$.

By induction on $\Phi \subseteq \Phi'$.

Lemma B.83 (Interaction of environment subsumption and extensions substitution) *If* $\Phi \subseteq \Phi'$ *then* $\Phi \cdot \sigma_\Psi \subseteq \Phi' \cdot \sigma_\Psi$.

By induction on $\Phi \subseteq \Phi'$.

Lemma B.84 (Interaction of extensions substitution and element access) *1.* $(\sigma.\mathbf{I}) \cdot \sigma_\Psi = (\sigma \cdot \sigma_\Psi).\mathbf{I} \cdot \sigma_\Psi$
2. $(\Phi.\mathbf{I}) \cdot \sigma_\Psi = (\Phi \cdot \sigma_\Psi).\mathbf{I} \cdot \sigma_\Psi$

By induction on $\mathbf{I}$ and taking into account the implicit assumption that $\mathbf{I} < |\sigma|$ or $\mathbf{I} < |\Phi|$.

Lemma B.85 (Extension of lemma B.30) *If* $\Psi \vdash \sigma_\Psi : \Psi'$ *and* $\sigma_\Psi.i = [\Phi]\,t$ *then* $t <^f |\Phi|$ *and* $t <^b 0$.

Identical as before.

Lemma B.86 (Extension of lemma B.31) *If* $t <^b n$ *then* $\lceil t \cdot \sigma \rceil_m^n = t \cdot \lceil \sigma \rceil_m^n$.

Identical as before.

Lemma B.87 (Extension of lemma B.32) *1. If* $\Psi \vdash \sigma_\Psi : \Psi'$ *then* $\lceil t \rceil_{\mathbf{I}}^n \cdot \sigma_\Psi = \lceil t \cdot \sigma_\Psi \rceil_{\mathbf{I} \cdot \sigma_\Psi}^n$
2. If $\Psi \vdash \sigma_\Psi : \Psi'$ *then* $\lceil \sigma \rceil_{\mathbf{I}}^n \cdot \sigma_\Psi = \lceil \sigma \cdot \sigma_\Psi \rceil_{\mathbf{I} \cdot \sigma_\Psi}^n$

Part 1 is proved by induction on t.
In the case $t = b_n$, we have that the left-hand side is equal to $f_{\mathbf{I}} \cdot \sigma_\Psi = f_{\mathbf{I}\cdot\sigma_\Psi}$. The right-hand side is equal to $\lceil b_n \rceil^n_{\mathbf{I}\cdot\sigma_\Psi} = f_{\mathbf{I}\cdot\sigma_\Psi}$.
In the case $t = X_i/\sigma$, this is proved entirely as before, with trivial changes to account for the new indexes.
Part 2 is proved by induction on σ, as previously. For the new case $\sigma = \sigma'$, $\mathbf{id}(X_i)$, the result is trivial.

Lemma B.88 (Extension of lemma B.33) *If $t <^b n$ then $\lfloor t \cdot \sigma \rfloor^n_{\mathbf{I}} = t \cdot \lfloor \sigma \rfloor^n_{\mathbf{I}}$.*

Identical as before.

Lemma B.89 (Extension of lemma B.34) *1. If $\Psi \vdash \sigma_\Psi : \Psi'$ then $\lfloor t \rfloor^n_{\mathbf{I}} \cdot \sigma_\Psi = \lfloor t \cdot \sigma_\Psi \rfloor^n_{\mathbf{I}\cdot\sigma_\Psi}$*
2. If $\Psi \vdash \sigma_\Psi : \Psi'$ then $\lfloor \sigma \rfloor^n_{\mathbf{I}} \cdot \sigma_\Psi = \lfloor \sigma \cdot \sigma_\Psi \rfloor^n_{\mathbf{I}\cdot\sigma_\Psi}$

Proved similarly to lemma B.87.
When $t = f_{\mathbf{I}}$, we have that the left-hand side is equal to b_n, while the right-hand side is equal to $\lfloor f_{\mathbf{I}\cdot\sigma_\Psi} \rfloor^n_{\mathbf{I}\cdot\sigma_\Psi} = b_n$.

Lemma B.90 (Extension of lemma B.35) *1.* $(t \cdot \sigma) \cdot \sigma_\Psi = (t \cdot \sigma_\Psi) \cdot (\sigma \cdot \sigma_\Psi)$
2. $(\sigma \cdot \sigma') \cdot \sigma_\Psi = (\sigma \cdot \sigma_\Psi) \cdot (\sigma' \cdot \sigma_\Psi)$

Part 1 is entirely similar as before, with the exception of case $t = f_{\mathbf{I}}$. This is proved using the lemma B.84. Part 2 is trivially proved for the new case of σ.

Lemma B.91 (Extension of lemma B.36) $id_\Phi \cdot \sigma_\Psi = id_{\Phi\cdot\sigma_\Psi}$

By induction on Φ.
When $\Phi = \bullet$, trivial.
When $\Phi = \Phi'$, t, by induction we have $\mathsf{id}_{\Phi'} \cdot \sigma_\Psi = \mathsf{id}_{\Phi'\cdot\sigma_\Psi}$. Thus $(\mathsf{id}_{\Phi'},\ f_{|\Phi'|}) \cdot \sigma_\Psi = \mathsf{id}_{\Phi'\cdot\sigma_\Psi},\ f_{|\Phi'|\cdot\sigma_\Psi} = \mathsf{id}_{\Phi'\cdot\sigma_\Psi},\ f_{|\Phi'\cdot\sigma_\Psi|} = \mathsf{id}_{\Phi\cdot\sigma_\Psi}$.
When $\Phi = \Phi'$, X_i, we have that $\mathsf{id}_{\Phi'\cdot\sigma_\Psi}, \mathsf{id}_{\sigma_\Psi.i} = \mathsf{id}_{\Phi'\cdot\sigma_\Psi,\ \sigma_\Psi.i}$ (by simple induction on $\Phi'' = \sigma_\Psi.i$).

Lemma B.92 (Extension of lemma B.37) *1. If $\Psi;\ \Phi \vdash t : t'$, $|\sigma_\Psi| = |\Psi|$ and $\sigma_\Psi \subseteq \sigma'_\Psi$ then $t \cdot \sigma'_\Psi = t \cdot \sigma_\Psi$.*
2. If $\Psi;\ \Phi \vdash \sigma : \Phi'$, $|\sigma_\Psi| = |\Psi|$ and $\sigma_\Psi \subseteq \sigma'_\Psi$ then $\sigma \cdot \sigma'_\Psi = \sigma \cdot \sigma_\Psi$.
3. If $\Psi \vdash \Phi$ wf, $|\sigma_\Psi| = |\Psi|$ and $\sigma_\Psi \subseteq \sigma'_\Psi$ then $\Phi \cdot \sigma'_\Psi = \Phi \cdot \sigma_\Psi$.
4. If $\Psi \vdash T : K$, $|\sigma_\Psi| = |\Psi|$ and $\sigma_\Psi \subseteq \sigma'_\Psi$ then $T \cdot \sigma'_\Psi = T \cdot \sigma_\Psi$.
5. If $K \cdot \sigma_\Psi$ is well-defined, and $\sigma_\Psi \subseteq \sigma'_\Psi$, then $K \cdot \sigma_\Psi = K \cdot \sigma'_\Psi$.
6. If $\Psi \cdot \sigma_\Psi$ is well-defined, and $\sigma_\Psi \subseteq \sigma'_\Psi$, then $\Psi \cdot \sigma_\Psi = \Psi \cdot \sigma'_\Psi$.

Parts 2 and 3 are trivially extended for the new cases; others are identical or easily provable by induction.

Lemma B.93 (Extension of lemma B.38) *If $\vdash \Psi$ wf and $\Psi \vdash \sigma_\Psi : \Psi'$ then $\Psi \vdash \sigma_\Psi.i : \Psi'.i \cdot \sigma_\Psi$.*

By induction on σ_Ψ and then cases on $i < |\sigma_\Psi|$.
If $i = |\sigma_\Psi| - 1$ then proceed by cases for σ_Ψ.
If $\sigma_\Psi = \bullet$, then the case is impossible.
If $\sigma_\Psi = \sigma'_\Psi,\ [\Phi]t$, we have by typing inversion for σ_Ψ that $\Psi \vdash [\Phi]t : (\Psi'.i) \cdot \sigma'_\Psi$, which by lemma B.92 is equal to the desired.
If $\sigma_\Psi = \sigma'_\Psi,\ [\Phi]\Phi'$, we get by typing inversion for σ_Ψ that $\Psi \vdash [\Phi]\Phi' : [\Psi'.i \cdot \sigma'_\Psi]\,\mathsf{ctx}$ which again by lemma B.92 is the desired.
If $i < |\sigma_\Psi| - 1$ then by inversion of σ_Ψ we have that either $\sigma_\Psi = \sigma'_\Psi,\ [\Phi]t$ or $\sigma_\Psi = \sigma'_\Psi,\ [\Phi]\Phi'$. In both cases $i < |\sigma'_\Psi| - 1$ so by induction hypothesis get $\sigma'_\Psi.i : \Psi'.i \cdot \sigma'_\Psi$ which, using B.92, is the desired.

Lemma B.94 (Interaction of two extension substitutions) *1.* $(\mathbf{I}\cdot\sigma_\Psi)\cdot\sigma'_\Psi = \mathbf{I}\cdot(\sigma_\Psi\cdot\sigma'_\Psi)$

2. $(t\cdot\sigma_\Psi)\cdot\sigma'_\Psi = t\cdot(\sigma_\Psi\cdot\sigma'_\Psi)$

3. $(\Phi\cdot\sigma_\Psi)\cdot\sigma'_\Psi = \Phi\cdot(\sigma_\Psi\cdot\sigma'_\Psi)$

4. $(\sigma\cdot\sigma_\Psi)\cdot\sigma'_\Psi = \sigma\cdot(\sigma_\Psi\cdot\sigma'_\Psi)$

5. $(T\cdot\sigma_\Psi)\cdot\sigma'_\Psi = T\cdot(\sigma_\Psi\cdot\sigma'_\Psi)$

6. $(K\cdot\sigma_\Psi)\cdot\sigma'_\Psi = K\cdot(\sigma_\Psi\cdot\sigma'_\Psi)$

7. $(\Psi\cdot\sigma_\Psi)\cdot\sigma'_\Psi = \Psi\cdot(\sigma_\Psi\cdot\sigma'_\Psi)$

Part 1 By induction on $\mathbf{I}$. The interesting case is $\mathbf{I} = \mathbf{I}'$, X_i. In that case we have $(\mathbf{I}\cdot\sigma_\Psi)\cdot\sigma'_\Psi = (\mathbf{I}'\cdot\sigma_\Psi)\cdot\sigma'_\Psi,\ \sigma_\Psi.i\cdot\sigma'_\Psi$. Trivially $\sigma_\Psi.i\cdot\sigma'_\Psi = (\sigma_\Psi\cdot\sigma'_\Psi).i$, and also using induction hypothesis, we have that the above is further equal to $\mathbf{I}'\cdot(\sigma_\Psi\cdot\sigma'_\Psi),\ (\sigma_\Psi\cdot\sigma'_\Psi).i$, which is exactly the desired.

Part 2 By induction on t. The interesting case is $t = X_i/\sigma$. The left-hand-side is then equal to $(\sigma_\Psi.i\cdot(\sigma\cdot\sigma_\Psi))\cdot\sigma'_\Psi$, with $\sigma_\Psi.i = [\Phi]t$. This is further rewritten as $(t\cdot(\sigma\cdot\sigma_\Psi))\cdot\sigma'_\Psi = (t\cdot\sigma'_\Psi)\cdot((\sigma\cdot\sigma_\Psi)\cdot\sigma'_\Psi)$ through lemma B.90. Furthermore through part 4 we get that this is equal to $(t\cdot\sigma'_\Psi)\cdot(\sigma\cdot(\sigma_\Psi\cdot\sigma'_\Psi))$.
The right-hand-side is written as: $(X_i/\sigma)\cdot(\sigma_\Psi\cdot\sigma'_\Psi)$. We have that $(\sigma_\Psi\cdot\sigma'_\Psi).i = (\sigma_\Psi.i)\cdot\sigma'_\Psi = [\Phi\cdot\sigma'_\Psi](t\cdot\sigma'_\Psi)$. Thus $(X_i/\sigma)\cdot(\sigma_\Psi\cdot\sigma'_\Psi) = (t\cdot\sigma'_\Psi)\cdot(\sigma\cdot(\sigma_\Psi\cdot\sigma'_\Psi))$.

Part 3 By induction on Φ. When $\Phi = \Phi,\ X_i$, we have that the left-hand-side is equal to $(\Phi\cdot\sigma_\Psi)\cdot\sigma'_\Psi,\ \Phi'\cdot\sigma'_\Psi$ with $\sigma_\Psi.i = [\Phi]\Phi'$. By induction hypothesis this is further equal to $\Phi\cdot(\sigma_\Psi\cdot\sigma'_\Psi),\ \Phi'\cdot\sigma'_\Psi$.
Also, we have that $(\sigma_\Psi\cdot\sigma'_\Psi).i = [\Phi\cdot\sigma'_\Psi]\,\Phi'\cdot\sigma'_\Psi$. Thus the right-hand-side is equal to $\Phi\cdot(\sigma_\Psi\cdot\sigma'_\Psi),\ \Phi'\cdot\sigma'_\Psi$, which is exactly equal to the left-hand-side.

Rest Similarly as above.

Lemma B.95 (Interaction of identity substitution and extension substitution) *If* $|\sigma_\Psi| = |\Psi|$ *then* $id_\Psi\cdot\sigma_\Psi = \sigma_\Psi$

By induction on Ψ. If $\Psi = \bullet$, trivial. If $\Psi = \Psi',\ K$ then $\mathsf{id}_{\Psi',K}\cdot\sigma_\Psi = (\mathsf{id}_{\Psi'},\ X_{|\Psi'|})\cdot\sigma_\Psi$. From $|\sigma_\Psi| = |\Psi|$ we have that $\sigma_\Psi = \sigma'_\Psi,\ T$, and from induction hypothesis for σ'_Ψ we get that the above is equal to $\sigma'_\Psi,\ X_{|\Psi'|}\cdot\sigma_\Psi = \sigma'_\Psi,\ T = \sigma_\Psi$.

Part 2

Lemma B.96 (Interaction of identity substitution and extension substitution) *1.* $t\cdot id_\Psi = t$

2. $\Phi\cdot id_\Psi = \Phi$

3. $\sigma\cdot id_\Psi = \sigma$

4. $T\cdot id_\Psi = T$

5. $K\cdot id_\Psi = K$

6. $\sigma_\Psi\cdot id_\Psi = \sigma_\Psi$

All are trivially proved by induction. We will give only details for the σ_Ψ case.
By induction on σ_Ψ. If $\sigma_\Psi = \bullet$, trivial. If $\sigma_\Psi = \sigma'_\Psi,\ T$, then we have that $(\sigma'_\Psi, T)\cdot\mathsf{id}_\Psi = \sigma'_\Psi\cdot\mathsf{id}_\Psi,\ T\cdot\mathsf{id}_\Psi$. The first part is equal to σ'_Ψ by induction hypothesis (and use of lemma B.92). For the second we split cases for T. We have $([\Phi]t)\cdot\mathsf{id}_\Psi = [\Phi\cdot\mathsf{id}_\Psi](t\cdot\mathsf{id}_\Psi) = [\Phi]t$, and similarly for $([\Phi]\Phi')\cdot\mathsf{id}_\Psi = [\Phi]\Phi'$, by use of the other parts.

Theorem B.97 (Extension of lemma B.39) *1. If* $\Psi;\ \Phi\vdash t : t'$ *and* $\Psi'\vdash\sigma_\Psi : \Psi$ *then* $\Psi';\ \Phi\cdot\sigma_\Psi\vdash t\cdot\sigma_\Psi : t'\cdot\sigma_\Psi$.

2. *If* $\Psi;\ \Phi \vdash \sigma : \Phi'$ *and* $\Psi' \vdash \sigma_\Psi : \Psi$ *then* $\Psi';\ \Phi \cdot \sigma_\Psi \vdash \sigma \cdot \sigma_\Psi : \Phi' \cdot \sigma_\Psi$.

3. *If* $\Psi \vdash \Phi$ *wf and* $\Psi' \vdash \sigma_\Psi : \Psi$ *then* $\Psi' \vdash \Phi \cdot \sigma_\Psi$ *wf.*

4. *If* $\Psi \vdash T : K$ *and* $\Psi' \vdash \sigma_\Psi : \Psi$ *then* $\Psi' \vdash T \cdot \sigma_\Psi : K \cdot \sigma_\Psi$.

5. *If* $\Psi' \vdash \sigma_\Psi : \Psi$ *and* $\Psi'' \vdash \sigma'_\Psi : \Psi'$ *then* $\Psi'' \vdash \sigma_\Psi \cdot \sigma'_\Psi : \Psi$.

Part 1. **Case** $\dfrac{\Phi.\mathbf{I} = t}{\Psi;\ \Phi \vdash f_{\mathbf{I}} : t}\ \triangleright$

We have $(\Phi \cdot \sigma_\Psi).\mathbf{I} \cdot \sigma_\Psi = (\Phi.\mathbf{I}) \cdot \sigma_\Psi$ from lemma B.84.

Case $\dfrac{\Psi.i = T \qquad T = [\Phi']\, t' \qquad \Psi;\ \Phi \vdash \sigma : \Phi'}{\Psi;\ \Phi \vdash X_i/\sigma : t' \cdot \sigma}\ \triangleright$

From lemma B.93 get $\Psi' \vdash \sigma_\Psi.i : (\Psi.i) \cdot \sigma_\Psi$.
Furthermore, this can be written as:
$\Psi' \vdash \sigma_\Psi.i : [\Phi' \cdot \sigma_\Psi]\, t' \cdot \sigma_\Psi$.
Thus by typing inversion, and assuming $\sigma_\Psi.i = [\Phi' \cdot \sigma_\Psi]\, t$ get:
$\Psi';\ \Phi' \cdot \sigma_\Psi \vdash t : t' \cdot \sigma_\Psi$. From part 2 for σ get $\Psi';\ \Phi \cdot \sigma_\Psi \vdash \sigma \cdot \sigma_\Psi : \Phi' \cdot \sigma_\Psi$.
From lemma B.67 and the above we get $\Psi';\ \Phi \cdot \sigma_\Psi \vdash t \cdot (\sigma \cdot \sigma_\Psi) : (t' \cdot \sigma_\Psi) \cdot (\sigma \cdot \sigma_\Psi)$.
Using the lemma B.90 we get that $(t' \cdot \sigma_\Psi) \cdot (\sigma \cdot \sigma_\Psi) = (t' \cdot \sigma) \cdot \sigma_\Psi$, thus the above is the desired.

Case (otherwise) $\triangleright$
The rest of the cases are trivial to adapt to account for indexes from lemma B.39.

Part 2. The cases for $\sigma = \bullet$ or $\sigma = \sigma',\ t$ are entirely similar as before.

Case $\dfrac{\Psi;\ \Phi \vdash \sigma : \Phi' \qquad \Psi.i = [\Phi']\,\mathsf{ctx} \qquad \Phi',\ X_i \subseteq \Phi}{\Psi;\ \Phi \vdash (\sigma,\ \mathbf{id}(X_i)) : (\Phi',\ X_i)}\ \triangleright$

In this case we need to prove that $\Psi';\ \Phi \cdot \sigma_\Psi \vdash (\sigma \cdot \sigma_\Psi,\ \mathsf{id}_{\sigma_\Psi.i}) : (\Phi' \cdot \sigma_\Psi,\ \sigma_\Psi.i)$.
By induction hypothesis for σ we get that $\Psi';\ \Phi \cdot \sigma_\Psi \vdash \sigma \cdot \sigma_\Psi : \Phi' \cdot \sigma_\Psi$.
From lemma B.93 we also get: $\Psi' \vdash \sigma_\Psi.i : \Psi.i \cdot \sigma_\Psi$.
We have that $\Psi.i = [\Phi']\,\mathsf{ctx}$, so this can be rewritten as: $\Psi' \vdash \sigma_\Psi.i : [\Phi' \cdot \sigma_\Psi]\,\mathsf{ctx}$.
By typing inversion get $\sigma_\Psi.i = [\Phi' \cdot \sigma_\Psi]\,\Phi''$ for some Φ'' and:
$\Psi' \vdash [\Phi' \cdot \sigma_\Psi]\,\Phi'' : [\Phi' \cdot \sigma_\Psi]\,\mathsf{ctx}$.
Now proceed by induction on Φ'' to prove that $\Psi';\ \Phi \cdot \sigma_\Psi \vdash (\sigma \cdot \sigma_\Psi,\ \mathsf{id}_{\sigma_\Psi.i}) : (\Phi' \cdot \sigma_\Psi,\ \sigma_\Psi.i)$.
When $\Phi'' = \bullet$, trivial.
When $\Phi'' = \Phi''',\ t$, have $\Psi';\ \Phi \cdot \sigma_\Psi \vdash \sigma \cdot \sigma_\Psi,\ \mathsf{id}_{[\Phi' \cdot \sigma_\Psi]\,\Phi'''} : (\Phi' \cdot \sigma_\Psi,\ \Phi''')$ by induction hypothesis. We can append $f_{|\Phi' \cdot \sigma_\Psi|,|\Phi'''|}$ to this substitution and get the desired, because $(|\Phi' \cdot \sigma_\Psi|, |\Phi'''|) < |\Phi \cdot \sigma_\Psi|$. This is because $(\Phi',\ X_i) \subseteq \Phi$ thus $(\Phi' \cdot \sigma_\Psi,\ \Phi''',\ t) \subseteq \Phi$ and thus $(|\Phi' \cdot \sigma_\Psi|,\ |\Phi'''|, \cdot) \leq |\Phi|$. When $\Phi'' = \Phi''',\ X_j$, have $\Psi';\ \Phi \cdot \sigma_\Psi \vdash \sigma \cdot \sigma_\Psi,\ \mathsf{id}_{[\Phi' \cdot \sigma_\Psi]\,\Phi'''} : (\Phi' \cdot \sigma_\Psi,\ \Phi''')$. Now we have that $\Phi',\ X_i \subseteq \Phi$, which also means that $(\Phi' \cdot \sigma_\Psi,\ \Phi''',\ X_j) \subseteq \Phi \cdot \sigma_\Psi$. Thus we can apply the typing rule for $\mathbf{id}(X_j)$ to get that $\Psi';\ \Phi \cdot \sigma_\Psi \vdash \sigma \cdot \sigma_\Psi,\ \mathsf{id}_{[\Phi' \cdot \sigma_\Psi]\,\Phi'''},\ \mathbf{id}(X_j) : (\Phi' \cdot \sigma_\Psi,\ \Phi''',\ X_j)$, which is the desired.

Part 3. **Case** $\dfrac{}{\Psi \vdash \bullet\ \mathrm{wf}}\ \triangleright$

Trivial.

Case $\dfrac{\Psi \vdash \Phi \text{ wf} \qquad \Psi;\, \Phi \vdash t : s}{\Psi \vdash (\Phi,\, t) \text{ wf}}$ $\triangleright$
By induction hypothesis we get $\Psi' \vdash \Phi \cdot \Psi$ wf.
By use of part 1 we get that $\Psi';\, \Phi \cdot \Psi \vdash t \cdot \Psi : s$.
Thus using the same typing rule we get the desired $\Psi' \vdash (\Phi \cdot \Psi,\, t \cdot \Psi)$ wf.

Case $\dfrac{\Psi \vdash \Phi \text{ wf} \qquad \Psi.i = [\Phi]\,\mathsf{ctx}}{\Psi \vdash (\Phi,\, X_i) \text{ wf}}$ $\triangleright$
By induction hypothesis we get $\Psi' \vdash \Phi \cdot \sigma_\Psi$ wf.
By use of lemma B.93 we get that $\Psi' \vdash \sigma_\Psi.i : \Psi.i \cdot \sigma_\Psi$.
We have $\Psi.i = [\Phi]\,\mathsf{ctx}$ thus the above can be rewritten as $\Psi' \vdash \sigma_\Psi.i : [\Phi \cdot \sigma_\Psi]\,\mathsf{ctx}$.
By inversion of typing get that $\sigma_\Psi.i = [\Phi \cdot \sigma_\Psi]\,\Phi'$ and that $\Psi' \vdash \Phi \cdot \sigma_\Psi,\, \Phi'$ wf. This is exactly the desired result.

Part 4. **Case** $\dfrac{\Psi;\, \Phi \vdash t : t'}{\Psi \vdash [\Phi]\,t : [\Phi]\,t'}$ $\triangleright$
By use of part 1 we get that $\Psi';\, \Phi \cdot \sigma_\Psi \vdash t \cdot \sigma_\Psi : t' \cdot \sigma_\Psi$.
Thus by application of the same typing rule we get exactly the desired.

Case $\dfrac{\Psi \vdash \Phi,\, \Phi' \text{ wf}}{\Psi \vdash [\Phi]\,\Phi' : [\Phi]\,\mathsf{ctx}}$ $\triangleright$
By use of part 3 we get $\Psi' \vdash \Phi \cdot \sigma_\Psi,\, \Phi' \cdot \sigma_\Psi$ wf.
Thus by the same typing rule we get exactly the desired.

Part 5. **Case** $\dfrac{}{\Psi' \vdash \bullet : \bullet}$ $\triangleright$
Trivial.

Case $\dfrac{\Psi' \vdash \sigma_\Psi : \Psi \qquad \Psi' \vdash T : K \cdot \sigma_\Psi}{\Psi' \vdash (\sigma_\Psi,\, T) : (\Psi,\, K)}$ $\triangleright$
By induction we get $\Psi'' \vdash \sigma_\Psi \cdot \sigma'_\Psi : \Psi$.
By use of part 4 we get $\Psi'' \vdash T \cdot \sigma'_\Psi : (K \cdot \sigma_\Psi) \cdot \sigma'_\Psi$.
This is equal to $K \cdot (\sigma_\Psi \cdot \sigma'_\Psi)$ by use of lemma B.94. Thus we get the desired result by applying the same typing rule.

Lemma B.98 *If* $\Psi \vdash \Psi''$ *wf and* $\Psi' \vdash \sigma_\Psi : \Psi$ *then* $\Psi' \vdash \Psi'' \cdot \sigma_\Psi$ *wf.*

By induction on the structure of Ψ''.

Case $\Psi'' = \bullet$ $\triangleright$ Trivial.

Case $\Psi'' = \Psi'',\, [\Phi]\,t$ $\triangleright$
By induction hypothesis we have that $\Psi' \vdash \Psi'' \cdot \sigma_\Psi$ wf.
By inversion of well-formedness for $\Psi'',\, [\Phi]\,t$ we get:
$\Psi,\, \Psi'' \vdash [\Phi]\,t : [\Phi]\,s$.
We have for $\sigma'_\Psi = \sigma_\Psi,\, X_{|\Psi'|},\, \cdots,\, X_{|\Psi'|+|\Psi''|}$, that $\Psi',\, \Psi'' \cdot \sigma_\Psi \vdash \sigma'_\Psi : \Psi,\, \Psi''$.
Thus by application of lemma B.97, we get that:

Ψ', $\Psi'' \cdot \sigma_\Psi \vdash [\Phi \cdot \sigma'_\Psi]\, t \cdot \sigma'_\Psi : [\Phi \cdot \sigma'_\Psi]\, s$.
Thus $\vdash \Psi'$, $(\Psi'', [\Phi]\, t) \cdot \sigma_\Psi$ wf, which is the desired.

Case $\Psi'' = \Psi''$, $[\Phi]\,\mathsf{ctx} \triangleright$ Similarly as the previous case.

B.3 Final extension: bound extension variables

The metatheory presented in the previous subsection only has to do with meta and context variables that are free. We now introduce bound extension variables, (which will be bound in the computational language), entirely similarly to how we have bound and free variables for the logic. We will not re-prove everything here; all theorems from above carry on exactly as they are. We will only prove two theorems that have to do with the interaction of freshen/bind and extension substitutions.

Definition B.99 (Syntax of the language) *The syntax of the logic language is extended below.*

$$
\begin{aligned}
\Phi &::= \cdots \mid \Phi, B_i \\
\sigma &::= \cdots \mid \sigma,\ \mathbf{id}(B_i) \\
t &::= \cdots \mid B_i/\sigma \\
\mathbf{I} &::= \cdots \mid \mathbf{I},\ |B_i|
\end{aligned}
$$

All the following definitions are extended trivially. Application of extension substitution leaves bound extension variables as they are. Bound extension variables are untypable.

Definition B.100 (Freshening of extension variables) *We define freshening similarly to normal variables. We do not define extension variables limits: we will use the condition of well-definedness later. (So if $\lceil t \rceil^M_{N,K}$ is well-defined, that means that it does not have extension variables larger than $N+K$).*

$\boxed{\lceil \mathbf{I} \rceil^M_N}$

$$
\begin{aligned}
\lceil \bullet \rceil &= \bullet \\
\lceil \mathbf{I}, \cdot \rceil &= \lceil \mathbf{I} \rceil, \cdot \\
\lceil \mathbf{I}, X_i \rceil &= \lceil \mathbf{I} \rceil, X_i \\
\lceil \mathbf{I}, B_{M+j} \rceil^M_{N,K} &= \lceil \mathbf{I} \rceil, X_{N+K-j-1} \text{ when } j < K \\
\lceil \mathbf{I}, B_i \rceil^M_{N,K} &= \lceil \mathbf{I} \rceil, B_i \text{ when } i < M
\end{aligned}
$$

$\boxed{\lceil t \rceil^M_{N,K}}$

$$
\begin{aligned}
\lceil f_{\mathbf{I}} \rceil^M_{N,K} &= f_{\lceil \mathbf{I} \rceil^M_{N,K}} \\
\lceil b_i \rceil^M_{N,K} &= b_i \\
\lceil \lambda(t_1).t_2 \rceil^M_{N,K} &= \lambda(\lceil t_1 \rceil^M_{N,K}).\, \lceil t_2 \rceil^M_{N,K} \\
&\cdots \\
\lceil X_i/\sigma \rceil^M_{N,K} &= X_i/(\lceil \sigma \rceil^M_{N,K}) \\
\lceil B_{M+j}/\sigma \rceil^M_{N,K} &= X_{N+K-j-1}/(\lceil \sigma \rceil^M_{N,K}) \text{ when } j < K \\
\lceil B_i/\sigma \rceil^M_{N,K} &= B_i/(\lceil \sigma \rceil^M_{N,K}) \text{ when } i < M
\end{aligned}
$$

$\boxed{\lceil \Phi \rceil^M_{N,K}}$

$$
\begin{array}{lcl}
\lceil \bullet \rceil & = & \bullet \\
\lceil \Phi, t \rceil & = & \lceil \Phi \rceil, \lceil t \rceil \\
\lceil \Phi, X_i \rceil & = & \lceil \Phi \rceil, X_i \\
\lceil \Phi, B_{M+j} \rceil^M_{N,K} & = & \lceil \Phi \rceil, X_{N+K-j-1} \text{ when } j < K \\
\lceil \Phi, B_i \rceil^M_{N,K} & = & \lceil \Phi \rceil, B_i \text{ when } i < M
\end{array}
$$

$\boxed{\lceil \sigma \rceil^M_{N,K}}$

$$
\begin{array}{lcl}
\lceil \bullet \rceil & = & \bullet \\
\lceil \sigma, t \rceil & = & \lceil \sigma \rceil, \lceil t \rceil \\
\lceil \sigma, \mathbf{id}(X_i) \rceil & = & \lceil \sigma \rceil, \mathbf{id}(X_i) \\
\lceil \sigma, \mathbf{id}(B_{M+j}) \rceil^M_{N,K} & = & \lceil \sigma \rceil, \mathbf{id}(X_{N+K-j-1}) \text{ when } j < K \\
\lceil \sigma, \mathbf{id}(B_i) \rceil^M_{N,K} & = & \lceil \sigma \rceil, \mathbf{id}(B_i) \text{ when } i < M
\end{array}
$$

$\boxed{\lceil T \rceil^M_{N,K}}$

$$
\begin{array}{lcl}
\lceil [\Phi] t \rceil & = & [\lceil \Phi \rceil](\lceil t \rceil) \\
\lceil [\Phi] \Phi' \rceil & = & [\lceil \Phi \rceil](\lceil \Phi' \rceil)
\end{array}
$$

$\boxed{\lceil K \rceil^M_{N,K}}$

$$
\begin{array}{lcl}
\lceil [\Phi] t \rceil & = & [\lceil \Phi \rceil](\lceil t \rceil) \\
\lceil [\Phi] \mathsf{ctx} \rceil & = & [\lceil \Phi \rceil] \mathsf{ctx}
\end{array}
$$

$\boxed{\lceil \Psi \rceil^M_{N,K}}$

$$
\begin{array}{lcl}
\lceil \bullet \rceil^M_{N,K} & = & \bullet \\
\lceil \Psi,\, K \rceil^M_{N,K} & = & \lceil \Psi \rceil^M_{N,K}, \lceil K \rceil^{M+|\Psi|}_{N,K}
\end{array}
$$

Definition B.101 (Binding of extension variables) *We define binding similarly to normal variables. Note that this is a bit different (because binding many variables at once is permitted), so the N parameter is the length of the resulting context (the number of free variables after binding has taken place), while $N+K$ is the length of the context where the bind argument is currently in.*

$\boxed{\lfloor \mathbf{I} \rfloor^M_{N,K}}$

$$
\begin{array}{lcl}
\lfloor \bullet \rfloor & = & \bullet \\
\lfloor \mathbf{I}, \cdot \rfloor & = & \lfloor \mathbf{I} \rfloor, \cdot \\
\lfloor \mathbf{I}, X_{N+j} \rfloor^M_{N,K} & = & \lfloor \mathbf{I} \rfloor, B_{M+K-j-1} \text{ when } j < K \\
\lfloor \mathbf{I}, X_i \rfloor^M_{N,K} & = & \lfloor \mathbf{I} \rfloor, X_i \text{ when } i < N \\
\lfloor \mathbf{I}, B_i \rfloor^M_{N,K} & = & \lfloor \mathbf{I} \rfloor, B_i
\end{array}
$$

$\boxed{\lfloor t \rfloor^M_{N,K}}$

$$\begin{array}{lcl}
\lfloor f_{\mathbf{I}} \rfloor^M_{N,K} & = & f_{\lfloor \mathbf{I} \rfloor^M_{N,K}} \\
\lfloor b_i \rfloor^M_{N,K} & = & b_i \\
\lfloor \lambda(t_1).t_2 \rfloor^M_{N,K} & = & \lambda(\lfloor t_1 \rfloor^M_{N,K}).\, \lfloor t_2 \rfloor^M_{N,K} \\
 & & \cdots \\
\lfloor X_{N+j}/\sigma \rfloor^M_{N,K} & = & B_{M+K-j-1}/(\lfloor \sigma \rfloor^M_{N,K}) \text{ when } j < K \\
\lfloor X_i/\sigma \rfloor^M_{N,K} & = & X_i/(\lfloor \sigma \rfloor^M_{N,K}) \text{ when } i < N \\
\lfloor B_i/\sigma \rfloor^M_{N,K} & = & B_i/(\lfloor \sigma \rfloor^M_{N,K})
\end{array}$$

$\boxed{\lfloor \Phi \rfloor^M_{N,K}}$

$$\begin{array}{lcl}
\lfloor \bullet \rfloor & = & \bullet \\
\lfloor \Phi, t \rfloor & = & \lfloor \Phi \rfloor,\, \lfloor t \rfloor \\
\left\lfloor \Phi, X_{N+j} \right\rfloor^M_{N,K} & = & \lfloor \Phi \rfloor,\, B_M \text{ when } j < K \\
\lfloor \Phi, X_i \rfloor^M_{N,K} & = & \lfloor \Phi \rfloor,\, X_i \text{ when } i < N \\
\lfloor \Phi, B_i \rfloor^M_{N,K} & = & \lfloor \Phi \rfloor,\, B_i
\end{array}$$

$\boxed{\lfloor \sigma \rfloor^M_{N,K}}$

$$\begin{array}{lcl}
\lfloor \bullet \rfloor & = & \bullet \\
\lfloor \sigma, t \rfloor & = & \lfloor \sigma \rfloor,\, \lfloor t \rfloor \\
\left\lfloor \sigma, \mathbf{id}(X_{N+j}) \right\rfloor^M_{N,K} & = & \lfloor \sigma \rfloor,\, \mathbf{id}(B_{M+K-j-1}) \text{ when } j < K \\
\lfloor \sigma, \mathbf{id}(X_i) \rfloor^M_{N,K} & = & \lfloor \sigma \rfloor,\, \mathbf{id}(X_i) \text{ when } i < N \\
\lfloor \sigma, \mathbf{id}(B_i) \rfloor^M_{N,K} & = & \lfloor \sigma \rfloor,\, \mathbf{id}(B_i)
\end{array}$$

$\boxed{\lfloor T \rfloor^M_{N,K}}$

$$\begin{array}{lcl}
\lfloor [\Phi] t \rfloor & = & [\lfloor \Phi \rfloor](\lfloor t \rfloor) \\
\lfloor [\Phi] \Phi' \rfloor & = & [\lfloor \Phi \rfloor](\lfloor \Phi' \rfloor)
\end{array}$$

$\boxed{\lfloor K \rfloor^M_{N,K}}$

$$\begin{array}{lcl}
\lfloor [\Phi] t \rfloor & = & [\lfloor \Phi \rfloor](\lfloor t \rfloor) \\
\lfloor [\Phi] \mathsf{ctx} \rfloor & = & [\lfloor \Phi \rfloor] \mathsf{ctx}
\end{array}$$

$\boxed{\lfloor \Psi \rfloor^M_{N,K}}$

$$\begin{array}{lcl}
\lfloor \bullet \rfloor^M_{N,K} & = & \bullet \\
\lfloor \Psi,\, K \rfloor^M_{N,K} & = & \lfloor \Psi \rfloor^M_{N,K},\, \lfloor K \rfloor^{M+|\Psi|}_{N,K}
\end{array}$$

Definition B.102 *Opening up and closing down an extension context works as follows:*

$\boxed{\uparrow \Psi \uparrow_N}$

$$\begin{array}{lcl} \uparrow \bullet \uparrow_N & = & \bullet \\ \uparrow \Psi,\, K \uparrow_N & = & \uparrow \Psi \uparrow_N,\, \uparrow K \uparrow^0_{N,|\Psi|} \end{array}$$

$\boxed{\downarrow \Psi \downarrow_N}$

$$\begin{array}{lcl} \downarrow \bullet \downarrow_N & = & \bullet \\ \downarrow \Psi,\, K \downarrow_N & = & \downarrow \Psi \downarrow_N,\, \downarrow K \downarrow^0_{N,|\Psi|} \end{array}$$

Now we prove a couple of theorems.

Lemma B.103 (Freshening of extension variables and extension substitution) *Assuming* $|\sigma_\Psi| = N$, $\lceil \cdot \rceil^M_{N,K}$ *and* $\lceil \cdot \rceil^M_{N',K}$ *are well-defined, we have:*

1. $\lceil \mathbf{I} \cdot \sigma_\Psi \rceil^M_{N',K} = \lceil \mathbf{I} \rceil^M_{N,K} \cdot (\sigma_\Psi,\, X_{N'},\, \cdots,\, X_{N'+K-1})$
2. $\lceil t \cdot \sigma_\Psi \rceil^M_{N',K} = \lceil t \rceil^M_{N,K} \cdot (\sigma_\Psi,\, X_{N'},\, \cdots,\, X_{N'+K-1})$
3. $\lceil \Phi \cdot \sigma_\Psi \rceil^M_{N',K} = \lceil \Phi \rceil^M_{N,K} \cdot (\sigma_\Psi,\, X_{N'},\, \cdots,\, X_{N'+K-1})$
4. $\lceil \sigma \cdot \sigma_\Psi \rceil^M_{N',K} = \lceil \sigma \rceil^M_{N,K} \cdot (\sigma_\Psi,\, X_{N'},\, \cdots,\, X_{N'+K-1})$
5. $\lceil T \cdot \sigma_\Psi \rceil^M_{N',K} = \lceil T \rceil^M_{N,K} \cdot (\sigma_\Psi,\, X_{N'},\, \cdots,\, X_{N'+K-1})$
6. $\lceil K \cdot \sigma_\Psi \rceil^M_{N',K} = \lceil K \rceil^M_{N,K} \cdot (\sigma_\Psi,\, X_{N'},\, \cdots,\, X_{N'+K-1})$
7. $\lceil \Psi \cdot \sigma_\Psi \rceil^M_{N',K} = \lceil \Psi \rceil^M_{N,K} \cdot (\sigma_\Psi,\, X_{N'},\, \cdots,\, X_{N'+K-1})$

Part 2 By induction on t and use of the rest of the parts. The interesting case is $t = B_{M+j}/sigma$ with $j < K$. We have that the left-hand-side is equal to $X_{N'+K-j-1}/(\lceil \sigma \cdot \sigma_\Psi \rceil^M_{N',K})$, which by part 4 is equal to $X_{N'+K-j-1}/(\lceil \sigma \rceil^M_{N,K} \cdot (\sigma_\Psi,\, X_{N'},\, \cdots,\, X_{N'+K-1})$. The right-hand-side is equal to $(X_{N+K-j-1}/\lceil \sigma \rceil^M_{N,K}) \cdot (\sigma_\Psi,\, X_{N'},\, \cdots,\, X_{N'+K-1}) = X_{N'+K-j-1}/(\lceil \sigma \rceil^M_{N,K} \cdot (\sigma_\Psi,\, X_{N'},\, \cdots,\, X_{N'+K-1})$, which is exactly equal to the left-hand-side.

Part 7 By induction on Ψ. The interesting case occurs when $\Psi = \Psi',\, K$.
In that case, we have that the left-hand-side is equal to:
$\left\lceil \Psi' \cdot \sigma_\Psi,\, K \cdot (\sigma_\Psi,\, X_{N'},\, \cdots,\, X_{N'+|\Psi'|}) \right\rceil^M_{N',K}$.
Since K does not contain variables bigger than $X_{|\sigma_\Psi|}$ (since $\lceil K \rceil^M_{N,K}$ is well-defined), we have that this is further equal to:
$\lceil \Psi' \cdot \sigma_\Psi,\, K \cdot \sigma_\Psi \rceil^M_{N',K}$.
This is then equal to:
$\lceil \Psi' \cdot \sigma_\Psi \rceil^M_{N',K},\, \lceil K \cdot \sigma_\Psi \rceil^{M+|\Psi' \cdot \sigma_\Psi|}_{N',K}$. Setting $\sigma'_\Psi = \sigma_\Psi,\, X_{N'},\, \cdots,\, X_{N'+K-1}$ we have by induction hypothesis and part 6 that this is equal to:
$\lceil \Psi' \rceil^M_{N,K} \cdot \sigma'_\Psi,\, \lceil K \rceil^{M+|\Psi' \cdot \sigma_\Psi|}_{N,K} \cdot \sigma'_\Psi$.
The right-hand-side is equal to: $\lceil \Psi' \rceil^M_{N,K} \cdot \sigma'_\Psi,\, \lceil K \rceil^{M+|\Psi'|}_{N,K} \cdot (\sigma'_\Psi,\, X_{N'+K-1},\, \cdots,\, X_{N'+K-1+|\Psi'|}$
Since $\lceil K \rceil^{M+|\Psi'|}_{N,K}$ is well-defined, we have that it does not contain variables larger than X_{N+K-1}, and thus we have:
$\lceil K \rceil^{M+|\Psi'|}_{N,K} \cdot (\sigma'_\Psi,\, X_{N'+K-1},\, \cdots,\, X_{N'+K-1+|\Psi'|} = \lceil K \rceil^{M+|\Psi'|}_{N,K} \cdot \sigma'_\Psi$.
Thus the two sides are equal.

Rest By direct application of the other parts.

Lemma B.104 (Binding of extension variables and extension substitution) *Assuming* $|\sigma_\Psi| = N$, $\lfloor\cdot\rfloor^M_{N,K}$ *and* $\lfloor\cdot\rfloor^M_{N',K}$ *are well-defined, we have:*

1. $\lfloor \mathbf{I} \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})\rfloor^M_{N',K} = \lfloor \mathbf{I}\rfloor^M_{N,K} \cdot \sigma_\Psi$
2. $\lfloor t \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})\rfloor^M_{N',K} = \lfloor t\rfloor^M_{N,K} \cdot \sigma_\Psi$
3. $\lfloor \Phi \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})\rfloor^M_{N',K} = \lfloor \Phi\rfloor^M_{N,K} \cdot \sigma_\Psi$
4. $\lfloor \sigma \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})\rfloor^M_{N',K} = \lfloor \sigma\rfloor^M_{N,K} \cdot \sigma_\Psi$
5. $\lfloor T \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})\rfloor^M_{N',K} = \lfloor T\rfloor^M_{N,K} \cdot \sigma_\Psi$
6. $\lfloor K \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})\rfloor^M_{N',K} = \lfloor K\rfloor^M_{N,K} \cdot \sigma_\Psi$
7. $\lfloor \Psi \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})\rfloor^M_{N',K} = \lfloor \Psi\rfloor^M_{N,K} \cdot \sigma_\Psi$

Part 2 The interesting case is when $t = X_{N+j}/\sigma$ with $j < K$. In that case, the left-hand-side becomes: $\left\lfloor X_{N'+j}/(\sigma \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1}))\right\rfloor^M_{N',K} = B_{M+K-j-1}/(\lfloor \sigma \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})\rfloor^M_{N',K}) = B_M/(\lfloor\sigma\rfloor^M_{N,K} \cdot \sigma_\Psi)$ by part 4.
The right-hand-side becomes $(B_{M+K-j-1}/(\lfloor\sigma\rfloor^M_{N,K})) \cdot \sigma_\Psi = B_M/(\lfloor\sigma\rfloor^M_{N,K} \cdot \sigma_\Psi)$.

Rest Again, simple by induction and use of other parts; similarly as above.

Lemma B.105 1. $\left\lceil \lfloor \mathbf{I}\rfloor^M_{N,K}\right\rceil^M_{N,K} = \mathbf{I}$
2. $\left\lceil \lfloor t\rfloor^M_{N,K}\right\rceil^M_{N,K} = t$
3. $\left\lceil \lfloor \Phi\rfloor^M_{N,K}\right\rceil^M_{N,K} = \Phi$
4. $\left\lceil \lfloor \sigma\rfloor^M_{N,K}\right\rceil^M_{N,K} = \sigma$
5. $\left\lceil \lfloor T\rfloor^M_{N,K}\right\rceil^M_{N,K} = T$
6. $\left\lceil \lfloor K\rfloor^M_{N,K}\right\rceil^M_{N,K} = K$
7. $\left\lceil \lfloor \Psi\rfloor^M_{N,K}\right\rceil^M_{N,K} = \Psi$

Trivial by structural induction.

Lemma B.106 *If* $|\sigma_\Psi| = |\Psi|$ *and* $\uparrow\cdot\restriction_{|\Psi|}$ *and* $\uparrow\cdot\restriction_{|\Psi'|}$ *are well-defined, then* $\uparrow\Psi''\restriction_{|\Psi|} \cdot \sigma_\Psi = \uparrow\Psi'' \cdot \sigma_\Psi\restriction_{|\Psi'|}$

By induction on Ψ''.
When $\Psi'' = \bullet$, trivial.
When $\Psi'' = \Psi''', K$ we have that:
$\uparrow\Psi''', K\restriction_{|\Psi|} = \uparrow\Psi'''\restriction_{|\Psi|}, \lceil K\rceil_{|\Psi|,|\Psi'''|}$.
Applying σ_Ψ to this we get:
$\uparrow\Psi'''\restriction_{|\Psi|} \cdot \sigma_\Psi, \lceil K\rceil_{|\Psi|,|\Psi'''|} \cdot (\sigma_\Psi, X_{|\Psi'|}, \cdots, X_{|\Psi'|+|\Psi''|})$.
By induction hypothesis the first part is equal to:

$\lceil \Psi'' \cdot \sigma_\Psi \rceil_{|\Psi'|}$.
Using lemma B.87 for the second part we get that it's equal to:
$\lceil K \cdot \sigma_\Psi \rceil_{|\Psi'|,|\Psi''|}$
Furthermore, since K does not contain variables greater than $X_{|\Psi|}$, we have that $K \cdot \sigma_\Psi = K \cdot (\sigma_\Psi, X_{|\Psi'|}, \cdots, X_{|\Psi'|+|\Psi''|}$.
Thus, the left hand side is equal to $\left\lceil \Psi'' \cdot \sigma_\Psi,\ K \cdot (\sigma_\Psi, X_{|\Psi'|}, \cdots, X_{|\Psi'|+|\Psi''|} \right\rceil_{|\Psi'|,|\Psi''|}$, which is equal to the right-hand-side.

C. Definition and metatheory of computational language

Definition C.1 *The syntax of the computational language is defined below.*

$$
\begin{array}{rl}
k ::= & \star \mid k \to k \mid \Pi(K).k \\
\tau ::= & \Pi(K).\tau \mid \Sigma(K).\tau \mid \lambda(K).\tau \mid \tau\ T \\
& \mid \mathsf{unit} \mid \bot \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \mu\alpha : k.\tau \mid \mathsf{ref}\ \tau \mid \forall\alpha : k.\tau \mid \lambda\alpha : k.\tau \mid \tau_1\ \tau_2 \mid \alpha \\
e ::= & \Lambda(K).e \mid e\ T \mid \mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ e \mid \mathsf{unpack}\ e\ (.)x.(e') \\
& \mid () \mid \mathsf{error} \mid \lambda x : \tau.e \mid e\ e' \mid x \mid (e,\ e') \mid \mathsf{proj}_i\ e \mid \mathsf{inj}_i\ e \mid \mathsf{case}(e, x.e', x.e'') \mid \mathsf{fold}\ e \mid \mathsf{unfold}\ e \mid \mathsf{ref}\ e \\
& \mid e := e' \mid !e \mid l \mid \Lambda\alpha : k.e \mid e\ \tau \mid \mathsf{fix}\ x : \tau.e \\
& \mid \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi.T' \mapsto e') \\
\Gamma ::= & \bullet \mid \Gamma,\ x : \tau \mid \Gamma,\ \alpha : k \\
\Sigma ::= & \bullet \mid \Sigma,\ l : \tau
\end{array}
$$

Definition C.2 *Freshening and binding for computational kinds, types and terms are defined as follows.*

$\boxed{\lceil k \rceil^M_{N,K}}$

$$
\begin{array}{lcl}
\lceil \star \rceil^M_{N,K} & = & \star \\
\lceil \Pi(K).k \rceil^M_{N,K} & = & \Pi(\lceil K \rceil^M_{N,K}).\ \lceil k \rceil^{M+1}_{N,K}
\end{array}
$$

$\boxed{\lceil \tau \rceil^M_{N,K}}$

$$
\begin{array}{lcl}
\lceil \Pi(K).\tau \rceil^M_{N,K} & = & \Pi(\lceil K \rceil^M_{N,K}).\ \lceil \tau \rceil^{M+1}_{N,K} \\
\lceil \Sigma(K).\tau \rceil^M_{N,K} & = & \Sigma(\lceil K \rceil^M_{N,K}).\ \lceil \tau \rceil^{M+1}_{N,K} \\
\lceil \lambda(K).\tau \rceil^M_{N,K} & = & \lambda(\lceil K \rceil^M_{N,K}).\ \lceil \tau \rceil^{M+1}_{N,K} \\
\lceil \tau\ T \rceil^M_{N,K} & = & \lceil \tau \rceil^M_{N,K}\ \lceil T \rceil^M_{N,K} \\
\lceil \mathsf{unit} \rceil^M_{N,K} & = & \mathsf{unit} \\
\lceil \bot \rceil^M_{N,K} & = & \bot \\
\lceil \tau_1 \to \tau_2 \rceil^M_{N,K} & = & \lceil \tau_1 \rceil^M_{N,K} \to \lceil \tau_2 \rceil^M_{N,K} \\
\lceil \tau_1 \times \tau_2 \rceil^M_{N,K} & = & \lceil \tau_1 \rceil^M_{N,K} \times \lceil \tau_2 \rceil^M_{N,K} \\
\lceil \tau_1 + \tau_2 \rceil^M_{N,K} & = & \lceil \tau_1 \rceil^M_{N,K} + \lceil \tau_2 \rceil^M_{N,K} \\
\lceil \mu\alpha : k.\tau \rceil^M_{N,K} & = & \mu\alpha : \lceil k \rceil^M_{N,K}.\ \lceil \tau \rceil^M_{N,K} \\
\lceil \mathsf{ref}\ \tau \rceil^M_{N,K} & = & \mathsf{ref}\ \lceil \tau \rceil^M_{N,K} \\
\lceil \forall\alpha : k.\tau \rceil^M_{N,K} & = & \forall\alpha : \lceil k \rceil^M_{N,K}.\ \lceil \tau \rceil^M_{N,K} \\
\lceil \lambda\alpha : k.\tau \rceil^M_{N,K} & = & \lambda\alpha : \lceil k \rceil^M_{N,K}.\ \lceil \tau \rceil^M_{N,K} \\
\lceil \tau_1\ \tau_2 \rceil^M_{N,K} & = & \lceil \tau_1 \rceil^M_{N,K}\ \lceil \tau_2 \rceil^M_{N,K} \\
\lceil \alpha \rceil^M_{N,K} & = & \alpha
\end{array}
$$

$\boxed{\lceil e \rceil^M_{N,K}}$

$$
\begin{array}{lcl}
\lceil \Lambda(K).e \rceil^M_{N,K} & = & \Lambda(\lceil K \rceil^M_{N,K}).\, \lceil e \rceil^{M+1}_{N,K} \\
\lceil e\ T \rceil^M_{N,K} & = & \lceil e \rceil^M_{N,K}\ \lceil T \rceil^M_{N,K} \\
\lceil \mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ e \rceil^M_{N,K} & = & \mathsf{pack}\ \lceil T \rceil^M_{N,K}\ \mathsf{return}\ (.\lceil \tau \rceil^{M+1}_{N,K})\ \mathsf{with}\ \lceil e \rceil^M_{N,K} \\
\lceil \mathsf{unpack}\ e\ (.)x.(e') \rceil^M_{N,K} & = & \mathsf{unpack}\ \lceil e \rceil^M_{N,K}\ (.)x.(\lceil e' \rceil^{M+1}_{N,K}) \\
\lceil () \rceil^M_{N,K} & = & () \\
\lceil \mathsf{error} \rceil^M_{N,K} & = & \mathsf{error} \\
\lceil \lambda x : \tau.e \rceil^M_{N,K} & = & \lambda x : \lceil \tau \rceil^M_{N,K}.\, \lceil e \rceil^M_{N,K} \\
\lceil e_1\ e_2 \rceil^M_{N,K} & = & \lceil e_1 \rceil^M_{N,K}\ \lceil e_2 \rceil^M_{N,K} \\
\lceil x \rceil^M_{N,K} & = & x \\
\lceil (e,\ e') \rceil^M_{N,K} & = & (\lceil e \rceil^M_{N,K},\ \lceil e' \rceil^M_{N,K}) \\
\lceil \mathsf{proj}_i\ e \rceil^M_{N,K} & = & \mathsf{proj}_i\ \lceil e \rceil^M_{N,K} \\
\lceil \mathsf{inj}_i\ e \rceil^M_{N,K} & = & \mathsf{inj}_i\ \lceil e \rceil^M_{N,K} \\
\lceil \mathsf{case}(e, x.e', x.e'') \rceil^M_{N,K} & = & \mathsf{case}(\lceil e \rceil^M_{N,K}, x.\lceil e' \rceil^M_{N,K}, x.\lceil e'' \rceil^M_{N,K}) \\
\lceil \mathsf{fold}\ e \rceil^M_{N,K} & = & \mathsf{fold}\ \lceil e \rceil^M_{N,K} \\
\lceil \mathsf{unfold}\ e \rceil^M_{N,K} & = & \mathsf{unfold}\ \lceil e \rceil^M_{N,K} \\
\lceil \mathsf{ref}\ e \rceil^M_{N,K} & = & \mathsf{ref}\ \lceil e \rceil^M_{N,K} \\
\lceil e_1 := e_2 \rceil^M_{N,K} & = & \lceil e_1 \rceil^M_{N,K} := \lceil e_2 \rceil^M_{N,K} \\
\lceil !e \rceil^M_{N,K} & = & !\lceil e \rceil^M_{N,K} \\
\lceil l \rceil^M_{N,K} & = & l \\
\lceil \Lambda\alpha : k.e \rceil^M_{N,K} & = & \Lambda\alpha : \lceil k \rceil^M_{N,K}.\, \lceil e \rceil^M_{N,K} \\
\lceil e\ \tau \rceil^M_{N,K} & = & \lceil e \rceil^M_{N,K}\ \lceil \tau \rceil^M_{N,K} \\
\lceil \mathsf{fix}\ x : \tau.e \rceil^M_{N,K} & = & \mathsf{fix}\ x : \lceil \tau \rceil^M_{N,K}.\, \lceil e \rceil^M_{N,K} \\
\lceil \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi.T' \mapsto e') \rceil & = & \mathsf{unify}\ \lceil T \rceil^M_{N,K}\ \mathsf{return}\ (.\lceil \tau \rceil^{M+1}_{N,K})\ \mathsf{with}\ (\lceil \Psi \rceil^M_{N,K}.\lceil T' \rceil^{M+|\Psi|}_{N,K} \mapsto \lceil e' \rceil^{M+|\Psi|}_{N,K})
\end{array}
$$

$\boxed{\lfloor k \rfloor^M_{N,K}}$

$$
\begin{array}{lcl}
\lfloor \star \rfloor^M_{N,K} & = & \star \\
\lfloor \Pi(K).k \rfloor^M_{N,K} & = & \Pi(\lfloor K \rfloor^M_{N,K}).\, \lfloor k \rfloor^{M+1}_{N,K}
\end{array}
$$

$\boxed{\lfloor \tau \rfloor^M_{N,K}}$

$$
\begin{array}{lcl}
\lfloor \Pi(K).\tau \rfloor^M_{N,K} & = & \Pi(\lfloor K \rfloor^M_{N,K}).\, \lfloor \tau \rfloor^{M+1}_{N,K} \\
\lfloor \Sigma(K).\tau \rfloor^M_{N,K} & = & \Sigma(\lfloor K \rfloor^M_{N,K}).\, \lfloor \tau \rfloor^{M+1}_{N,K} \\
\lfloor \lambda(K).\tau \rfloor^M_{N,K} & = & \lambda(\lfloor K \rfloor^M_{N,K}).\, \lfloor \tau \rfloor^{M+1}_{N,K} \\
\lfloor \tau\ T \rfloor^M_{N,K} & = & \lfloor \tau \rfloor^M_{N,K}\ \lfloor T \rfloor^M_{N,K} \\
\lfloor \mathsf{unit} \rfloor^M_{N,K} & = & \mathsf{unit}
\end{array}
$$

$\boxed{\lfloor\tau\rfloor^M_{N,K} \text{ (continued)}}$

$$
\begin{array}{lcl}
\lfloor\bot\rfloor^M_{N,K} & = & \bot \\
\lfloor\tau_1 \to \tau_2\rfloor^M_{N,K} & = & \lfloor\tau_1\rfloor^M_{N,K} \to \lfloor\tau_2\rfloor^M_{N,K} \\
\lfloor\tau_1 \times \tau_2\rfloor^M_{N,K} & = & \lfloor\tau_1\rfloor^M_{N,K} \times \lfloor\tau_2\rfloor^M_{N,K} \\
\lfloor\tau_1 + \tau_2\rfloor^M_{N,K} & = & \lfloor\tau_1\rfloor^M_{N,K} + \lfloor\tau_2\rfloor^M_{N,K} \\
\lfloor\mu\alpha : k.\tau\rfloor^M_{N,K} & = & \mu\alpha : \lfloor k\rfloor^M_{N,K} . \lfloor\tau\rfloor^M_{N,K} \\
\lfloor\mathsf{ref}\ \tau\rfloor^M_{N,K} & = & \mathsf{ref}\ \lfloor\tau\rfloor^M_{N,K} \\
\lfloor\forall\alpha : k.\tau\rfloor^M_{N,K} & = & \forall\alpha : \lfloor k\rfloor^M_{N,K} . \lfloor\tau\rfloor^M_{N,K} \\
\lfloor\lambda\alpha : k.\tau\rfloor^M_{N,K} & = & \lambda\alpha : \lfloor k\rfloor^M_{N,K} . \lfloor\tau\rfloor^M_{N,K} \\
\lfloor\tau_1\ \tau_2\rfloor^M_{N,K} & = & \lfloor\tau_1\rfloor^M_{N,K}\ \lfloor\tau_2\rfloor^M_{N,K} \\
\lfloor\alpha\rfloor^M_{N,K} & = & \alpha
\end{array}
$$

$\boxed{\lfloor e\rfloor^M_{N,K}}$

$$
\begin{array}{lcl}
\lfloor\Lambda(K).e\rfloor^M_{N,K} & = & \Lambda(\lfloor K\rfloor^M_{N,K}) . \lfloor e\rfloor^{M+1}_{N,K} \\
\lfloor e\ T\rfloor^M_{N,K} & = & \lfloor e\rfloor^M_{N,K}\ \lfloor T\rfloor^M_{N,K} \\
\lfloor\mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ e\rfloor^M_{N,K} & = & \mathsf{pack}\ \lfloor T\rfloor^M_{N,K}\ \mathsf{return}\ (.\lfloor\tau\rfloor^{M+1}_{N,K})\ \mathsf{with}\ \lfloor e\rfloor^M_{N,K} \\
\lfloor\mathsf{unpack}\ e\ (.)x.(e')\rfloor^M_{N,K} & = & \mathsf{unpack}\ \lfloor e\rfloor^M_{N,K}\ (.)x.(\lfloor e'\rfloor^{M+1}_{N,K}) \\
\lfloor()\rfloor^M_{N,K} & = & () \\
\lfloor\mathsf{error}\rfloor^M_{N,K} & = & \mathsf{error} \\
\lfloor\lambda x : \tau.e\rfloor^M_{N,K} & = & \lambda x : \lfloor\tau\rfloor^M_{N,K} . \lfloor e\rfloor^M_{N,K} \\
\lfloor e_1\ e_2\rfloor^M_{N,K} & = & \lfloor e_1\rfloor^M_{N,K}\ \lfloor e_2\rfloor^M_{N,K} \\
\lfloor x\rfloor^M_{N,K} & = & x \\
\lfloor(e,\ e')\rfloor^M_{N,K} & = & (\lfloor e\rfloor^M_{N,K},\ \lfloor e'\rfloor^M_{N,K}) \\
\lfloor\mathsf{proj}_i\ e\rfloor^M_{N,K} & = & \mathsf{proj}_i\ \lfloor e\rfloor^M_{N,K} \\
\lfloor\mathsf{inj}_i\ e\rfloor^M_{N,K} & = & \mathsf{inj}_i\ \lfloor e\rfloor^M_{N,K} \\
\lfloor\mathsf{case}(e, x.e', x.e'')\rfloor^M_{N,K} & = & \mathsf{case}(\lfloor e\rfloor^M_{N,K}, x.\lfloor e'\rfloor^M_{N,K}, x.\lfloor e''\rfloor^M_{N,K}) \\
\lfloor\mathsf{fold}\ e\rfloor^M_{N,K} & = & \mathsf{fold}\ \lfloor e\rfloor^M_{N,K} \\
\lfloor\mathsf{unfold}\ e\rfloor^M_{N,K} & = & \mathsf{unfold}\ \lfloor e\rfloor^M_{N,K} \\
\lfloor\mathsf{ref}\ e\rfloor^M_{N,K} & = & \mathsf{ref}\ \lfloor e\rfloor^M_{N,K} \\
\lfloor e_1 := e_2\rfloor^M_{N,K} & = & \lfloor e_1\rfloor^M_{N,K} := \lfloor e_2\rfloor^M_{N,K} \\
\lfloor !e\rfloor^M_{N,K} & = & !\lfloor e\rfloor^M_{N,K} \\
\lfloor l\rfloor^M_{N,K} & = & l \\
\lfloor\Lambda\alpha : k.e\rfloor^M_{N,K} & = & \Lambda\alpha : \lfloor k\rfloor^M_{N,K} . \lfloor e\rfloor^M_{N,K} \\
\lfloor e\ \tau\rfloor^M_{N,K} & = & \lfloor e\rfloor^M_{N,K}\ \lfloor\tau\rfloor^M_{N,K} \\
\lfloor\mathsf{fix}\ x : \tau.e\rfloor^M_{N,K} & = & \mathsf{fix}\ x : \lfloor\tau\rfloor^M_{N,K} . \lfloor e\rfloor^M_{N,K} \\
\lfloor\mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi.T' \mapsto e')\rfloor & = & \mathsf{unify}\ \lfloor T\rfloor^M_{N,K}\ \mathsf{return}\ (.\lfloor\tau\rfloor^{M+1}_{N,K})\ \mathsf{with}\ (\lfloor\Psi\rfloor^M_{N,K} . \lfloor T'\rfloor^{M+|\Psi|}_{N,K} \mapsto \lfloor e'\rfloor^{M+|\Psi|}_{N,K})
\end{array}
$$

Definition C.3 *Extension substitution application to computational-level kinds, types and terms.*

$\boxed{k \cdot \sigma_\Psi}$

$$
\begin{array}{lcl}
\star \cdot \sigma_\Psi & = & \star \\
(k \to k) \cdot \sigma_\Psi & = & k \cdot \sigma_\Psi \to k \cdot \sigma_\Psi \\
(\Pi(K).k) \cdot \sigma_\Psi & = & \Pi(K \cdot \sigma_\Psi).k \cdot \sigma_\Psi
\end{array}
$$

$\boxed{\tau \cdot \sigma_\Psi}$

$$
\begin{array}{lcl}
(\Pi(K).\tau) \cdot \sigma_\Psi & = & \Pi(K \cdot \sigma_\Psi).\tau \cdot \sigma_\Psi \\
(\Sigma(K).\tau) \cdot \sigma_\Psi & = & \Sigma(K \cdot \sigma_\Psi).\tau \cdot \sigma_\Psi \\
(\lambda(K).\tau) \cdot \sigma_\Psi & = & \lambda(K \cdot \sigma_\Psi).\tau \cdot \sigma_\Psi \\
(\tau\ T) \cdot \sigma_\Psi & = & \tau \cdot \sigma_\Psi\ T \cdot \sigma_\Psi \\
\mathsf{unit} \cdot \sigma_\Psi & = & \mathsf{unit} \\
\bot \cdot \sigma_\Psi & = & \bot \\
(\tau_1 \to \tau_2) \cdot \sigma_\Psi & = & \tau_1 \cdot \sigma_\Psi \to \tau_2 \cdot \sigma_\Psi \\
(\tau_1 \times \tau_2) \cdot \sigma_\Psi & = & \tau_1 \cdot \sigma_\Psi \times \tau_2 \cdot \sigma_\Psi \\
(\tau_1 + \tau_2) \cdot \sigma_\Psi & = & \tau_1 \cdot \sigma_\Psi + \tau_2 \cdot \sigma_\Psi \\
(\mu\alpha : k.\tau) \cdot \sigma_\Psi & = & \mu\alpha : k \cdot \sigma_\Psi.\tau \cdot \sigma_\Psi \\
(\mathsf{ref}\ \tau) \cdot \sigma_\Psi & = & \mathsf{ref}\ \tau \cdot \sigma_\Psi \\
(\forall\alpha : k.\tau) \cdot \sigma_\Psi & = & \forall\alpha : k \cdot \sigma_\Psi.\tau \cdot \sigma_\Psi \\
(\lambda\alpha : k.\tau) \cdot \sigma_\Psi & = & \lambda\alpha : k \cdot \sigma_\Psi.\tau \cdot \sigma_\Psi \\
(\tau_1\ \tau_2) \cdot \sigma_\Psi & = & \tau_1 \cdot \sigma_\Psi\ \tau_2 \cdot \sigma_\Psi \\
\alpha \cdot \sigma_\Psi & = & \alpha
\end{array}
$$

$\boxed{e \cdot \sigma_\Psi}$

$$
\begin{array}{lcl}
(\Lambda(K).e) \cdot \sigma_\Psi & = & \Lambda(K \cdot \sigma_\Psi).e \cdot \sigma_\Psi \\
(e\ T) \cdot \sigma_\Psi & = & e \cdot \sigma_\Psi\ T \cdot \sigma_\Psi \\
(\mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ e) \cdot \sigma_\Psi & = & \mathsf{pack}\ T \cdot \sigma_\Psi\ \mathsf{return}\ (.\tau \cdot \sigma_\Psi)\ \mathsf{with}\ e \cdot \sigma_\Psi \\
(\mathsf{unpack}\ e\ (.)x.(e')) \cdot \sigma_\Psi & = & \mathsf{unpack}\ e \cdot \sigma_\Psi\ (.)x.(e' \cdot \sigma_\Psi) \\
() \cdot \sigma_\Psi & = & () \\
\mathsf{error} \cdot \sigma_\Psi & = & \mathsf{error} \\
(\lambda x : \tau.e) \cdot \sigma_\Psi & = & \lambda x : \tau \cdot \sigma_\Psi.e \cdot \sigma_\Psi \\
(e\ e') \cdot \sigma_\Psi & = & e \cdot \sigma_\Psi\ e' \cdot \sigma_\Psi \\
x \cdot \sigma_\Psi & = & x \\
(e,\ e') \cdot \sigma_\Psi & = & (e \cdot \sigma_\Psi,\ e' \cdot \sigma_\Psi) \\
(\mathsf{proj}_i\ e) \cdot \sigma_\Psi & = & \mathsf{proj}_i\ e \cdot \sigma_\Psi \\
(\mathsf{inj}_i\ e) \cdot \sigma_\Psi & = & \mathsf{inj}_i\ e \cdot \sigma_\Psi \\
(\mathsf{case}(e, x.e', x.e'')) \cdot \sigma_\Psi & = & \mathsf{case}(e \cdot \sigma_\Psi, x.e' \cdot \sigma_\Psi, x.e'' \cdot \sigma_\Psi) \\
(\mathsf{fold}\ e) \cdot \sigma_\Psi & = & \mathsf{fold}\ e \cdot \sigma_\Psi \\
(\mathsf{unfold}\ e) \cdot \sigma_\Psi & = & \mathsf{unfold}\ e \cdot \sigma_\Psi \\
(\mathsf{ref}\ e) \cdot \sigma_\Psi & = & \mathsf{ref}\ e \cdot \sigma_\Psi \\
(e := e') \cdot \sigma_\Psi & = & e \cdot \sigma_\Psi := e' \cdot \sigma_\Psi \\
(!e) \cdot \sigma_\Psi & = & !e \cdot \sigma_\Psi \\
l \cdot \sigma_\Psi & = & l \\
(\Lambda\alpha : k.e) \cdot \sigma_\Psi & = & \Lambda\alpha : k \cdot \sigma_\Psi.e \cdot \sigma_\Psi \\
(e\ \tau) \cdot \sigma_\Psi & = & e \cdot \sigma_\Psi\ \tau \cdot \sigma_\Psi \\
(\mathsf{fix}\ x : \tau.e) \cdot \sigma_\Psi & = & \mathsf{fix}\ x : \tau \cdot \sigma_\Psi.e \cdot \sigma_\Psi \\
(\mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi.T' \mapsto e')) \cdot \sigma_\Psi & = & \mathsf{unify}\ T \cdot \sigma_\Psi\ \mathsf{return}\ (.\tau \cdot \sigma_\Psi)\ \mathsf{with}\ (\Psi \cdot \sigma_\Psi.T' \cdot \sigma_\Psi \mapsto e' \cdot \sigma_\Psi)
\end{array}
$$

$\boxed{\Gamma \cdot \sigma_\Psi}$

$$
\begin{array}{lcl}
\bullet \cdot \sigma_\Psi & = & \bullet \\
(\Gamma,\ x : \tau) \cdot \sigma_\Psi & = & \Gamma \cdot \sigma_\Psi,\ x : \tau \cdot \sigma_\Psi \\
(\Gamma,\ \alpha : k) \cdot \sigma_\Psi & = & \Gamma \cdot \sigma_\Psi,\ \alpha : k \cdot \sigma_\Psi
\end{array}
$$

Definition C.4 *The typing judgements for the computational language are given below.*

$\boxed{\Psi \vdash k \text{ wf}}$

$$\frac{\vdash \Psi \text{ wf}}{\Psi \vdash \star \text{ wf}} \qquad \frac{\Psi \vdash k \text{ wf} \qquad \Psi \vdash k' \text{ wf}}{\Psi \vdash k \to k' \text{ wf}} \qquad \frac{\vdash \Psi,\, K \text{ wf} \qquad \Psi,\, K \vdash \lceil k \rceil_{|\Psi|,1} \text{ wf}}{\Psi \vdash \Pi(K).k \text{ wf}}$$

$\boxed{\Psi;\, \Gamma \vdash \tau : k}$

$$\frac{\Psi,\, K;\, \Gamma \vdash \lceil \tau \rceil_{|\Psi|,1} : \star}{\Psi;\, \Gamma \vdash \Pi(K).\tau : \star} \qquad \frac{\Psi,\, K;\, \Gamma \vdash \lceil \tau \rceil_{|\Psi|,1} : \star}{\Psi;\, \Gamma \vdash \Sigma(K).\tau : \star} \qquad \frac{\Psi,\, K;\, \Gamma \vdash \lceil \tau \rceil_{|\Psi|,1} : k}{\Psi;\, \Gamma \vdash \lambda(K).\tau : \Pi(K).\, \lfloor k \rfloor_{|\Psi|,1}}$$

$$\frac{\Psi;\, \Gamma \vdash \tau : \Pi(K).k \qquad \Psi \vdash T : K}{\Psi;\, \Gamma \vdash \tau\, T : \lceil k \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, T)} \qquad \frac{}{\Psi;\, \Gamma \vdash \mathsf{unit} : \star} \qquad \frac{}{\Psi;\, \Gamma \vdash \bot : \star} \qquad \frac{\Psi;\, \Gamma \vdash \tau_1 : \star \qquad \Psi;\, \Gamma \vdash \tau_2 : \star}{\Psi;\, \Gamma \vdash \tau_1 \to \tau_2 : \star}$$

$$\frac{\Psi;\, \Gamma \vdash \tau_1 : \star \qquad \Psi;\, \Gamma \vdash \tau_2 : \star}{\Psi;\, \Gamma \vdash \tau_1 \times \tau_2 : \star} \qquad \frac{\Psi;\, \Gamma \vdash \tau_1 : \star \qquad \Psi;\, \Gamma \vdash \tau_2 : \star}{\Psi;\, \Gamma \vdash \tau_1 + \tau_2 : \star} \qquad \frac{\Psi \vdash k \text{ wf} \qquad \Psi;\, \Gamma,\, \alpha : k \vdash \tau : k}{\Psi;\, \Gamma \vdash \mu\alpha : k.\tau : k}$$

$$\frac{\Psi;\, \Gamma \vdash \tau : \star}{\Psi;\, \Gamma \vdash \mathsf{ref}\, \tau : \star} \qquad \frac{\Psi \vdash k \text{ wf} \qquad \Psi;\, \Gamma,\, \alpha : k \vdash \tau : \star}{\Psi;\, \Gamma \vdash \forall\alpha : k.\tau : \star} \qquad \frac{\Psi \vdash k \text{ wf} \qquad \Psi;\, \Gamma,\, \alpha : k \vdash \tau : k'}{\Psi;\, \Gamma \vdash \lambda\alpha : k.\tau : k \to k'}$$

$$\frac{\Psi;\, \Gamma \vdash \tau_1 : k \to k' \qquad \Psi;\, \Gamma \vdash \tau_2 : k}{\Psi;\, \Gamma \vdash \tau_1\, \tau_2 : k'} \qquad \frac{(\alpha : k) \in \Gamma}{\Psi;\, \Gamma \vdash \alpha : k}$$

$\boxed{\Psi;\, \Sigma;\, \Gamma \vdash e : \tau}$

$$\frac{\Psi,\, K;\, \Sigma;\, \Gamma \vdash \lceil e \rceil_{|\Psi|,1} : \tau}{\Psi;\, \Sigma;\, \Gamma \vdash \Lambda(K).e : \Pi(K).\, \lfloor \tau \rfloor_{|\Psi|,1}} \qquad \frac{\Psi;\, \Sigma;\, \Gamma \vdash e : \Pi(K).\tau \qquad \Psi \vdash T : K}{\Psi;\, \Sigma;\, \Gamma \vdash e\, T : \lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, T)}$$

$$\frac{\Psi \vdash T : K \qquad \Psi,\, K;\, \Gamma \vdash \lceil \tau \rceil_{|\Psi|,1} : \star \qquad \Psi;\, \Sigma;\, \Gamma \vdash e : \lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, T)}{\Psi;\, \Sigma;\, \Gamma \vdash \mathsf{pack}\, T\, \mathsf{return}\, (.\tau)\, \mathsf{with}\, e : \Sigma(K).\tau}$$

$$\frac{\Psi;\, \Sigma;\, \Gamma \vdash e : \Sigma(K).\tau \qquad \Psi,\, K,\, \Sigma;\, \Gamma,\, x : \lceil \tau \rceil_{|\Psi|,1} \vdash \lceil e' \rceil_{|\Psi|,1} : \tau' \qquad \Psi;\, \Gamma \vdash \tau' : \star}{\Psi;\, \Sigma;\, \Gamma \vdash \mathsf{unpack}\, e\, (.)x.(e') : \tau'} \qquad \frac{}{\Psi;\, \Sigma;\, \Gamma \vdash () : \mathsf{unit}}$$

$$\frac{}{\Psi;\, \Sigma;\, \Gamma \vdash \mathsf{error} : \tau} \qquad \frac{\Psi;\, \Sigma;\, \Gamma,\, x : \tau \vdash e : \tau'}{\Psi;\, \Sigma;\, \Gamma \vdash \lambda x : \tau.e : \tau \to \tau'}$$

$$\frac{\Psi;\,\Sigma;\,\Gamma \vdash e : \tau \to \tau' \qquad \Psi;\,\Sigma;\,\Gamma \vdash e' : \tau}{\Psi;\,\Sigma;\,\Gamma \vdash e\ e' : \tau'} \qquad \frac{(x : \tau) \in \Gamma}{\Psi;\,\Sigma;\,\Gamma \vdash x : \tau} \qquad \frac{\Psi;\,\Sigma;\,\Gamma \vdash e_1 : \tau_1 \qquad \Psi;\,\Sigma;\,\Gamma \vdash e_2 : \tau_2}{\Psi;\,\Sigma;\,\Gamma \vdash (e_1,\, e_2) : \tau_1 \times \tau_2}$$

$$\frac{\Psi;\,\Sigma;\,\Gamma \vdash e : \tau_1 \times \tau_2 \qquad i = 1 \text{ or } 2}{\Psi;\,\Sigma;\,\Gamma \vdash \mathsf{proj}_i\ e : \tau_i} \qquad \frac{\Psi;\,\Sigma;\,\Gamma \vdash e : \tau_i \qquad i = 1 \text{ or } 2}{\Psi;\,\Sigma;\,\Gamma \vdash \mathsf{inj}_i\ e : \tau_1 + \tau_2}$$

$$\frac{\Psi;\,\Sigma;\,\Gamma \vdash e : \tau_1 + \tau_2 \qquad \Psi;\,\Sigma;\,\Gamma,\, x : \tau_1 \vdash e_1 : \tau \qquad \Psi;\,\Sigma;\,\Gamma,\, x : \tau_2 \vdash e_2 : \tau}{\Psi;\,\Sigma;\,\Gamma \vdash \mathsf{case}(e, x.e_1, x.e_2) : \tau}$$

$$\frac{\Psi;\,\Gamma \vdash \mu\alpha : k.\tau : k \qquad \Psi;\,\Sigma;\,\Gamma \vdash e : \tau[\mu\alpha : k.\tau/\alpha]\ \tau_1\ \tau_2\ \cdots\ \tau_n}{\Psi;\,\Sigma;\,\Gamma \vdash \mathsf{fold}\ e : (\mu\alpha : k.\tau)\ \tau_1\ \tau_2\ \cdots\ \tau_n}$$

$$\frac{\Psi;\,\Gamma \vdash \mu\alpha : k.\tau : k \qquad \Psi;\,\Sigma;\,\Gamma \vdash e : (\mu\alpha : k.\tau)\ \tau_1\ \tau_2\ \cdots\ \tau_n}{\Psi;\,\Sigma;\,\Gamma \vdash \mathsf{unfold}\ e : \tau[\mu\alpha : k.\tau/\alpha]\ \tau_1\ \tau_2\ \cdots\ \tau_n} \qquad \frac{\Psi;\,\Sigma;\,\Gamma \vdash e : \tau}{\Psi;\,\Sigma;\,\Gamma \vdash \mathsf{ref}\ e : \mathsf{ref}\ \tau}$$

$$\frac{\Psi;\,\Sigma;\,\Gamma \vdash e : \mathsf{ref}\ \tau \qquad \Psi;\,\Sigma;\,\Gamma \vdash e' : \tau}{\Psi;\,\Sigma;\,\Gamma \vdash e := e' : \mathsf{unit}} \qquad \frac{\Psi;\,\Sigma;\,\Gamma \vdash e : \mathsf{ref}\ \tau}{\Psi;\,\Sigma;\,\Gamma \vdash !e : \tau} \qquad \frac{(l : \tau) \in \Sigma}{\Psi;\,\Sigma;\,\Gamma \vdash l : \mathsf{ref}\ \tau}$$

$$\frac{\Psi;\,\Sigma;\,\Gamma,\, \alpha : k \vdash e : \tau}{\Psi;\,\Sigma;\,\Gamma \vdash \Lambda\alpha : k.e : \Pi\alpha : k.\tau} \qquad \frac{\Psi;\,\Sigma;\,\Gamma \vdash e : \Pi\alpha : k.\tau' \qquad \Psi;\,\Gamma \vdash \tau : k}{\Psi;\,\Sigma;\,\Gamma \vdash e\ \tau : \tau'[\tau/\alpha]} \qquad \frac{\Psi;\,\Sigma;\,\Gamma,\, x : \tau \vdash e : \tau}{\Psi;\,\Sigma;\,\Gamma \vdash \mathsf{fix}\ x : \tau.e : \tau}$$

$$\frac{\begin{array}{c}\Psi \vdash T : K \qquad \Psi,\, K;\,\Gamma \vdash \lceil\tau\rceil_{|\Psi|,1} : \star \qquad \Psi \vdash \lceil\Psi'\rceil_{|\Psi|}\ \mathrm{wf} \\ \Psi,\, \lceil\Psi'\rceil_{|\Psi|} \vdash \lceil T'\rceil_{|\Psi|,|\Psi'|} : K \qquad \Psi,\, \lceil\Psi'\rceil_{|\Psi|};\,\Sigma;\,\Gamma \vdash \lceil e'\rceil_{|\Psi|,|\Psi'|} : \lceil\tau\rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, \lceil T'\rceil_{|\Psi|,|\Psi'|})\end{array}}{\Psi;\,\Sigma;\,\Gamma \vdash \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi'.T' \mapsto e') : (\lceil\tau\rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, T)) + \mathsf{unit}}$$

$\boxed{\Psi \vdash \Gamma\ \mathrm{wf}}$

$$\frac{}{\Psi \vdash \bullet\ \mathrm{wf}} \qquad \frac{\Psi \vdash \Gamma\ \mathrm{wf} \qquad \Psi;\,\Gamma \vdash k\ \mathrm{wf}}{\Psi \vdash (\Gamma,\, \alpha : k)\ \mathrm{wf}} \qquad \frac{\Psi \vdash \Gamma\ \mathrm{wf} \qquad \Psi;\,\Gamma \vdash \tau : \star}{\Psi \vdash (\Gamma,\, x : \tau)\ \mathrm{wf}}$$

$\boxed{\vdash \Sigma\ \mathrm{wf}}$

$$\frac{}{\vdash \bullet\ \mathrm{wf}} \qquad \frac{\vdash \Sigma\ \mathrm{wf} \qquad \bullet;\,\bullet \vdash \tau : \star}{\vdash (\Sigma,\, l : \tau)}$$

Definition C.5 β*-equivalence for types* τ *is the symmetric, reflexive, transitive congruence closure of the following relation. Types of the language are viewed implicitly up to* β*-equivalence. This means that the lemmas that we prove about types need to agree on* β*-equivalent types.*

$$(\lambda\alpha : \mathcal{K}.\tau)\ \tau' \quad = \quad \tau[\tau'/\alpha]$$

Definition C.6 *Small-step operational semantics for the language are defined below.*

$$\begin{array}{l} v ::= \Lambda(K).e \mid \mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ v \mid () \mid \lambda x : \tau.e \mid (v,\, v') \mid \mathsf{inj}_i\ v \mid \mathsf{fold}\ v \mid l \mid \Lambda\alpha : k.e \\ \mathcal{E} ::= \bullet \mid \mathcal{E}\ T \mid \mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ \mathcal{E} \mid \mathsf{unpack}\ \mathcal{E}\ (.)x.(e') \mid \mathcal{E}\ e' \mid v\ \mathcal{E} \mid (\mathcal{E},\, e) \mid (v,\, \mathcal{E}) \mid \mathsf{proj}_i\ \mathcal{E} \mid \mathsf{inj}_i\ \mathcal{E} \\ \qquad \mid \mathsf{case}(\mathcal{E}, x.e_1, x.e_2) \mid \mathsf{fold}\ \mathcal{E} \mid \mathsf{unfold}\ \mathcal{E} \mid \mathsf{ref}\ \mathcal{E} \mid \mathcal{E} := e' \mid v := \mathcal{E} \mid !\mathcal{E} \mid \mathcal{E}\ \tau \\ \mu ::= \bullet \mid \mu,\, l \mapsto v \end{array}$$

$\boxed{(\,\mu\,,\,e\,) \longrightarrow ((\,\mu\,,\,e'\,)|\mathsf{error})}$

$$\frac{(\,\mu\,,\,e\,) \longrightarrow (\,\mu'\,,\,e'\,)}{(\,\mu\,,\,\mathcal{E}[e]\,) \longrightarrow (\,\mu'\,,\,\mathcal{E}[e']\,)} \qquad (\,\mu\,,\,\mathcal{E}[\mathsf{error}]\,) \longrightarrow \mathsf{error} \qquad (\,\mu\,,\,(\Lambda(K).e)\;T\,) \longrightarrow (\,\mu\,,\,\lceil e\rceil_{0,1}\cdot T\,)$$

$$(\,\mu\,,\,\mathsf{unpack}\;\langle T,\,\tau\rangle\,v\,(.)x.(e')\,) \longrightarrow (\,\mu\,,\,(\lceil e'\rceil_{0,1}\cdot T)[v/x]\,) \qquad (\,\mu\,,\,(\lambda x:\tau.e)\;v\,) \longrightarrow (\,\mu\,,\,e[v/x]\,)$$

$$(\,\mu\,,\,\mathsf{proj}_i(v_1,\,v_2)\,) \longrightarrow (\,\mu\,,\,v_i\,) \qquad (\,\mu\,,\,\mathsf{case}(\mathsf{inj}_i\;v, x.e_1, x.e_2)\,) \longrightarrow (\,\mu\,,\,e_i[v/x]\,)$$

$$(\,\mu\,,\,\mathsf{unfold}\;(\mathsf{fold}\;v)\,) \longrightarrow (\,\mu\,,\,v\,) \qquad \frac{\neg(l \mapsto \_ \in \mu)}{(\,\mu\,,\,\mathsf{ref}\;v\,) \longrightarrow (\,(\mu,\,l\mapsto v)\,,\,l\,)} \qquad \frac{l \mapsto \_ \in \mu}{(\,\mu\,,\,l := v\,) \longrightarrow (\,\mu[l\mapsto v]\,,\,()\,)}$$

$$\frac{l \mapsto v \in \mu}{(\,\mu\,,\,!l\,) \longrightarrow (\,\mu\,,\,v\,)} \qquad (\,\mu\,,\,(\Lambda\alpha:k.e)\;\tau\,) \longrightarrow (\,\mu\,,\,e[\tau/\alpha]\,) \qquad (\,\mu\,,\,\mathsf{fix}\;x:\tau.e\,) \longrightarrow (\,\mu\,,\,e[\mathsf{fix}\;x:\tau.e/x]\,)$$

$$\frac{\exists\sigma_\Psi.(\bullet \vdash \sigma_\Psi : \lceil\Psi\rceil_0 \quad\wedge\quad \lceil T'\rceil_{0,|\Psi|}\cdot\sigma_\Psi = T)}{(\,\mu\,,\,\mathsf{unify}\;T\;\mathsf{return}\;(.\tau)\;\mathsf{with}\;(\Psi.T' \mapsto e')\,) \longrightarrow (\,\mu\,,\,\mathsf{inj}_1\;(\lceil e'\rceil_{0,|\Psi|}\cdot\sigma_\Psi)\,)}$$

$$\frac{\nexists\sigma_\Psi.(\bullet \vdash \sigma_\Psi : \lceil\Psi\rceil_0 \quad\wedge\quad \lceil T'\rceil_{0,|\Psi|}\cdot\sigma_\Psi = T)}{(\,\mu\,,\,\mathsf{unify}\;T\;\mathsf{return}\;(.\tau)\;\mathsf{with}\;(\Psi.T' \mapsto e')\,) \longrightarrow (\,\mu\,,\,\mathsf{inj}_2\;()\,)}$$

$\boxed{(l\mapsto v)\in\mu}$

$$\begin{array}{rcl}(l\mapsto v) & \in & (\mu,\,l\mapsto v)\\ (l\mapsto v) & \in & (\mu,\,l'\mapsto v') \Leftarrow (l\mapsto v)\in\mu\end{array}$$

$\boxed{(l:\tau)\in\Sigma}$

$$\begin{array}{rcl}(l:\tau) & \in & (\Sigma,\,l:\tau)\\ (l:\tau) & \in & (\Sigma,\,l':\tau') \Leftarrow (l:\tau)\in\Sigma\end{array}$$

$\boxed{\mu\sim\Sigma}$

$$\begin{array}{rcl}(l\mapsto v)\in\mu & \Rightarrow & \exists\tau.(l:\tau)\in\Sigma\wedge\bullet;\,\Sigma;\,\bullet\vdash v:\tau\\ (l:\tau)\in\Sigma & \Rightarrow & \exists v.(l\mapsto v)\in\mu\wedge\bullet;\,\Sigma;\,\bullet\vdash v:\tau\end{array}$$

$\boxed{\Sigma\subseteq\Sigma'}$

$$(l:\tau)\in\Sigma \quad\Rightarrow\quad (l:\tau)\in\Sigma'$$

$\boxed{\mu[l:=v]}$

$$\begin{array}{rcl}(\mu,\,l'\mapsto v')[l:=v] & = & \mu[l:=v],\,l'\mapsto v'\\ (\mu,\,l\mapsto v')[l:=v] & = & \mu, l\mapsto v\end{array}$$

Lemma C.7 (Computational substitution commutes with logic operations) *1.* $\lceil\tau[\tau'/\alpha]\rceil^M_{N,K} = \lceil\tau\rceil^M_{N,K}[\lceil\tau'\rceil^M_{N,K}/\alpha]$

2. $\lfloor \tau[\tau'/\alpha] \rfloor^M_{N,K} = \lfloor \tau \rfloor^M_{N,K}[\lfloor \tau' \rfloor^M_{N,K}/\alpha]$
3. $(\tau[\tau'/\alpha]) \cdot \sigma_\Psi = \tau \cdot \sigma_\Psi[\tau' \cdot \sigma_\Psi/\alpha]$
4. $\lceil e[\tau/\alpha] \rceil^M_{N,K} = \lceil e \rceil^M_{N,K}[\lceil \tau \rceil^M_{N,K}/\alpha]$
5. $\lfloor e[\tau/\alpha] \rfloor^M_{N,K} = \lfloor e \rfloor^M_{N,K}[\lfloor \tau \rfloor^M_{N,K}/\alpha]$
6. $(e[\tau/\alpha]) \cdot \sigma_\Psi = e \cdot \sigma_\Psi[\tau \cdot \sigma_\Psi/\alpha]$
7. $\lceil e[e'/x] \rceil^M_{N,K} = \lceil e \rceil^M_{N,K}[\lceil e' \rceil^M_{N,K}/x]$
8. $\lfloor e[e'/x] \rfloor^M_{N,K} = \lfloor e \rfloor^M_{N,K}[\lfloor e' \rfloor^M_{N,K}/x]$
9. $(e[e'/x]) \cdot \sigma_\Psi = e \cdot \sigma_\Psi[e' \cdot \sigma_\Psi/x]$

Simple by induction.

Lemma C.8 (Compatibility of β conversion with logic operations) *If $\tau =_\beta \tau'$ then:*

1. $\lceil \tau \rceil^M_{N,K} =_\beta \lceil \tau' \rceil^M_{N,K}$
2. $\lfloor \tau \rfloor^M_{N,K} =_\beta \lceil \tau' \rceil^M_{N,K}$
3. $\tau \cdot \sigma_\Psi =_\beta \tau' \cdot \sigma_\Psi$

In all cases it's trivially provable by expansion for $\tau = (\lambda\alpha : k.\tau_1)\tau_2$ and $\tau' = \tau_1[\tau_2/\alpha]$ and use of lemma C.7. The congruence cases for other τ, τ' are provable by induction hypothesis.

Lemma C.9 *Assuming $|\sigma_\Psi| = N$, $\lceil \cdot \rceil^M_{N,K}$ and $\lceil \cdot \rceil^M_{N',K}$ are well-defined for their respective arguments, we have:*

1. $\lceil k \cdot \sigma_\Psi \rceil^M_{N',K} = \lceil k \rceil^M_{N,K} \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})$
2. $\lceil \tau \cdot \sigma_\Psi \rceil^M_{N',K} = \lceil \tau \rceil^M_{N,K} \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})$
3. $\lceil e \cdot \sigma_\Psi \rceil^M_{N',K} = \lceil e \rceil^M_{N,K} \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})$

By structural induction. We prove only the interesting cases.

Part 1 When $k = \Pi(K).k'$, we have that the left-hand-side is equal to:
$\lceil \Pi(K \cdot \sigma_\Psi).(k' \cdot \sigma_\Psi) \rceil^M_{N',K} = \Pi(\lceil K \cdot \sigma_\Psi \rceil^M_{N',K}).\lceil k' \cdot \sigma_\Psi \rceil^{M+1}_{N',K}$
We have by lemma B.103 that $\lceil K \cdot \sigma_\Psi \rceil^M_{N',K} = \lceil K \rceil^M_{N,K} \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})$.
By induction hypothesis we have that $\lceil k' \cdot \sigma_\Psi \rceil^{M+1}_{N',K} = \lceil k' \rceil^{M+1}_{N,K} \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})$.
After expansion of freshening for the right-hand-side, we see that it is equal to the above.

Part 3 The most interesting case occurs when $e = \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi.T' \mapsto e')$. We have that the left-hand-side is equal to:
$\lceil \mathsf{unify}\ T \cdot \sigma_\Psi\ \mathsf{return}\ (.\tau \cdot \sigma_\Psi)\ \mathsf{with}\ (\Psi \cdot \sigma_\Psi.T' \cdot \sigma_\Psi \mapsto e' \cdot \sigma_\Psi) \rceil^M_{N',K}$
By expansion of the definition of freshening we get that this is equal to:
$\mathsf{unify}\ \lceil T \cdot \sigma_\Psi \rceil^M_{N',K}\ \mathsf{return}\ (.\lceil \tau \cdot \sigma_\Psi \rceil^{M+1}_{N',K})\ \mathsf{with}\ (\lceil \Psi \cdot \sigma_\Psi \rceil^M_{N',K}.\lceil T' \rceil^{M+|\Psi \cdot \sigma_\Psi|}_{N,K} \mapsto \lceil e' \rceil^{M+|\Psi \cdot \sigma_\Psi|}_{N',K})$
The right-hand-side is equal to:
(assuming $\sigma'_\Psi = \sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1}$)
$\mathsf{unify}\ \lceil T \rceil^M_{N,K} \cdot \sigma'_\Psi\ \mathsf{return}\ (.\lceil \tau \rceil^{M+1}_{N,K} \cdot \sigma'_\Psi)\ \mathsf{with}\ (\lceil \Psi \rceil^M_{N,K} \cdot \sigma'_\Psi.\lceil T' \rceil^{M+|\Psi|}_{N,K} \cdot \sigma'_\Psi \mapsto \lceil e' \rceil^{M+|\Psi|}_{N,K} \cdot \sigma'_\Psi)$
In all cases, the respective terms match, by use of induction hypothesis, lemma B.103, and also the fact that $|\Psi| = |\Psi \cdot \sigma_\Psi|$.

Lemma C.10 *Assuming $|\sigma_\Psi| = N$, $\lfloor\cdot\rfloor^M_{N,K}$ and $\lfloor\cdot\rfloor^M_{N',K}$ are well-defined for their respective arguments, we have:*

1. $\lfloor k \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})\rfloor^M_{N',K} = \lfloor k\rfloor^M_{N,K} \cdot \sigma_\Psi$
2. $\lfloor \tau \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})\rfloor^M_{N',K} = \lfloor \tau\rfloor^M_{N,K} \cdot \sigma_\Psi$
3. $\lfloor e \cdot (\sigma_\Psi, X_{N'}, \cdots, X_{N'+K-1})\rfloor^M_{N',K} = \lfloor e\rfloor^M_{N,K} \cdot \sigma_\Psi$

Similarly as above, and use of lemma B.104.

Lemma C.11 1. $(k \cdot \sigma_\Psi) \cdot \sigma'_\Psi = k \cdot (\sigma_\Psi \cdot \sigma'_\Psi)$
2. $(\tau \cdot \sigma_\Psi) \cdot \sigma'_\Psi = \tau \cdot (\sigma_\Psi \cdot \sigma'_\Psi)$
3. $(e \cdot \sigma_\Psi) \cdot \sigma'_\Psi = e \cdot (\sigma_\Psi \cdot \sigma'_\Psi)$

By induction, and use of lemma B.94.

Lemma C.12 *If $\sigma_\Psi \subseteq \sigma'_\Psi$ then:*

1. *If $k \cdot \sigma_\Psi$ is defined, then $k \cdot \sigma_\Psi = k \cdot \sigma'_\Psi$*
2. *If $\tau \cdot \sigma_\Psi$ is defined, then $\tau \cdot \sigma_\Psi = \tau \cdot \sigma'_\Psi$*
3. *If $e \cdot \sigma_\Psi$ is defined, then $e \cdot \sigma_\Psi = e \cdot \sigma'_\Psi$*
4. *If $\Gamma \cdot \sigma_\Psi$ is defined, then $\Gamma \cdot \sigma_\Psi = \Gamma \cdot \sigma'_\Psi$*

Most are trivial based on induction and use of lemma B.92

Lemma C.13 1. *If $\Psi \vdash k$ wf and $\Psi' \vdash \sigma_\Psi : \Psi$ then $\Psi' \vdash k \cdot \sigma_\Psi$ wf.*
2. *If $\Psi;\ \Gamma \vdash \tau : k$ and $\Psi' \vdash \sigma_\Psi : \Psi$ then $\Psi';\ \Gamma \cdot \sigma_\Psi \vdash \tau \cdot \sigma_\Psi : k \cdot \sigma_\Psi$.*
3. *If $\Psi;\ \Sigma;\ \Gamma \vdash e : \tau$ and $\Psi' \vdash \sigma_\Psi : \Psi$ then $\Psi';\ \Sigma;\ \Gamma \cdot \sigma_\Psi \vdash e \cdot \sigma_\Psi : \tau \cdot \sigma_\Psi$.*

We only prove the interesting cases.

Part 1 **Case** $\dfrac{\vdash \Psi,\ K\ \text{wf} \qquad \Psi,\ K \vdash \lceil k\rceil_{|\Psi|,1}\ \text{wf}}{\Psi \vdash \Pi(K).k\ \text{wf}}\ \triangleright$

We use the induction hypothesis for $\Psi',\ K \cdot \sigma_\Psi \vdash (\sigma_\Psi,\ X_{|\Psi'|}) : \Psi,\ K$ and $\lceil k\rceil$ to get:
$\Psi',\ K \cdot \sigma_\Psi \vdash \lceil k\rceil_{|\Psi|,1} \cdot (\sigma_\Psi,\ X_{|\Psi'|})$ wf.
From lemma C.9 we have that $\lceil k\rceil_{|\Psi|,1} \cdot (\sigma_\Psi,\ X_{|\Psi'|}) = \lceil k \cdot \sigma_\Psi\rceil_{|\Psi'|,1}$.
Therefore by use of the same typing rule we have the desired result.

Part 2 **Case** $\dfrac{\Psi,\ K;\ \Gamma \vdash \lceil \tau\rceil_{|\Psi|,1} : k}{\Psi;\ \Gamma \vdash \lambda(K).\tau : \Pi(K).\ \lfloor k\rfloor_{|\Psi|,1}}\ \triangleright$

We use the induction hypothesis for $\Psi',\ K \vdash (\sigma_\Psi,\ X_{|\Psi'|}) : \Psi,\ K$ and $\lceil \tau\rceil$ to get, together with lemma C.9:
$\Psi',\ K \cdot \sigma_\Psi;\ \Gamma \cdot (\sigma_\Psi,\ X_{|\Psi'|}) \vdash \lceil \tau \cdot \sigma_\Psi\rceil_{|\Psi'|,1}$
By C.12 and the fact that $\Psi \vdash \Gamma$ wf, we have that $\Gamma \cdot (\sigma_\Psi,\ X_{|\Psi'|}) = \Gamma \cdot \sigma_\Psi$, so:
$\Psi',\ K \cdot \sigma_\Psi;\ \Gamma \cdot \sigma_\Psi \vdash \lceil \tau \cdot \sigma_\Psi\rceil_{|\Psi'|,1} : k \cdot (\sigma_\Psi,\ X_{|\Psi'|} \cdot \sigma_\Psi)$
By use of the same typing rule we get:
$\Psi';\ \Gamma \cdot \sigma_\Psi \vdash \lambda(K \cdot \sigma_\Psi).(\tau \cdot \sigma_\Psi) : \Pi(K \cdot \sigma_\Psi).(\lfloor k \cdot (\sigma_\Psi,\ X_{|\Psi'|})\rfloor_{|\Psi'|,1})$.
We have that $\lfloor k \cdot (\sigma_\Psi,\ X_{|\Psi'|})\rfloor_{|\Psi'|,1} = \lfloor k\rfloor_{|\Psi|,1} \cdot \sigma_\Psi$ by lemma C.10, so this is the desired result.

Case $\dfrac{\Psi;\,\Gamma \vdash \tau : \Pi(K).k \qquad \Psi \vdash T : K}{\Psi;\,\Gamma \vdash \tau\, T : \lceil k \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, T)}\ \triangleright$

By induction hypothesis we have:
$\Psi';\, \Gamma \cdot \sigma_\Psi \vdash \tau \cdot \sigma_\Psi : \Pi(K \cdot \sigma_\Psi).(k \cdot \sigma_\Psi)$
By use of B.97 for T we have:
$\Psi' \vdash T \cdot \sigma_\Psi : K \cdot \sigma_\Psi$
By use of the same typing rule we get:
$\Psi';\, \Gamma \cdot \sigma_\Psi \vdash (\tau \cdot \sigma_\Psi)\ (T \cdot \sigma_\Psi) : \lceil k \cdot \sigma_\Psi \rceil_{|\Psi'|,1} \cdot (\mathsf{id}_{\Psi'},\, T \cdot \sigma_\Psi)$
Now we need to prove that $\lceil k \cdot \sigma_\Psi \rceil_{|\Psi'|,1} \cdot (\mathsf{id}_{\Psi'},\, T \cdot \sigma_\Psi) = (\lceil k \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, T)) \cdot \sigma_\Psi$.
From lemma C.9, we get that the left-hand side is equal to:
$(\lceil k \rceil_{|\Psi|,1} \cdot (\sigma_\Psi,\, X_{|\Psi'|})) \cdot (\mathsf{id}_{\Psi'},\, T \cdot \sigma_\Psi)$.
By application of lemma C.11 we get that it is further equal to:
$(\lceil k \rceil_{|\Psi|,1}) \cdot ((\sigma_\Psi,\, X_{|\Psi'|}) \cdot (\mathsf{id}_{\Psi'},\, T \cdot \sigma_\Psi))$.
By application of the same lemma to the right-hand side we have that it is equal to: $(\lceil k \rceil_{|\Psi|,1}) \cdot$
$((\mathsf{id}_{|\Psi|},\, T) \cdot \sigma_\Psi)$.
Thus we only need to prove that $(\sigma_\Psi,\, X_{|\Psi'|}) \cdot (\mathsf{id}_{\Psi'},\, T \cdot \sigma_\Psi) = (\mathsf{id}_{|\Psi|},\, T) \cdot \sigma_\Psi$.
We have that the left-hand side is equal to:
$\sigma_\Psi \cdot (\mathsf{id}_{\Psi'},\, T \cdot \sigma_\Psi),\, T \cdot \sigma_\Psi = \sigma_\Psi \cdot \mathsf{id}_{\Psi'},\, T \cdot \sigma_\Psi$ by lemma B.92.
Furthermore by lemma B.96 we have that $\sigma_\Psi \cdot \mathsf{id}_{\Psi'} = \sigma_\Psi$.
The right-hand side is equal to:
$\mathsf{id}_{|\Psi|} \cdot \sigma_\Psi,\, T \cdot \sigma_\Psi = \sigma_\Psi,\, T \cdot \sigma_\Psi$ due to lemma B.95.

Part 3 Most cases are proved as above, using the above lemmas. The most difficult case is the pattern matching construct.

$$\textbf{Case}\ \frac{\begin{array}{c}\Psi \vdash T : K \qquad \Psi,\, K;\, \Gamma \vdash \lceil \tau \rceil_{|\Psi|,1} : \star \qquad \Psi \vdash \lceil \Psi'' \rceil_{|\Psi|}\ \mathrm{wf} \\ \Psi,\, \lceil \Psi'' \rceil_{|\Psi|} \vdash \lceil T' \rceil_{|\Psi|,|\Psi''|} : K \qquad \Psi,\, \lceil \Psi'' \rceil_{|\Psi|};\, \Sigma;\, \Gamma \vdash \lceil e' \rceil_{|\Psi|,|\Psi''|} : \lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, \lceil T' \rceil_{|\Psi|,|\Psi''|})\end{array}}{\Psi;\, \Sigma;\, \Gamma \vdash \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi''.T' \mapsto e') : (\lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, T)) + \mathsf{unit}}\ \triangleright$$

From $\Psi \vdash T : K$ and lemma B.97 we have:
$\Psi' \vdash T \cdot \sigma_\Psi : K \cdot \sigma_\Psi$
From $\Psi,\, K;\, \Gamma \vdash \lceil \tau \rceil_{|\Psi|,1} : \star$, $\Psi \vdash \Gamma$ wf, part 2 and lemma C.9 we have:
$\Psi',\, K \cdot \sigma_\Psi;\, \Gamma \cdot \sigma_\Psi \vdash \lceil \tau \cdot \sigma_\Psi \rceil_{|\Psi'|,1} : \star$
From $\Psi \vdash \lceil \Psi'' \rceil_{|\Psi|}$ wf and lemmas B.98 and B.106 we have:
$\Psi' \vdash \lceil \Psi'' \cdot \sigma_\Psi \rceil_{|\Psi'|}$ wf
From $\Psi,\, \lceil \Psi'' \rceil_{|\Psi|};\, \Sigma;\, \Gamma \vdash \lceil e' \rceil_{|\Psi|,|\Psi''|} : \lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, T')$, $\sigma'_\Psi = \sigma_\Psi,\, X_{|\Psi'|},\, \cdots,\, X_{|\Psi'|+|\Psi'' \cdot \sigma_\Psi|-1}$ and $\Psi',\, \lceil \Psi'' \cdot \sigma_\Psi \rceil_{|\Psi'|} \vdash$
$\sigma'_\Psi : (\Psi,\, \lceil \Psi'' \rceil_{|\Psi|})$, lemma B.97, lemma B.103, and lemma B.92 we have:
$\Psi',\, \lceil \Psi'' \cdot \sigma_\Psi \rceil_{|\Psi'|} \vdash \lceil T' \cdot \sigma_\Psi \rceil_{|\Psi'|,|\Psi'' \cdot \sigma_\Psi|} : K \cdot \sigma_\Psi$
Similarly for the same σ'_Ψ, and from $\Psi,\, \lceil \Psi'' \rceil_{|\Psi|};\, \Sigma;\, \Gamma \vdash \lceil e' \rceil_{|\Psi|,|\Psi''|} : \lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, \lceil T' \rceil_{|\Psi|,|\Psi''|})$, lemma C.9 and induction hypothesis, we get:
$\Psi',\, \lceil \Psi'' \cdot \sigma_\Psi \rceil_{|\Psi'|};\, \Sigma;\, \Gamma \cdot \sigma_\Psi \vdash \lceil e' \cdot \sigma_\Psi \rceil_{|\Psi'|,|\Psi'' \cdot \sigma_\Psi|} : (\lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, \lceil T' \rceil_{|\Psi|,|\Psi''|})) \cdot \sigma'_\Psi$.
Thus we only need to prove that $(\lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, \lceil T' \rceil_{|\Psi|,|\Psi''|})) \cdot \sigma'_\Psi = \lceil \tau \cdot \sigma_\Psi \rceil_{|\Psi'|,1} \cdot (\mathsf{id}_{\Psi'},\, \lceil T' \cdot \sigma_\Psi \rceil_{|\Psi'|,|\Psi'' \cdot \sigma_\Psi|})$.
In that case we will use the same typing rule to get the desired result, using a similar proof as this last step, to go from $\lceil \tau \cdot \sigma_\Psi \rceil_{|\Psi'|,1} \cdot (\mathsf{id}_{\Psi'},\, T \cdot \sigma_\Psi)$ to $(\lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, T)) \cdot \sigma_\Psi$.

So we now prove $(\lceil\tau\rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\ \lceil T'\rceil_{|\Psi|,|\Psi''|})) \cdot \sigma'_\Psi = \lceil\tau \cdot \sigma_\Psi\rceil_{|\Psi'|,1} \cdot (\mathsf{id}_{\Psi'},\ \lceil T' \cdot \sigma_\Psi\rceil_{|\Psi'|,|\Psi'' \cdot \sigma_\Psi|})$:
By lemma C.9 and lemma B.103, we have that the right-hand side is equal to:
$(\lceil\tau\rceil_{|\Psi|,1} \cdot (\sigma_\Psi,\ X_{|\Psi'|})) \cdot (\mathsf{id}_{\Psi'},\ \lceil T'\rceil_{|\Psi|,|\Psi''|} \cdot \sigma'_\Psi)$.
By application of lemma C.11 we see that both sides are equal if $(\sigma_\Psi,\ X_{|\Psi'|}) \cdot (\mathsf{id}_{\Psi'},\ \lceil T'\rceil_{|\Psi|,|\Psi''|} \cdot \sigma'_\Psi) = (\mathsf{id}_\Psi,\ \lceil T'\rceil_{|\Psi|,|\Psi''|}) \cdot \sigma'_\Psi$.
The left-hand side of this is equal to $\sigma_\Psi \cdot (\mathsf{id}_{\Psi'},\ \lceil T'\rceil_{|\Psi|,|\Psi''|} \cdot \sigma'_\Psi),\ \lceil T'\rceil_{|\Psi|,|\Psi''|} \cdot \sigma'_\Psi$.
By lemma B.92 and B.96 we get that this is further equal to $\sigma_\Psi,\ \lceil T'\rceil_{|\Psi|,|\Psi''|} \cdot \sigma'_\Psi$.
The right-hand side is equal to $\mathsf{id}_\Psi \cdot \sigma'_\Psi,\ \lceil T'\rceil_{|\Psi|,|\Psi''|} \cdot \sigma'_\Psi$, which is equal to the above using lemmas B.92 and B.95.

Lemma C.14 *1.* $\left\lceil \lfloor k \rfloor^M_{N,K} \right\rceil^M_{N,K} = k$

2. $\left\lceil \lfloor \tau \rfloor^M_{N,K} \right\rceil^M_{N,K} = \tau$

3. $\left\lceil \lfloor e \rfloor^M_{N,K} \right\rceil^M_{N,K} = e$

Trivial by induction and use of lemma B.105.

Lemma C.15 (Substitution) *1. If* $\Psi,\ \Psi';\ \Gamma,\ \alpha' : k',\ \Gamma' \vdash \tau : k$ *and* $\Psi;\ \Gamma \vdash \tau' : k'$ *then* $\Psi,\ \Psi';\ \Gamma,\ \Gamma'[\tau'/\alpha'] \vdash \tau[\tau'/\alpha'] : k$.

2. If $\Psi,\ \Psi';\ \Sigma\,\Gamma,\ \alpha' : k',\ \Gamma' \vdash e : \tau$ *and* $\Psi;\ \Gamma \vdash \tau' : k'$ *then* $\Psi,\ \Psi';\ \Sigma;\ \Gamma,\ \Gamma'[\tau'/\alpha'] \vdash e[\tau'/\alpha'] : \tau[\tau'/\alpha']$.

3. If $\Psi,\ \Psi';\ \Sigma\,\Gamma,\ x' : \tau',\ \Gamma' \vdash e : \tau$ *and* $\Psi;\ \Sigma;\ \Gamma \vdash e' : \tau'$ *then* $\Psi,\ \Psi';\ \Sigma;\ \Gamma,\ \Gamma' \vdash e[e'/x'] : \tau$.

Easily proved by structural induction on the typing derivations.

Let us now proceed to prove the main preservation theorem.

Theorem C.16 (Preservation) *If* $\bullet;\ \Sigma;\ \bullet \vdash e : \tau,\ \mu \sim \Sigma,\ (\ \mu\ ,\ e\ ) \longrightarrow (\ \mu'\ ,\ e'\ )$ *then there exists* Σ' *such that* $\Sigma \subseteq \Sigma',\ \mu' \sim \Sigma'$ *and* $\bullet;\ \Sigma';\ \bullet \vdash e' : \tau$.

Proceed by induction on the derivation of $(\ \mu\ ,\ e\ ) \longrightarrow (\ \mu'\ ,\ e\ )$. When we don't specify a different μ', we have that $\mu' = \mu$, with the desired properties obviously holding.

Case $\dfrac{(\ \mu\ ,\ e\ ) \longrightarrow (\ \mu'\ ,\ e'\ )}{(\ \mu\ ,\ \mathcal{E}[e]\ ) \longrightarrow (\ \mu'\ ,\ \mathcal{E}[e']\ )}\ \triangleright$
By induction hypothesis for $(\ \mu\ ,\ e\ ) \longrightarrow (\ \mu'\ ,\ e'\ )$ we get a Σ' such that $\Sigma \subseteq \Sigma',\ \mu' \sim \Sigma'$ and $\bullet;\ \Sigma;\ \bullet \vdash e' : \tau$. By inversion of typing for $\mathcal{E}[e]$ and re-application of the same typing rule for $\mathcal{E}[e']$ we get that $\bullet;\ \Sigma';\ \bullet \vdash \mathcal{E}[e'] : \tau$.

Case $(\ \mu\ ,\ (\Lambda(K).e)\ T\ ) \longrightarrow (\ \mu\ ,\ \lceil e\rceil_{0,1} \cdot T\ )\ \triangleright$
By inversion of typing we get:
$\bullet;\ \Sigma;\ \bullet \vdash \Lambda(K).e : \Pi(K).\tau'$
$\bullet \vdash T : K$
$\tau = \lceil\tau'\rceil_{0,1} \cdot T$
By further typing inversion for $\Lambda(K).e$ we get:
$\bullet,\ K;\ \Sigma;\ \bullet \vdash \lceil e\rceil_{0,1} : \tau''$
$\tau' = \lfloor\tau''\rfloor_{0,1}$
For $\sigma_\Psi = \bullet,\ T$ we have $\bullet \vdash (\bullet,\ T) : (\bullet,\ K)$ trivially from the above.
By lemma C.13 for σ_Ψ we get that:

$\bullet; \Sigma; \bullet \vdash \lceil e \rceil_{0,1} \cdot T : \tau'' \cdot T$.
Now it remains to show that $\tau'' \cdot T = \left\lceil \lfloor \tau'' \rfloor_{0,1} \right\rceil_{0,1} \cdot T$, which is proved by C.14.

Case $(\ \mu\ ,\ \mathsf{unpack}\ \langle T,\ \tau \rangle\ v\ (.)x.(e')\ ) \longrightarrow (\ \mu\ ,\ (\lceil e' \rceil_{0,1} \cdot T)[v/x]\ )\ \triangleright$
By inversion of typing we get:
$\bullet; \Sigma; \bullet \vdash \langle T,\ \tau'' \rangle\ v : \Sigma(K).\tau'$
$\bullet,\ K; \Sigma; x : \lceil \tau' \rceil_{0,1} \vdash \lceil e' \rceil_{0,1} : \tau$
$\bullet;\ \bullet \vdash \tau : \star$
By further typing inversion for $\langle T,\ \tau'' \rangle\ v$ we get:
$\tau'' = \tau'$
$\bullet \vdash T : K$
$\bullet,\ K; \bullet \vdash \lceil \tau' \rceil_{0,1} : \star$
$\bullet; \Sigma; \bullet \vdash v : \lceil \tau' \rceil_{0,1} \cdot (T)$
First by lemma C.13 for e', $\sigma_\Psi = T$ we get:
$\bullet; \Sigma; x : \lceil \tau' \rceil_{0,1} \cdot T \vdash \lceil e' \rceil_{0,1} \cdot T : \tau \cdot T$.
Second by lemma C.12 for τ we get that $\tau \cdot T = \tau$.
Thus $\bullet; \Sigma; x : \lceil \tau' \rceil_{0,1} \cdot T \vdash \lceil e' \rceil_{0,1} \cdot T : \tau$.
Furthermore by lemma C.15 for $[v/x]$ we get that $\bullet; \Sigma; \bullet \vdash (\lceil e' \rceil_{0,1} \cdot T)[v/x] : \tau$, which is the desired.

Case $(\ \mu\ ,\ (\lambda x : \tau.e)\ v\ ) \longrightarrow (\ \mu\ ,\ e[v/x]\ )\ \triangleright$
By inversion of typing we get:
$\bullet; \Sigma; \bullet \vdash \lambda x : \tau.e : \tau' \to \tau$
$\bullet; \Sigma; \bullet \vdash v : \tau'$
By further typing inversion for $\lambda x : \tau.e$ we get:
$\bullet; \Sigma; x : \tau \vdash e : \tau$
By lemma C.15 for $[v/x]$ we get:
$\bullet; \Sigma; \bullet \vdash e[v/x] : \tau$, which is the desired.

Case $(\ \mu\ ,\ \mathsf{proj}_i(v_1,\ v_2)\ ) \longrightarrow (\ \mu\ ,\ v_i\ )\ \triangleright$
By typing inversion we get:
$\bullet; \Sigma; \bullet \vdash (v_1,\ v_2) : \tau_1 \times \tau_2$
$\tau = \tau_i$
By further inversion for $(v_1,\ v_2)$ we have:
$\bullet; \Sigma; \bullet \vdash v_i : \tau_i$, which is the desired.

Case $(\ \mu\ ,\ \mathsf{case}(\mathsf{inj}_i\ v, x.e_1, x.e_2)\ ) \longrightarrow (\ \mu\ ,\ e_i[v/x]\ )\ \triangleright$
By typing inversion we get:
$\bullet; \Sigma; \bullet \vdash v : \tau_i$
$\bullet; \Sigma; x : \tau_i \vdash e_i : \tau$
Using the lemma C.15 for [v/x] we get:
$\bullet; \Sigma; \bullet \vdash e_i[v/x] : \tau$

Case $(\ \mu\ ,\ \mathsf{unfold}\ (\mathsf{fold}\ v)\ ) \longrightarrow (\ \mu\ ,\ v\ )\ \triangleright$
By inversion we get: $\bullet; \bullet \vdash \mu\alpha : k.\tau' : k$
$\bullet; \Sigma; \bullet \vdash \mathsf{fold}\ v : (\mu\alpha : k.\tau')\ \tau_1\ \tau_2\ \cdots\ \tau_n$
$\tau = \tau'[\mu\alpha : k.\tau']\ \tau_1\ \tau_2\ \cdots\ \tau_n$
By further typing inversion for $\mathsf{fold}\ v$:

$\bullet;\ \Sigma;\ \bullet \vdash v : \tau'[\mu\alpha : k.\tau/\alpha]\ \tau_1\ \tau_2\ \cdots\ \tau_n$
Which is the desired.

Case $\dfrac{l \mapsto \_ \notin \mu}{(\ \mu\ ,\ \mathsf{ref}\ v\ ) \longrightarrow (\ (\mu,\ l \mapsto v)\ ,\ l\ )}\ \triangleright$
By typing inversion we get: $\bullet;\ \Sigma;\ \bullet \vdash v : \tau$
For $\Sigma' = \Sigma,\ l : \tau$ and $\mu' = \mu,\ l \mapsto v$ we have that $\mu' \sim \Sigma'$ and $\bullet;\ \Sigma';\bullet \vdash l : \mathsf{ref}\ \tau$.

Case $\dfrac{l \mapsto \_ \in \mu}{(\ \mu\ ,\ l := v\ ) \longrightarrow (\ \mu[l \mapsto v]\ ,\ ()\ )}\ \triangleright$
By typing inversion get:
$\bullet;\ \Sigma;\ \bullet \vdash l : \mathsf{ref}\ \tau$
$\bullet;\ \Sigma;\ \bullet \vdash v : \tau$
Thus for $\mu' = \mu[l \mapsto v]$ we have that $\mu' \sim \Sigma$ and $\bullet;\ \Sigma;\bullet \vdash () : \mathsf{unit}$.

Case $\dfrac{l \mapsto v \in \mu}{(\ \mu\ ,\ !l\ ) \longrightarrow (\ \mu\ ,\ v\ )}\ \triangleright$
By typing inversion get: $\bullet;\ \Sigma;\ \bullet \vdash l : \mathsf{ref}\ \tau$
By inversion of $\mu \sim \Sigma$ get:
$\bullet;\ \Sigma;\ \bullet \vdash v : \tau$, which is the desired.

Case $(\ \mu\ ,\ (\Lambda\alpha : k.e)\ \tau''\ ) \longrightarrow (\ \mu\ ,\ e[\tau''/\alpha]\ )\ \triangleright$
By typing inversion we get:
$\bullet;\ \Sigma;\ \bullet \vdash \Lambda\alpha : k.e : \Pi\alpha : k.\tau'$
$\bullet;\ \bullet \vdash \tau'' : k$
$\tau = \tau'[\tau''/\alpha]$
By further typing inversion for $\Lambda\alpha : k.e$ we get:
$\bullet;\ \Sigma;\ \alpha : k \vdash e : \tau'$
Using the lemma C.15 for $[\tau''/\alpha]$ we get:
$\bullet;\ \Sigma;\ \bullet \vdash e[\tau''/\alpha] : \tau'[\tau''/\alpha]$, which is the desired.

Case $(\ \mu\ ,\ \mathsf{fix}\ x : \tau.e\ ) \longrightarrow (\ \mu\ ,\ e[\mathsf{fix}\ x : \tau.e/x]\ )\ \triangleright$
By typing inversion get:
$\bullet;\ \Sigma;\ x : \tau \vdash e : \tau$
By application of the lemma C.15 for $[\mathsf{fix}\ x : \tau.e/x]$ we get:
$\bullet;\ \Sigma;\ \bullet \vdash e[\mathsf{fix}\ x : \tau.e/x] : \tau$

Case $\dfrac{\exists\sigma_\Psi.(\bullet \vdash \sigma_\Psi : \lceil\Psi'\rceil_0 \quad \wedge \quad \lceil T'\rceil_{0,|\Psi'|} \cdot \sigma_\Psi = T)}{(\ \mu\ ,\ \mathsf{unify}\ T\ \mathsf{return}\ (.\tau')\ \mathsf{with}\ (\Psi'.T' \mapsto e')\ ) \longrightarrow (\ \mu\ ,\ \mathsf{inj}_1\ (\lceil e'\rceil_{0,|\Psi'|} \cdot \sigma_\Psi)\ )}\ \triangleright$
By inversion of typing we get:
$\bullet \vdash T : K$
$\bullet,\ K;\ \Sigma;\ \bullet \vdash \lceil\tau'\rceil_{0,1} : \star$
$\bullet \vdash \lceil\Psi'\rceil_{|\Psi|}\ \mathrm{wf}$
$\lceil\Psi'\rceil_0 \vdash \lceil T'\rceil_{0,|\Psi'|} : K$
$\lceil\Psi'\rceil_0;\ \Sigma;\ \bullet \vdash \lceil e'\rceil_{0,|\Psi'|} : \lceil\tau'\rceil_{0,1} \cdot (T')$
$\tau = (\lceil\tau'\rceil_{0,1} \cdot T) + \mathsf{unit}$
By application of lemma C.15 for σ_Ψ and $\lceil e'\rceil_{0,|\Psi'|}$ we get:

$\bullet; \Sigma; \bullet \vdash \lceil e' \rceil_{0,|\Psi'|} \cdot \sigma_\Psi : (\lceil \tau' \rceil_{0,1} \cdot T') \cdot \sigma_\Psi$.
All we now need to prove is that $\lceil \tau' \rceil_{0,1} \cdot T = (\lceil \tau' \rceil_{0,1} \cdot T') \cdot \sigma_\Psi$.
Using the lemma C.11 we get that:
$(\lceil \tau' \rceil_{0,1} \cdot T') \cdot \sigma_\Psi = \lceil \tau' \rceil_{0,1} \cdot (T' \cdot \sigma_\Psi) = \lceil \tau' \rceil_{0,1} \cdot T$
It is now easy to complete the desired result using the typing rule for inj_1.

Case $\dfrac{\nexists \sigma_\Psi . (\bullet \vdash \sigma_\Psi : \lceil \Psi \rceil_0 \quad \wedge \quad \lceil T' \rceil_{0,|\Psi|} \cdot \sigma_\Psi = T)}{(\ \mu\ ,\ \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi.T' \mapsto e')\ ) \longrightarrow (\ \mu\ ,\ \mathsf{inj}_2\ ()\ )}\ \triangleright$
Trivial by application of the typing rule for inj_2.

Lemma C.17 (Canonical forms) *If* $\bullet; \Sigma; \bullet \vdash v : \tau$ *then*

1. *If* $\tau = \Pi(K).\tau'$, *then* $\exists e$ *such that* $v = \Lambda(K).e$.
2. *If* $\tau = \Sigma(K).\tau'$, *then* $\exists T, v'$ *such that* $v = \mathsf{pack}\ T\ \mathsf{return}\ (.\tau'')\ \mathsf{with}\ v'$ *with* $\tau' =_\beta \tau''$.
3. *If* $\tau = \mathsf{unit}$, *then* $v = ()$.
4. *If* $\tau = \tau_1 \to \tau_2$, *then* $\exists e$ *such that* $v = \lambda x : \tau_1 . e$.
5. *If* $\tau = \tau_1 \times \tau_2$, *then* $\exists v_1, v_2$ *such that* $v = (v_1,\ v_2)$.
6. *If* $\tau = \tau_1 + \tau_2$, *then* $\exists v'$ *such that either* $v = \mathsf{inj}_1\ v'$ *or* $v = \mathsf{inj}_2\ v'$.
7. *If* $\tau = (\mu\alpha : k.\tau')\ \tau_1\ \tau_2\ \cdots\ \tau_n$, *then* $\exists v'$ *such that* $v = \mathsf{fold}\ v'$.
8. *If* $\tau = \mathsf{ref}\ \tau'$, *then* $\exists l$ *such that* $v = l$.
9. *If* $\tau = \Lambda\alpha : k.\tau'$, *then* $\exists e$ *such that* $v = \Lambda\alpha : k.e$.

Directly by typing inversion.

Theorem C.18 (Progress) *If* $\bullet; \Sigma; \bullet \vdash e : \tau$ *and* $\mu \sim \Sigma$, *then either* $\mu,\ e \longrightarrow \mathsf{error}$, *or* e *is a value* v, *or there exist* μ' *and* e' *such that* $\mu,\ e \longrightarrow \mu',\ e'$.

We proceed by induction on the typing derivation for e. We do not consider cases where $e = v$ (since the theorem is trivial in that case), or where $e = \mathcal{E}[e']$ with $e \neq v$. In that case, by typing inversion we can get that e' is well-typed under the empty context, so by induction hypothesis we can either prove that $\mu,\ e \longrightarrow \mathsf{error}$, or there exist μ', e'' such that $\mu,\ \mathcal{E}[e] \longrightarrow \mu',\ \mathcal{E}[e'']$ by the environment closure small-step rule. Thus we only consider cases where $e = \mathcal{E}[v]$, or where e cannot be further decomposed into $\mathcal{E}[e']$ with $\mathcal{E} \neq \bullet$. Last, when we don't mention a specific μ', we have that $\mu' = \mu$ with the desired properties obviously holding.

Case $\dfrac{\bullet; \Sigma; \bullet \vdash v : \Pi(K).\tau \qquad \bullet \vdash T : K}{\bullet; \Sigma; \bullet \vdash v\ T : \lceil \tau \rceil_{0,1} \cdot (T)}\ \triangleright$
By use of the canonical forms lemma C.17, we get that $v = \Lambda(K).e$.
By typing inversion we get that $K; \Sigma; \bullet \vdash \lceil e \rceil_{0,1} : \tau'$.
So applying the appropriate operational semantics rule we get an $e' = \lceil e \rceil_{0,1} \cdot T$ such that $(\ \mu\ ,\ e\ ) \longrightarrow (\ \mu\ ,\ e'\ )$.

Case $\dfrac{\bullet; \Sigma; \bullet \vdash v : \Sigma(K).\tau \qquad \bullet,\ K,\ \Sigma; \bullet,\ x : \lceil \tau \rceil_{0,1} \vdash \lceil e' \rceil_{0,1} : \tau' \qquad \bullet; \bullet \vdash \tau' : \star}{\bullet; \Sigma; \bullet \vdash \mathsf{unpack}\ v\ (.)x.(e') : \tau'}\ \triangleright$
By use of the canonical forms lemma C.17, we get that $v = \mathsf{pack}\ T\ \mathsf{return}\ (.\tau'')\ \mathsf{with}\ v'$.
Furthermore we have that $\lceil e' \rceil_{0,1}$ is well-defined, so such will be $\lceil e' \rceil_{0,1} \cdot T$ too.
Thus the relevant operational semantics rule applies.

Case $\dfrac{\bullet;\ \Sigma;\ \bullet \vdash v : \tau \to \tau' \qquad \bullet;\ \Sigma;\ \bullet \vdash e' : \tau}{\bullet;\ \Sigma;\ \bullet \vdash v\ e' : \tau'}\ \triangleright$

From canonical forms, we have that $v = \lambda x : \tau''.e''$, so the relevant step rule applies.

Case $\dfrac{\bullet;\ \Sigma;\ \bullet \vdash e : \tau_1 \times \tau_2 \qquad i = 1 \text{ or } 2}{\bullet;\ \Sigma;\ \bullet \vdash \mathsf{proj}_i\ e : \tau_i}\ \triangleright$

From canonical forms, we have that $v = (v_1,\ v_2)$, so using the relevant step rule for proj_i we get that $(\ \mu\ ,\ \mathsf{proj}_i\ e\ ) \longrightarrow (\ \mu\ ,\ v_i\ )$.

Case $\dfrac{\bullet;\ \Sigma;\ \bullet \vdash v : \tau_1 + \tau_2 \qquad \bullet;\ \Sigma;\ \bullet,\ x : \tau_1 \vdash e_1 : \tau \qquad \bullet;\ \Sigma;\ \bullet,\ x : \tau_2 \vdash e_2 : \tau}{\bullet;\ \Sigma;\ \bullet \vdash \mathsf{case}(v, x.e_1, x.e_2) : \tau}\ \triangleright$

From canonical forms, we have that either $v = \mathsf{inj}_1\ v'$ or $v = \mathsf{inj}_2\ v'$; in each case a step rule applies to give an appropriate e'.

Case $\dfrac{\bullet;\ \bullet \vdash \mu\alpha : k.\tau : k \qquad \bullet;\ \Sigma;\ \bullet \vdash v : (\mu\alpha : k.\tau)\ \tau_1\ \tau_2\ \cdots\ \tau_n}{\bullet;\ \Sigma;\ \bullet \vdash \mathsf{unfold}\ v : \tau[\mu\alpha : k.\tau/\alpha]\ \tau_1\ \tau_2\ \cdots\ \tau_n}\ \triangleright$

From canonical forms, we get that $v = \mathsf{fold}\ v'$, so the relevant step rule trivially applies.

Case $\dfrac{\bullet;\ \Sigma;\ \bullet \vdash v : \tau}{\bullet;\ \Sigma;\ \bullet \vdash \mathsf{ref}\ v : \mathsf{ref}\ \tau}\ \triangleright$

Assuming an infinite heap, we can find a l such that $l \notin \mu$, and construct $\mu' = \mu,\ l \mapsto v$. Thus the relevant step rule applies giving $e' = l$.

Case $\dfrac{\bullet;\ \Sigma;\ \bullet \vdash v : \mathsf{ref}\ \tau}{\bullet;\ \Sigma;\ \bullet \vdash !v : \tau}\ \triangleright$

From canonical forms, we get that $v = l$. By typing inversion, we get that $(l : \tau) \in \Sigma$.
From $\mu \sim \Sigma$, we get that there exists v' such that $(l \mapsto v') \in \mu$.
Thus the relevant step rule applies and gives $e' = v'$.

Case $\dfrac{\bullet;\ \Sigma;\ \bullet \vdash v : \Pi\alpha : k.\tau' \qquad \bullet;\ \bullet \vdash \tau : k}{\bullet;\ \Sigma;\ \bullet \vdash v\ \tau : \tau'[\tau/\alpha]}\ \triangleright$

From canonical forms, we get that $v = \Lambda\alpha : k.e$. The relevant step rule trivially applies to give $e' = e[\tau/\alpha]$.

Case $\dfrac{\bullet;\ \Sigma;\ \bullet,\ x : \tau \vdash e : \tau}{\bullet;\ \Sigma;\ \bullet \vdash \mathsf{fix}\ x : \tau.e : \tau}\ \triangleright$

Trivially we have that the relevant step rule applies giving $e' = e[\mathsf{fix}\ x : \tau.e/x]$.

Case $\dfrac{\begin{array}{c}\bullet \vdash T : K \qquad \bullet,\ K;\ \bullet \vdash \lceil\tau\rceil_{0,1} : \star \\ \bullet \vdash \left\lceil\Psi'\right\rceil_0\ \mathrm{wf} \qquad \bullet,\ \left\lceil\Psi'\right\rceil_0 \vdash \left\lceil T'\right\rceil_{0,|\Psi'|} : K \qquad \bullet,\ \left\lceil\Psi'\right\rceil_0;\ \Sigma;\ \bullet \vdash \left\lceil e'\right\rceil_{0,|\Psi'|} : \lceil\tau\rceil_{0,1} \cdot \left(\left\lceil T'\right\rceil_{0,|\Psi'|}\right)\end{array}}{\bullet;\ \Sigma;\ \bullet \vdash \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi'.T' \mapsto e') : (\lceil\tau\rceil_{0,1} \cdot (T)) + \mathsf{unit}}\ \triangleright$

We have non-determinism here in the semantics, which we will fix in the next section, giving more precise semantics to the patterns and unification procedure. In either case, we split cases on whether an σ_Ψ with the desired properties exists or not, and use the appropriate step rule to get an e' in each case.

D. Typing and unification for patterns

D.1 Adjusting computational language typing

First, we will define two new notions: one is a stricter typing for patterns, allowing only certain forms to be used; the second is relevant typing for patterns, making sure that all declared unification variables are actually used somewhere inside the pattern. Together they are supposed to make sure that unification is possible using a decidable deterministic algorithm; so there is only one unifying substitution, or there is none.

We change the pattern matching typing rule for the computational language as follows:

$$\frac{\begin{array}{c}\Psi \vdash T : K \qquad \Psi,\, K;\, \Gamma \vdash \lceil \tau \rceil_{|\Psi|,1} : \star \qquad \Psi \vdash_p \lceil \Psi' \rceil_{|\Psi|}\ \mathrm{wf} \\ \Psi,\, \lceil \Psi' \rceil_{|\Psi|} \vdash_p \lceil T' \rceil_{|\Psi|,|\Psi'|} : K \qquad \mathsf{relevant}\left(\Psi,\, \lceil \Psi' \rceil_{|\Psi|} \vdash_p \lceil T' \rceil_{|\Psi|,|\Psi'|} : K\right) = \widehat{\Psi},\, \lceil \Psi' \rceil_{|\Psi|} \\ \Psi,\, \lceil \Psi' \rceil_{|\Psi|};\, \Sigma;\, \Gamma \vdash \lceil e' \rceil_{|\Psi|,|\Psi'|} : \lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, \lceil T' \rceil_{|\Psi|,|\Psi'|})\end{array}}{\Psi;\, \Sigma;\, \Gamma \vdash \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi'.T' \mapsto e') : (\lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\, T)) + \mathsf{unit}}$$

Then we define the stricter typing for patterns $\vdash_p$. This will be entirely identical to normal typing, but will disallow forms that would lead to non-determinism (e.g. context unification variables allowed anywhere inside a pattern).

Then we define the notion of relevancy for extension variables. For a judgement $\Psi; \mathcal{J}$, $\mathsf{relevant}\,(\Psi; \mathcal{J}) = \widehat{\Psi}$, where $\widehat{\Psi}$ is a partial context, containing only the extension variables that actually get used. We will show that functions used during typing and evaluation commute with this function.

Then, we prove that either a unique unification exists for a pair of a pattern and a term, yielding a partial substitution for the relevant variables, or that no such unification exists. From this proof we derive an algorithm for unification.

D.2 Strict typing for patterns

Definition D.1 (Pattern typing) *We will adapt the typing rules for extended terms T, to show which of those terms are accepted as valid patterns. We assume that the Ψ is split into two parts, Ψ, Ψ_u, where Ψ_u contains only newly-introduced unification variables just for the purpose of type-checking the current pattern and branch.*

$\boxed{\Psi \vdash_p \Psi_u\ \mathrm{wf}}$

$$\frac{}{\Psi \vdash_p \bullet\ \mathrm{wf}} \qquad \frac{\Psi \vdash_p \Psi_u\ \mathrm{wf} \qquad \Psi,\, \Psi_u \vdash_p [\Phi]t : [\Phi]s}{\Psi \vdash_p (\Psi_u,\, [\Phi]t)\ \mathrm{wf}} \qquad \frac{\Psi \vdash_p \Psi_u\ \mathrm{wf} \qquad \Psi,\, \Psi_u \vdash_p \Phi\ \mathrm{wf}}{\Psi \vdash_p (\Psi_u,\, [\Phi]\,\mathsf{ctx})\ \mathrm{wf}}$$

$\boxed{\Psi,\, \Psi_u \vdash_p T : K}$

$$\frac{\Psi,\, \Psi_u;\, \Phi \vdash_p t : t' \qquad \Psi,\, \Psi_u;\, \Phi \vdash t' : s}{\Psi,\, \Psi_u \vdash_p [\Phi]t : [\Phi]t'} \qquad \frac{\Psi,\, \Psi_u \vdash_p \Phi,\, \Phi'\ \mathrm{wf}}{\Psi,\, \Psi_u \vdash_p [\Phi]\Phi' : [\Phi]\,\mathsf{ctx}}$$

$\boxed{\Psi,\, \Psi_u \vdash_p \Phi\ \mathrm{wf}}$

$$\frac{}{\Psi,\, \Psi_u \vdash_p \bullet\ \mathrm{wf}} \qquad \frac{\Psi,\, \Psi_u \vdash_p \Phi\ \mathrm{wf} \qquad \Psi,\, \Psi_u;\, \Phi \vdash_p t : s}{\Psi,\, \Psi_u \vdash_p (\Phi,\, t)\ \mathrm{wf}}$$

$$\frac{\Psi,\, \Psi_u \vdash_p \Phi\ \mathrm{wf} \qquad (\Psi,\, \Psi_u).i = [\Phi]\,\mathsf{ctx} \qquad i < |\Psi|}{\Psi,\, \Psi_u \vdash_p (\Phi,\, X_i)\ \mathrm{wf}} \qquad \frac{\Psi \vdash_p \Phi\ \mathrm{wf} \qquad (\Psi,\, \Psi_u).i = [\Phi]\,\mathsf{ctx} \qquad i \geq |\Psi|}{\Psi,\, \Psi_u \vdash_p \Phi,\, X_i\ \mathrm{wf}}$$

$\boxed{\Psi,\ \Psi_u;\ \Phi \vdash_p \sigma : \Phi'}$

$$\frac{}{\Psi,\ \Psi_u;\ \Phi \vdash_p \bullet : \bullet} \qquad \frac{\Psi,\ \Psi_u;\ \Phi \vdash_p \sigma : \Phi' \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p t : t' \cdot \sigma}{\Psi,\ \Psi_u;\ \Phi \vdash_p (\sigma,\ t) : (\Phi',\ t')}$$

$$\frac{\Psi,\ \Psi_u;\ \Phi \vdash_p \sigma : \Phi' \qquad (\Psi,\ \Psi_u).i = [\Phi']\,\mathsf{ctx} \qquad \Phi',\ X_i \subseteq \Phi}{\Psi,\ \Psi_u;\ \Phi \vdash_p (\sigma,\ \mathbf{id}(X_i)) : (\Phi',\ X_i)}$$

$\boxed{\Psi,\ \Psi_u;\ \Phi \vdash_p t : t'}$

$$\frac{c : t \in \Sigma}{\Psi,\ \Psi_u;\ \Phi \vdash_p c : t} \qquad \frac{\Phi.\mathbf{I} = t}{\Psi,\ \Psi_u;\ \Phi \vdash_p f_{\mathbf{I}} : t} \qquad \frac{(s,s') \in \mathcal{A}}{\Psi,\ \Psi_u;\ \Phi \vdash_p s : s'}$$

$$\frac{\Psi,\ \Psi_u;\ \Phi \vdash_p t_1 : s \qquad \Psi,\ \Psi_u;\ \Phi,\ t_1 \vdash_p \lceil t_2 \rceil_{|\Phi|} : s' \qquad (s,s',s'') \in \mathcal{R}}{\Psi,\ \Psi_u;\ \Phi \vdash_p \Pi(t_1).t_2 : s''}$$

$$\frac{\Psi,\ \Psi_u;\ \Phi \vdash_p t_1 : s \qquad \Psi,\ \Psi_u;\ \Phi,\ t_1 \vdash_p \lceil t_2 \rceil_{|\Phi|} : t' \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p \Pi(t_1).\ \lfloor t' \rfloor_{|\Phi|,.} : s'}{\Psi,\ \Psi_u;\ \Phi \vdash_p \lambda(t_1).t_2 : \Pi(t_1).\ \lfloor t' \rfloor_{|\Phi|,.}}$$

$$\frac{\Psi,\ \Psi_u;\ \Phi \vdash_p t_1 : \Pi(t).t' \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p t_2 : t}{\Psi,\ \Psi_u;\ \Phi \vdash_p t_1\ t_2 : \lceil t' \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2)}$$

$$\frac{\Psi,\ \Psi_u;\ \Phi \vdash_p t_1 : t \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p t_2 : t \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p t : \mathsf{Type}}{\Psi,\ \Psi_u;\ \Phi \vdash_p t_1 = t_2 : \mathsf{Prop}}$$

$$\frac{(\Psi,\ \Psi_u).i = T \qquad T = [\Phi']t' \qquad i < |\Psi| \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p \sigma : \Phi'}{\Psi,\ \Psi_u;\ \Phi \vdash_p X_i/\sigma : t' \cdot \sigma}$$

$$\frac{(\Psi,\ \Psi_u).i = T \qquad T = [\Phi']t' \qquad i \geq |\Psi| \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p \sigma : \Phi' \qquad \Phi' \subseteq \Phi \qquad \sigma = \mathsf{id}_{\Phi'}}{\Psi,\ \Psi_u;\ \Phi \vdash_p X_i/\sigma : t' \cdot \sigma}$$

$$\frac{\Psi,\ \Psi_u;\ \Phi \vdash_p t : t_1 \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p t_1 : \mathsf{Prop} \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p t' : t_1 = t_2}{\Psi,\ \Psi_u;\ \Phi \vdash_p \mathsf{conv}\ t\ t' : t_2}$$

$$\frac{\Psi,\ \Psi_u;\ \Phi \vdash_p t_1 : t \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p t_1 = t_1 : \mathsf{Prop}}{\Psi,\ \Psi_u;\ \Phi \vdash_p \mathsf{refl}\ t_1 : t_1 = t_1} \qquad \frac{\Psi,\ \Psi_u;\ \Phi \vdash_p t_a : t_1 = t_2}{\Psi,\ \Psi_u;\ \Phi \vdash_p \mathsf{symm}\ t_a : t_2 = t_1}$$

$$\frac{\Psi,\ \Psi_u;\ \Phi \vdash_p t_a : t_1 = t_2 \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p t_b : t_2 = t_3}{\Psi,\ \Psi_u;\ \Phi \vdash_p \mathsf{trans}\ t_a\ t_b : t_1 = t_3}$$

$$\frac{\begin{array}{c}\Psi,\ \Psi_u;\ \Phi \vdash_p t_a : M_1 = M_2 \\ \Psi,\ \Psi_u;\ \Phi \vdash_p M_1 : A \to B \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p t_b : N_1 = N_2 \end{array} \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p N_1 : A}{\Psi,\ \Psi_u;\ \Phi \vdash_p \mathsf{congapp}\ t_a\ t_b : M_1\ N_1 = M_2\ N_2}$$

$$\frac{\begin{array}{c}\Psi,\ \Psi_u;\ \Phi \vdash_p t_a : A_1 = A_2 \\ \Psi,\ \Psi_u;\ \Phi,A_1 \vdash_p \lceil t_b \rceil : B_1 = B_2 \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p A_1 : \mathsf{Prop} \qquad \Psi,\ \Psi_u;\ \Phi,A_1 \vdash_p \lceil B_1 \rceil : \mathsf{Prop} \end{array}}{\Psi,\ \Psi_u;\ \Phi \vdash_p \mathsf{congimpl}\ t_a\ (\lambda(A_1).t_b) : \Pi(A_1).\ \lfloor B_1 \rfloor = \Pi(A_2).\ \lfloor B_2 \rfloor}$$

$$\frac{\Psi,\ \Psi_u;\ \Phi,\ A \vdash_p \lceil t_b \rceil : B = B' \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p \Pi(A).\ \lfloor B \rfloor = \Pi(A).\ \lfloor B' \rfloor : \mathsf{Prop}}{\Psi,\ \Psi_u;\ \Phi \vdash_p \mathsf{congpi}\ (\lambda(A).t_b) : \Pi(A).\ \lfloor B \rfloor = \Pi(A).\ \lfloor B' \rfloor}$$

$$\frac{\Psi,\ \Psi_u;\ \Phi,\ A \vdash_p \lceil t_b \rceil : B_1 = B_2 \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p \lambda(A).\ \lfloor B_1 \rfloor = \lambda(A).\ \lfloor B_2 \rfloor : \mathsf{Prop}}{\Psi,\ \Psi_u;\ \Phi \vdash_p \mathsf{conglam}\ (\lambda(A).t_b) : \lambda(A).\ \lfloor B_1 \rfloor = \lambda(A).\ \lfloor B_2 \rfloor}$$

$$\frac{\Psi,\ \Psi_u;\ \Phi \vdash_p \lambda(A).M : A \to B \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p N : A \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p A \to B : \mathsf{Type}}{\Psi,\ \Psi_u;\ \Phi \vdash_p \mathsf{beta}\ (\lambda(A).M)\ N : (\lambda(A).M)\ N = \lceil M \rceil \cdot (\mathsf{id}_\Phi, N)}$$

Now we need to prove that all theorems that had to do with these typing judgements still hold. In most cases, this holds entirely trivially, since the $\vdash_p$ judgements are exactly the same as the $\vdash$ judgements, with some extra restrictions as side-conditions. The only theorems that we need to reprove are the ones that require special care in exactly those rules that now have side-conditions. As these rules all have to do just with the use of extension variables, we understand that the theorems that we need to adapt are the extension substitution lemmas. Their statements need to be adjusted to account for part of the substitution corresponding to the Ψ part, and part of it corresponding to the Ψ_u part (both in the source and target extension contexts of the substitution). Though we do not provide the details here, the main argument why these continue to hold is the following: we never substitute variables from Ψ_u with anything other than the same variable in a context that includes the same Ψ_u. Thus the side-conditions will continue to hold.

Theorem D.2 (Extension of lemma B.97) *If* Ψ', $\Psi_u \cdot \sigma_\Psi \vdash (\sigma_\Psi, X_{|\Psi'|}, \cdots, X_{|\Psi'|+|\Psi_u|}) : (\Psi,\ \Psi_u)$ *then:*

1. *If* $\Psi,\ \Psi_u;\ \Phi \vdash_p t : t'$ *then* $\Psi',\ \Psi_u \cdot \sigma_\Psi;\ \Phi \cdot \sigma_\Psi \vdash_p t \cdot \sigma_\Psi : t' \cdot \sigma_\Psi$.
2. *If* $\Psi,\ \Psi_u;\ \Phi \vdash_p \sigma : \Phi'$ *then* $\Psi',\ \Psi_u \cdot \sigma_\Psi;\ \Phi \cdot \sigma_\Psi \vdash_p \sigma \cdot \sigma_\Psi : \Phi' \cdot \sigma_\Psi$.
3. *If* $\Psi,\ \Psi_u \vdash_p \Phi$ *wf then* $\Psi',\ \Psi_u \cdot \sigma_\Psi \vdash_p \Phi \cdot \sigma_\Psi$ *wf.*
4. *If* $\Psi,\ \Psi_u \vdash_p T : K$ *then* $\Psi',\ \Psi_u \cdot \sigma_\Psi \vdash_p T \cdot \sigma_\Psi : K \cdot \sigma_\Psi$.

In all cases proceed entirely similarly as before. The only special cases that need to be accounted for are the ones that have to do with restrictions on variables coming out of Ψ_u.

Case $\dfrac{\Psi \vdash_p \Phi\ \mathsf{wf} \qquad (\Psi,\ \Psi_u).i = [\Phi]\,\mathsf{ctx} \qquad i \geq |\Psi|}{\Psi,\ \Psi_u \vdash_p \Phi,\ X_i\ \mathsf{wf}}\ \triangleright$

We need to prove that $\Psi',\ \Psi'_u \vdash_p \Phi \cdot \sigma'_\Psi,\ X_i \cdot \sigma'_\Psi$ wf, where $\sigma'_\Psi = \sigma_\Psi,\ X_{|\Psi'|},\ \cdots,\ X_{|\Psi'|+|\Psi_u|}$. By induction hypothesis for $\Psi_u = \bullet$ we get that $\Psi \vdash_p \Phi \cdot \sigma_\Psi$ wf, and because of lemma B.92 we get that also $\Psi \vdash_p \Phi \cdot \sigma'_\Psi$ wf.
Also, we have that $X_i \cdot \sigma'_\Psi = X_{i-|\Psi|+|\Psi'|}$.
We have that $(\Psi',\ \Psi'_u).i - |\Psi| + |\Psi'| = [\Phi \cdot \sigma'_\Psi]\,\mathsf{ctx}$.
Last, since $i \geq |\Psi|$, we also have that $i - |\Psi| + |\Psi'| \geq |\Psi'|$.
Thus by the use of the same typing rule, we arrive at the desired.

Case $\dfrac{(\Psi,\ \Psi_u).i = T \qquad T = [\Phi']t' \qquad i \geq |\Psi| \qquad \Psi,\ \Psi_u;\ \Phi \vdash_p \sigma : \Phi' \qquad \sigma = \mathsf{id}_{\Phi'}}{\Psi,\ \Psi_u;\ \Phi \vdash_p X_i/\sigma : t' \cdot \sigma}\ \triangleright$

Similarly as above. Furthermore, we need to show that $\mathsf{id}_{\Phi'} \cdot \sigma'_\Psi = \mathsf{id}_{\Phi' \cdot \sigma'_\Psi}$, which is simple to prove by induction on Φ'.

Lemma D.3 (Extension of lemma B.98) *If* $\Psi \vdash_p \Psi_u$ *wf then* $\Psi \vdash_p \Psi_u \cdot \sigma_\Psi$ *wf.*

Similarly to lemma B.98 and use of the above lemma.

D.3 Relevant typing

We will proceed to define a notion of partial contexts: these are extension contexts where certain elements are unspecified. It is presumed that in the judgements that they appear, only specified elements are relevant; the judgements do not depend on the other elements at all (save for them being well-formed). We will use this notion in order to make sure that all unification variables introduced during pattern matching are relevant. Otherwise, the irrelevant variables could be substituted by arbitrary terms, resulting in the existence of an infinite number of valid unification substitutions.

Definition D.4 *The syntax for partial contexts is defined as follows.*

$\widehat{\Psi} ::= \bullet \mid \widehat{\Psi},\, K \mid \widehat{\Psi},\, ?$

Definition D.5 *Well-formedness for partial contexts is defined as follows.*

$\boxed{\vdash \widehat{\Psi}\ \mathrm{wf}}$

$$\frac{}{\vdash \bullet\ \mathrm{wf}} \qquad \frac{\vdash \widehat{\Psi}\ \mathrm{wf} \qquad \widehat{\Psi} \vdash [\Phi]t : [\Phi]s}{\vdash (\widehat{\Psi},\, [\Phi]t)\ \mathrm{wf}} \qquad \frac{\vdash \widehat{\Psi}\ \mathrm{wf} \qquad \widehat{\Psi} \vdash \Phi\ \mathrm{wf}}{\vdash (\widehat{\Psi},\, [\Phi]\,\mathsf{ctx})\ \mathrm{wf}} \qquad \frac{\vdash \widehat{\Psi}\ \mathrm{wf}}{\vdash (\widehat{\Psi},\, ?)\ \mathrm{wf}}$$

Other than the above change in the well-formedness definition, partial contexts are used with entirely the same definitions as before. This means that if a typing judgement like $\widehat{\Psi};\ \Phi \vdash t : t'$ tries to access the i-th metavariable, this metavariable should be specified in $\widehat{\Psi}$ rather than being the unspecified element $?$ – because the side-condition $\widehat{\Psi}.i = K$ would otherwise be violated.

We proceed to define a judgement that extracts the relevant extension variables out of typing judgements that use a concrete context Ψ, yielding a partial context $\widehat{\Psi}$. We first need a couple of definitions.

Definition D.6 *The fully-unspecified partial context is defined as follows.*

$\boxed{\mathsf{unspec}_\Psi}$

$$\begin{array}{lcl} \mathsf{unspec}_\bullet & = & \bullet \\ \mathsf{unspec}_{\Psi,\,K} & = & \mathsf{unspec}_\Psi,\, ? \end{array}$$

Definition D.7 *The partial context specified solely at i is defined as follows.*

$\boxed{\Psi \widehat{@} i}$

$$\begin{array}{lcl} (\Psi,\, K)\,\widehat{@}\, i & = & \mathsf{unspec}_\Psi,\, K \text{ when } |\Psi| = i \\ (\Psi,\, K)\,\widehat{@}\, i & = & (\Psi \widehat{@} i),\, ? \text{ when } |\Psi| > i \end{array}$$

Definition D.8 *Joining two partial contexts is defined as follows.*

$\boxed{\widehat{\Psi} \circ \widehat{\Psi}'}$

$$\begin{array}{lcl} \bullet \circ \bullet & = & \bullet \\ (\widehat{\Psi},\, K) \circ (\widehat{\Psi}',\, K) & = & (\widehat{\Psi} \circ \widehat{\Psi}'),\, K \\ (\widehat{\Psi},\, ?) \circ (\widehat{\Psi}',\, K) & = & (\widehat{\Psi} \circ \widehat{\Psi}'),\, K \\ (\widehat{\Psi},\, K) \circ (\widehat{\Psi}',\, ?) & = & (\widehat{\Psi} \circ \widehat{\Psi}'),\, K \\ (\widehat{\Psi},\, ?) \circ (\widehat{\Psi}',\, ?) & = & (\widehat{\Psi} \circ \widehat{\Psi}'),\, ? \end{array}$$

Definition D.9 *The notion of one partial context being a less precise version of another one is defined as follows.*

$\boxed{\widehat{\Psi} \sqsubseteq \widehat{\Psi}'}$

$$\begin{array}{l} \bullet \sqsubseteq \bullet \\ (\widehat{\Psi},\, K) \sqsubseteq (\widehat{\Psi}',\, K) \quad \Leftarrow \quad \widehat{\Psi} \sqsubseteq \widehat{\Psi}' \\ (\widehat{\Psi},\, ?) \sqsubseteq (\widehat{\Psi}',\, K) \quad \Leftarrow \quad \widehat{\Psi} \sqsubseteq \widehat{\Psi}' \\ (\widehat{\Psi},\, ?) \sqsubseteq (\widehat{\Psi}',\, ?) \quad \Leftarrow \quad \widehat{\Psi} \sqsubseteq \widehat{\Psi}' \end{array}$$

Definition D.10 *We define a judgement to extract the relevant extension variables out of a context.*

$$\boxed{\mathsf{relevant}\,(\Psi \vdash T : K) = \widehat{\Psi}}$$

$$\mathsf{relevant}\left(\frac{\Psi;\ \Phi \vdash t : t' \qquad \Psi;\ \Phi \vdash t' : s}{\Psi \vdash [\Phi]t : [\Phi]t'}\right) = \mathsf{relevant}\left(\Psi;\ \Phi \vdash t : t'\right)$$

$$\mathsf{relevant}\left(\frac{\Psi \vdash \Phi,\ \Phi'\ \mathrm{wf}}{\Psi \vdash [\Phi]\Phi' : [\Phi]\,\mathsf{ctx}}\right) = \mathsf{relevant}\left(\Psi \vdash \Phi,\ \Phi'\ \mathrm{wf}\right)$$

$$\boxed{\mathsf{relevant}\,(\Psi \vdash \Phi\ \mathrm{wf}) = \widehat{\Psi}}$$

$$\mathsf{relevant}\left(\frac{}{\Psi \vdash \bullet\ \mathrm{wf}}\right) = \mathsf{unspec}_{\Psi} \qquad \mathsf{relevant}\left(\frac{\Psi \vdash \Phi\ \mathrm{wf} \qquad \Psi;\ \Phi \vdash t : s}{\Psi \vdash (\Phi,\ t)\ \mathrm{wf}}\right) = \mathsf{relevant}\,(\Psi;\ \Phi \vdash t : s)$$

$$\mathsf{relevant}\left(\frac{\Psi \vdash \Phi\ \mathrm{wf} \qquad (\Psi).i = [\Phi]\,\mathsf{ctx}}{\Psi \vdash (\Phi,\ X_i)\ \mathrm{wf}}\right) = \mathsf{relevant}\,(\Psi \vdash \Phi\ \mathrm{wf}) \circ (\Psi \widehat{@}\, i)$$

$$\boxed{\mathsf{relevant}\,(\Psi;\ \Phi \vdash t : t') = \widehat{\Psi}}$$

$$\mathsf{relevant}\left(\frac{c : t \in \Sigma}{\Psi;\ \Phi \vdash c : t}\right) = \mathsf{relevant}\,(\Psi \vdash \Phi\ \mathrm{wf}) \qquad \mathsf{relevant}\left(\frac{\Phi.\mathbf{I} = t}{\Psi;\ \Phi \vdash f_{\mathbf{I}} : t}\right) = \mathsf{relevant}\,(\Psi \vdash \Phi\ \mathrm{wf})$$

$$\mathsf{relevant}\left(\frac{(s,s') \in \mathcal{A}}{\Psi;\ \Phi \vdash s : s'}\right) = \mathsf{relevant}\,(\Psi \vdash \Phi\ \mathrm{wf})$$

$$\mathsf{relevant}\left(\frac{\Psi;\ \Phi \vdash t_1 : s \qquad \Psi;\ \Phi,\ t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : s' \qquad (s,s',s'') \in \mathcal{R}}{\Psi;\ \Phi \vdash \Pi(t_1).t_2 : s''}\right) = \mathsf{relevant}\left(\Psi;\ \Phi,\ t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : s'\right)$$

$$\mathsf{relevant}\left(\frac{\Psi;\ \Phi \vdash t_1 : s \qquad \Psi;\ \Phi,\ t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : t' \qquad \Psi;\ \Phi \vdash \Pi(t_1).\ \lfloor t' \rfloor_{|\Phi|,\,.} : s'}{\Psi;\ \Phi \vdash \lambda(t_1).t_2 : \Pi(t_1).\ \lfloor t' \rfloor_{|\Phi|,\,.}}\right) =$$

$$\mathsf{relevant}\left(\Psi;\ \Phi,\ t_1 \vdash \lceil t_2 \rceil_{|\Phi|} : t'\right)$$

$$\mathsf{relevant}\left(\frac{\Psi;\ \Phi \vdash t_1 : \Pi(t).t' \qquad \Psi;\ \Phi \vdash t_2 : t}{\Psi;\ \Phi \vdash t_1\ t_2 : \lceil t' \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2)}\right) = \mathsf{relevant}\left(\Psi;\ \Phi \vdash t_1 : \Pi(t).t'\right) \circ \mathsf{relevant}\,(\Psi;\ \Phi \vdash t_2 : t)$$

$$\mathsf{relevant}\left(\frac{\Psi;\ \Phi \vdash t_1 : t \qquad \Psi;\ \Phi \vdash t_2 : t \qquad \Psi;\ \Phi \vdash t : \mathsf{Type}}{\Psi;\ \Phi \vdash t_1 = t_2 : \mathsf{Prop}}\right) =$$

$$\mathsf{relevant}\,(\Psi;\ \Phi \vdash t_1 : t) \circ \mathsf{relevant}\,(\Psi;\ \Phi \vdash t_2 : t) \qquad \mathsf{relevant}\left(\frac{(\Psi).i = T \qquad T = [\Phi']t' \qquad \Psi;\ \Phi \vdash \sigma : \Phi'}{\Psi;\ \Phi \vdash X_i/\sigma : t' \cdot \sigma}\right) =$$

$$(\mathsf{relevant}\left(\Psi\lfloor_i \vdash [\Phi']t' : [\Phi']s\right), \overbrace{?, ?, \cdots, ?}^{|\Psi|-i\ \text{times}}) \circ \mathsf{relevant}\left(\Psi;\ \Phi \vdash \sigma : \Phi'\right) \circ (\Psi \widehat{@} i)$$

$$\mathsf{relevant}\left(\frac{\Psi;\ \Phi \vdash t : t_1 \qquad \Psi;\ \Phi \vdash t_1 : \mathsf{Prop} \qquad \Psi;\ \Phi \vdash t' : t_1 = t_2}{\Psi;\ \Phi \vdash \mathsf{conv}\ t\ t' : t_2}\right) =$$

$$\mathsf{relevant}\,(\Psi;\ \Phi \vdash t : t_1) \circ \mathsf{relevant}\left(\Psi;\ \Phi \vdash t' : t_1 = t_2\right)$$

$$\mathsf{relevant}\left(\frac{\Psi;\ \Phi \vdash t_1 : t \qquad \Psi;\ \Phi \vdash t_1 = t_1 : \mathsf{Prop}}{\Psi;\ \Phi \vdash \mathsf{refl}\ t_1 : t_1 = t_1}\right) = \mathsf{relevant}\,(\Psi;\ \Phi \vdash t_1 : t)$$

$$\mathsf{relevant}\left(\frac{\Psi;\ \Phi \vdash t_a : t_1 = t_2}{\Psi;\ \Phi \vdash \mathsf{symm}\ t_a : t_2 = t_1}\right) = \mathsf{relevant}\,(\Psi;\ \Phi \vdash t_a : t_1 = t_2)$$

$$\mathsf{relevant}\left(\frac{\Psi;\ \Phi \vdash t_a : t_1 = t_2 \qquad \Psi;\ \Phi \vdash t_b : t_2 = t_3}{\Psi;\ \Phi \vdash \mathsf{trans}\ t_a\ t_b : t_1 = t_3}\right) =$$

$$\mathsf{relevant}\,(\Psi;\ \Phi \vdash t_a : t_1 = t_2) \circ \mathsf{relevant}\,(\Psi;\ \Phi \vdash t_b : t_2 = t_3)$$

$$\mathsf{relevant}\left(\frac{\Psi;\ \Phi \vdash t_a : M_1 = M_2 \qquad \Psi;\ \Phi \vdash M_1 : A \to B \qquad \Psi;\ \Phi \vdash t_b : N_1 = N_2 \qquad \Psi;\ \Phi \vdash N_1 : A}{\Psi;\ \Phi \vdash \mathsf{congapp}\ t_a\ t_b : M_1\ N_1 = M_2\ N_2}\right) =$$

$$\mathsf{relevant}\,(\Psi;\ \Phi \vdash t_a : M_1 = M_2) \circ \mathsf{relevant}\,(\Psi;\ \Phi \vdash t_b : N_1 = N_2)$$

$$\mathsf{relevant}\left(\frac{\Psi;\,\Phi\vdash t_a : A_1 = A_2 \qquad \Psi;\,\Phi, A_1 \vdash \lceil t_b\rceil : B_1 = B_2 \qquad \Psi;\,\Phi\vdash A_1 : \mathsf{Prop} \qquad \Psi;\,\Phi, A_1 \vdash \lceil B_1\rceil : \mathsf{Prop}}{\Psi;\,\Phi\vdash \mathsf{congimpl}\; t_a\; (\lambda(A_1).t_b) : \Pi(A_1).\lfloor B_1\rfloor = \Pi(A_2).\lfloor B_2\rfloor}\right) =$$

$$\mathsf{relevant}\,(\Psi;\,\Phi\vdash t_a : A_1 = A_2) \circ \mathsf{relevant}\,(\Psi;\,\Phi, A_1 \vdash \lceil t_b\rceil : B_1 = B_2)$$

$$\mathsf{relevant}\left(\frac{\Psi;\,\Phi,\, A\vdash \lceil t_b\rceil : B = B' \qquad \Psi;\,\Phi\vdash \Pi(A).\lfloor B\rfloor = \Pi(A).\lfloor B'\rfloor : \mathsf{Prop}}{\Psi;\,\Phi\vdash \mathsf{congpi}\;(\lambda(A).t_b) : \Pi(A).\lfloor B\rfloor = \Pi(A).\lfloor B'\rfloor}\right) =$$

$$\mathsf{relevant}\left(\Psi;\,\Phi,\, A\vdash \lceil t_b\rceil : B = B'\right)$$

$$\mathsf{relevant}\left(\frac{\Psi;\,\Phi,\, A\vdash \lceil t_b\rceil : B_1 = B_2 \qquad \Psi;\,\Phi\vdash \lambda(A).\lfloor B_1\rfloor = \lambda(A).\lfloor B_2\rfloor : \mathsf{Prop}}{\Psi;\,\Phi\vdash \mathsf{conglam}\;(\lambda(A).t_b) : \lambda(A).\lfloor B_1\rfloor = \lambda(A).\lfloor B_2\rfloor}\right) =$$

$$\mathsf{relevant}\,(\Psi;\,\Phi,\, A\vdash \lceil t_b\rceil : B_1 = B_2)$$

$$\mathsf{relevant}\left(\frac{\Psi;\,\Phi\vdash \lambda(A).M : A\to B \qquad \Psi;\,\Phi\vdash N : A \qquad \Psi;\,\Phi\vdash A\to B : \mathsf{Type}}{\Psi;\,\Phi\vdash \mathsf{beta}\;(\lambda(A).M)\;N : (\lambda(A).M)\;N = \lceil M\rceil\cdot(\mathsf{id}_\Phi, N)}\right) =$$

$$\mathsf{relevant}\,(\Psi;\,\Phi\vdash \lambda(A).M : A\to B) \circ \mathsf{relevant}\,(\Psi;\,\Phi\vdash N : A)$$

$\boxed{\Psi;\,\Phi\vdash\sigma : \Phi'}$

$$\mathsf{relevant}\left(\frac{}{\Psi;\,\Phi\vdash\bullet:\bullet}\right) = \mathsf{relevant}\,(\Psi\vdash\Phi\;\mathrm{wf})$$

$$\mathsf{relevant}\left(\frac{\Psi;\,\Phi\vdash\sigma:\Phi' \qquad \Psi;\,\Phi\vdash t : t'\cdot\sigma}{\Psi;\,\Phi\vdash(\sigma,\,t):(\Phi',\,t')}\right) = \mathsf{relevant}\left(\Psi;\,\Phi\vdash\sigma:\Phi'\right)\circ\mathsf{relevant}\left(\Psi;\,\Phi\vdash t : t'\cdot\sigma\right)$$

$$\mathsf{relevant}\left(\frac{\Psi;\,\Phi\vdash\sigma:\Phi' \qquad (\Psi).i = [\Phi']\,\mathsf{ctx} \qquad \Phi',\, X_i\subseteq\Phi}{\Psi;\,\Phi\vdash(\sigma,\,\mathbf{id}(X_i)):(\Phi',\,X_i)}\right) = \mathsf{relevant}\left(\Psi;\,\Phi\vdash\sigma:\Phi'\right)$$

Lemma D.11 (More-informed contexts preserve judgements) *Assuming* $\widehat{\Psi}\sqsubseteq\widehat{\Psi}'$*:*

1. *If* $\widehat{\Psi}\vdash T : K$ *then* $\widehat{\Psi}'\vdash T : K$.
2. *If* $\widehat{\Psi}\vdash\Phi$ *wf then* $\widehat{\Psi}'\vdash\Phi$ *wf.*
3. *If* $\widehat{\Psi};\,\Phi\vdash t : t'$ *then* $\widehat{\Psi}';\,\Phi\vdash t : t'$.
4. *If* $\widehat{\Psi};\,\Phi\vdash\sigma:\Phi'$ *then* $\widehat{\Psi}';\,\Phi\vdash\sigma:\Phi'$.

Simple by structural induction on the judgements. The interesting cases are the ones mentioning extension variables, as for example when $\Phi = \Phi', X_i$, or $t = X_i/\sigma$. In both such cases, the typing rule has a side condition requiring that $\widehat{\Psi}.i = T$. Since $\widehat{\Psi}\sqsubseteq\widehat{\Psi}'$, we have that $\widehat{\Psi}'.i = T$.

Lemma D.12 (Relevancy is decidable) *1. If* $\Psi\vdash T : K$*, then there exists a unique* $\widehat{\Psi}$ *such that* relevant$(\Psi\vdash T : K) = \widehat{\Psi}$.

2. *If* $\Psi\vdash\Phi$ *wf, then there exists a unique* $\widehat{\Psi}$ *such that* relevant$(\Psi\vdash\Phi\ wf) = \widehat{\Psi}$.
3. *If* $\Psi;\,\Phi\vdash t : t'$*, then there exists a unique* $\widehat{\Psi}$ *such that* relevant$(\Psi;\,\Phi\vdash t : t') = \widehat{\Psi}$.

4. If $\Psi;\ \Phi \vdash \sigma : \Phi'$*, then there exists a unique* $\widehat{\Psi}$ *such that* $\mathsf{relevant}(\Psi;\ \Phi \vdash \sigma : \Phi') = \widehat{\Psi}$.

The relevancy judgements are defined by structural induction on the corresponding typing derivations. It is crucial to take into account the fact that $\vdash \Psi$ wf and $\Psi \vdash \Phi$ wf are implicitly present along any typing derivation that mentions such contexts; thus these derivations themselves, as well as their sub-derivations, are structurally included in derivations like $\Psi;\ \Phi \vdash t : t'$. Furthermore, it is easy to see that all the joins used are defined, since in most cases two results of the relevancy procedure on a judgement using the same context Ψ are joined, which is always well-defined. The only case where this does not hold (use of extension variables in terms), the joins are still defined because of the adaptation of the resulting $\widehat{\Psi}$ by affixing the unspecified elements.

Lemma D.13 (Properties of context join) *1.* $\widehat{\Psi}_1 \circ \widehat{\Psi}_2 \sqsubseteq \widehat{\Psi}_1$

2. $\widehat{\Psi}_1 \circ \widehat{\Psi}_2 \sqsubseteq \widehat{\Psi}_2$

3. $\widehat{\Psi}_1 \circ \widehat{\Psi}_2 = \widehat{\Psi}_2 \circ \widehat{\Psi}_1$

4. $(\widehat{\Psi}_1 \circ \widehat{\Psi}_2) \circ \widehat{\Psi}_3 = \widehat{\Psi}_1 \circ (\widehat{\Psi}_2 \circ \widehat{\Psi}_3)$

5. If $\widehat{\Psi}_1 \sqsubseteq \widehat{\Psi}_2$ *then* $\widehat{\Psi}_1 \circ \widehat{\Psi}_2 = \widehat{\Psi}_2$

6. If $\widehat{\Psi}_1 \sqsubseteq \widehat{\Psi}'_1$ *then* $\widehat{\Psi}_1 \circ \widehat{\Psi}_2 \sqsubseteq \widehat{\Psi}'_1 \circ \widehat{\Psi}_2$

All are simple to prove by induction.

Lemma D.14 (Relevancy when weakening the extensions context) *1. If* $\Psi \vdash T : K$*, then* $\mathsf{relevant}(\Psi,\ \Psi' \vdash T : K) = \mathsf{relevant}(\Psi \vdash T : K), \overbrace{?, \cdots, ?}^{|\Psi'|}$.

2. If $\Psi \vdash \Phi$ *wf, then* $\mathsf{relevant}(\Psi,\ \Psi' \vdash \Phi\ \mathit{wf}) = \mathsf{relevant}(\Psi \vdash \Phi\ \mathit{wf}), \overbrace{?, \cdots, ?}^{|\Psi'|}$.

3. If $\Psi;\ \Phi \vdash t : t'$*, then* $\mathsf{relevant}(\Psi,\ \Psi';\ \Phi \vdash t : t') = \mathsf{relevant}(\Psi;\ \Phi \vdash t : t'), \overbrace{?, \cdots, ?}^{|\Psi'|}$.

4. If $\Psi;\ \Phi \vdash \sigma : \Phi'$*, then* $\mathsf{relevant}(\Psi,\ \Psi';\ \Phi \vdash \sigma : \Phi') = \mathsf{relevant}(\Psi;\ \Phi \vdash \sigma : \Phi'), \overbrace{?, \cdots, ?}^{|\Psi'|}$.

Simple to prove by induction.

Lemma D.15 (Relevancy of sub-judgements is implied) *1.(a)* $\mathsf{relevant}(\Psi \vdash \Phi\ \mathit{wf}) \sqsubseteq \mathsf{relevant}(\Psi \vdash \Phi,\ \Phi'\ \mathit{wf})$

(b) $\mathsf{relevant}(\Psi \vdash \Phi\ \mathit{wf}) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi \vdash t : t')$

(c) $\mathsf{relevant}(\Psi \vdash \Phi\ \mathit{wf}) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi \vdash \sigma : \Phi')$.

2.(a) If $\Psi;\ \Phi \vdash t : t'$ *then* $\mathsf{relevant}(\Psi;\ \Phi,\ \Phi' \vdash t : t') = \mathsf{relevant}(\Psi;\ \Phi \vdash t : t') \circ \mathsf{relevant}(\Psi \vdash \Phi,\ \Phi'\ \mathit{wf})$.

(b) If $\Psi;\ \Phi \vdash \sigma : \Phi'$ *then* $\mathsf{relevant}(\Psi;\ \Phi,\ \Phi'' \vdash \sigma : \Phi') = \mathsf{relevant}(\Psi;\ \Phi \vdash \sigma : \Phi') \circ \mathsf{relevant}(\Psi \vdash \Phi,\ \Phi''\ \mathit{wf})$.

3.(a) If $\Psi;\ \Phi \vdash t : t'$ *and* $\Psi;\ \Phi \vdash t' : s$ *then* $\mathsf{relevant}(\Psi;\ \Phi \vdash t' : s) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi \vdash t : t')$.

(b) If $\Psi;\ \Phi' \vdash \sigma : \Phi$ *then* $\mathsf{relevant}(\Psi \vdash \Phi\ \mathit{wf}) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma : \Phi)$.

4.(a) If $\Psi;\ \Phi \vdash t : t'$ *and* $\Psi;\ \Phi' \vdash \sigma : \Phi$*, then* $\mathsf{relevant}(\Psi;\ \Phi' \vdash t \cdot \sigma : t' \cdot \sigma) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi \vdash t : t') \circ \mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma : \Phi)$.

(b) If $\Psi;\ \Phi' \vdash \sigma : \Phi$ *and* $\Psi;\ \Phi'' \vdash \sigma' : \Phi'$*, then* $\mathsf{relevant}(\Psi;\ \Phi'' \vdash \sigma \cdot \sigma' : \Phi) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma : \Phi) \circ \mathsf{relevant}(\Psi;\ \Phi'' \vdash \sigma' : \Phi')$.

Part 1(a) Trivial by induction the derivation of relevancy.

Part 1(b) By inversion of the derivation of $\mathsf{relevant}(\Psi \vdash \Phi\ \mathrm{wf}) = \widehat{\Psi}$. In the base cases, this is directly proved by the relevancy judgement; in the case where we have $\mathsf{relevant}(\Psi;\ \Phi \vdash t : t') = \mathsf{relevant}(\Psi,\ \Phi,\ t_1 \vdash t_2 : t_3)$, by induction hypothesis get that $\mathsf{relevant}(\Psi \vdash \Phi,\ t_1\ \mathrm{wf})$, which by inversion gives us the desired; in the metavariables case trivially follows from repeated inversions of $\mathsf{relevant}(\Psi;\ \Phi \vdash \sigma : \Phi')$.

Part 1(c) Trivial by induction and use of part 1(b).

Part 2 By induction on the typing derivations of t and t' all cases follow trivially.

Part 3(a) By induction on the derivation of $\Psi;\ \Phi \vdash t : t'$.

Case $t = c\ \triangleright$ Simply using the above parts and the fact that $\Psi;\ \bullet \vdash t' : s$, we have that $\mathsf{relevant}(\Psi;\ \Phi \vdash t' : s) = \mathsf{unspec}_\Psi \circ \mathsf{relevant}(\Psi \vdash \Phi\ \mathrm{wf}) = \mathsf{relevant}(\Psi \vdash \Phi\ \mathrm{wf}) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi \vdash t : t')$.

Case $t = s\ \triangleright$ Similarly as the above case.

Case $t = v_{\mathbf{I}}\ \triangleright$ We have that $\Psi;\ \Phi|_{\mathbf{I}} \vdash \Phi.\mathbf{I} : s$, by inversion of the well-formedness derivation for Φ. Therefore $\mathsf{relevant}(\Psi;\ \Phi \vdash t' : s) = \mathsf{relevant}(\Psi;\ \Phi|_{\mathbf{I}} \vdash t' : s) \circ \mathsf{relevant}(\Psi \vdash \Phi\ \mathrm{wf})$. By repeated inversion of $\Psi \vdash \Phi$ wf we get that $\mathsf{relevant}(\Psi \vdash (\Phi|_{\mathbf{I}},\ \Phi.\mathbf{I})\ \mathrm{wf}) \sqsubseteq \mathsf{relevant}(\Psi \vdash \Phi\ \mathrm{wf})$. Thus we have that $\mathsf{relevant}(\Psi;\ \Phi \vdash t' : s) \sqsubseteq \mathsf{relevant}(\Psi \vdash \Phi\ \mathrm{wf})$, which proves the desired.

Case $t = \Pi(t_1).t_2\ \triangleright$ Trivially from the fact that $\mathsf{relevant}(\Psi;\ \Phi \vdash s'' : s''') = \mathsf{relevant}(\Psi \vdash \Phi\ \mathrm{wf}) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi, t_1 \vdash \lceil t_2 \rceil)$

Case $t = \lambda(t_1).t_2\ \triangleright$ We have that $\mathsf{relevant}(\Psi;\ \Phi \vdash \Pi(t_1).\lfloor t' \rfloor : s') = \mathsf{relevant}(\Psi;\ \Phi,\ t_1 \vdash \lceil \lfloor t' \rfloor \rceil : s) = \mathsf{relevant}(\Psi;\ \Phi,\ t_1 \vdash t' : s)$. So by induction hypothesis, since $\Psi;\ \Phi,\ t_1 \vdash t' : s$ is a sub-derivation in $\Psi;\ \Phi,\ t_1 \vdash \lceil t_2 \rceil : t'$, we have that $\mathsf{relevant}(\Psi;\ \Phi,\ t_1 \vdash t' : s) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi,\ t_1 \vdash t_2 : t')$, which is the desired.

Case $t = t_1\ t_2\ \triangleright$ By induction hypothesis get that $\mathsf{relevant}(\Psi;\ \Phi \vdash \Pi(t).t' : s) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi \vdash t_1 : \Pi(t).t')$. (*Here we assume unique typing for* Π *types*). Furthermore, we have that $\mathsf{relevant}(\Psi;\ \Phi \vdash \Pi(t).t' : s) = \mathsf{relevant}(\Psi;\ \Phi,\ t \vdash \lceil t' \rceil : s)$. Otherwise, it is simple to prove that $\mathsf{relevant}(\Psi;\ \Phi \vdash (\mathsf{id}_\Phi,\ t_2) : (\Phi,\ \lceil t' \rceil)) = \mathsf{relevant}(\Psi;\ \Phi \vdash t_2 : \lceil t' \rceil)$, thus the desired follows trivially following part 4.

Case $t = X_i/\sigma\ \triangleright$ We have that $\mathsf{relevant}(\Psi;\ \Phi' \vdash t' : s) = \mathsf{relevant}(\Psi\lfloor_i;\ \Phi' \vdash t' : s),\ \overbrace{?, \cdots, ?}^{|\Psi| - i \text{ times}}$ from inversion of well-formedness for Ψ. Furthermore, we have that $\Psi;\ \Phi \vdash \sigma : \Phi'$ from typing inversion. Thus, using part 4, we get that $\mathsf{relevant}(\Psi;\ \Phi \vdash t' \cdot \sigma : s) \sqsubseteq (\mathsf{relevant}(\Psi\lfloor_i;\ \Phi' \vdash t' : s),\ \overbrace{?, \cdots, ?}^{|\Psi| - i \text{ times}}) \circ \mathsf{relevant}(\Psi;\ \Phi \vdash \sigma : \Phi')$. Thus the result follows directly, taking the properties of join into account.

Case $t = (t_1 = t_2)\ \triangleright$ Trivial.

Case $t = \mathsf{conv}\ t\ t'\ \triangleright$ By induction hypothesis we get that $\mathsf{relevant}(\Psi;\ \Phi \vdash t_1 = t_2 : \mathsf{Prop}) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi \vdash t' : t_1 = t_2)$. By inversion of relevancy for $t_1 = t_2$ we get that it is equal to $\mathsf{relevant}(\Psi;\ \Phi \vdash t_1 : \mathsf{Prop}) \circ \mathsf{relevant}(\Psi;\ \Phi \vdash t_2 : \mathsf{Prop}) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi \vdash t_2 : \mathsf{Prop})$. Thus the desired follows trivially using the properties of joining contexts.

Case (rest) $\triangleright$ Following the techniques used above.

Part 3(b) By induction on the derivation of $\Psi;\ \Phi' \vdash \sigma : \Phi$.

Case $\sigma = \bullet$ $\triangleright$ Trivial.

Case $\sigma = \sigma', t'$ $\triangleright$ By induction hypothesis for σ', use of part 3(a) for t', and definition of relevancy for Φ.

Case $\sigma = \sigma', \mathbf{id}(X_i)$ $\triangleright$ By induction hypothesis for σ', and also using the side condition for X_i being part of Φ': by inversion of well-formedness for Φ', we get that $\Psi\widehat{@}i \sqsubseteq \mathsf{relevant}(\Psi \vdash \Phi' \text{ wf})$ and thus also $\Psi\widehat{@}i \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma : \Phi)$, proving the desired.

Part 4(a) By induction on the typing derivation for t.

Case $t = c$ $\triangleright$ We have that $\mathsf{relevant}(\Psi;\ \Phi \vdash c : t') = \mathsf{relevant}(\Psi \vdash \Phi \text{ wf})$, and $\mathsf{relevant}(\Psi;\ \Phi' \vdash c \cdot \sigma : t' \cdot \sigma) = \mathsf{relevant}(\Psi \vdash \Phi' \text{ wf})$. We need to show that $\mathsf{relevant}(\Psi \vdash \Phi \text{ wf}) \sqsubseteq \mathsf{relevant}(\Psi \vdash \Phi' \text{ wf}) \circ \mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma : \Phi)$. We have that $\mathsf{relevant}(\Psi \vdash \Phi' \text{ wf}) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma : \Phi)$, so the join in the above equality is well-defined; from the properties of join it is evident that it is enough to show $\mathsf{relevant}(\Psi \vdash \Phi \text{ wf}) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma : \Phi)$. This is trivially proved by part 3(b).

Case $t = s$ $\triangleright$ Similarly.

Case $t = f_{\mathbf{I}}$ $\triangleright$ We have that $\mathsf{relevant}(\Psi;\ \Phi' \vdash f_{\mathbf{I}} \cdot \sigma : t' \cdot \sigma) = \mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma.i : t' \cdot \sigma)$. By inversion for σ, we have that $\mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma.i : \Phi.i \cdot \sigma) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma : \Phi)$. Thus the desired directly follows.

Case $t = \Pi(t_1).t_2$ $\triangleright$ By induction hypothesis for $\lceil t_2 \rceil$ and $\sigma = \sigma,\ f_{|\Phi|}$, we get that:
$\mathsf{relevant}\big(\Psi;\ \Phi', t_1 \cdot \sigma \vdash \lceil t_2 \rceil \cdot (\sigma,\ f_{|\Phi|}) : s''\big) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi',\ t_1 \cdot \sigma \vdash \lceil t_2 \rceil : s'') \circ \mathsf{relevant}\big(\Psi;\ \Phi',\ t_1 \cdot \sigma \vdash (\sigma,\ f_{|\Phi|}) : (\Phi,\ t_1)\big)$.
Also we have that $\mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma : \Phi) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi',\ t_1 \cdot \sigma \vdash \sigma : \Phi)$. Using the known properties of freshening and substitutions, we know that $\mathsf{relevant}(\Psi;\ \Phi' \vdash t \cdot \sigma : s'') = \mathsf{relevant}\big(\Psi;\ \Phi', t_1 \cdot \sigma \vdash \lceil t_2 \rceil \cdot (\sigma,\ f_{|\Phi|}) : s''\big)$, thus this is the desired.

Case $t = \lambda(t_1).t_2$ $\triangleright$ Similar to the above.

Case $t = t_1\ t_2$ $\triangleright$ By induction hypothesis we get that:
$\mathsf{relevant}(\Psi;\ \Phi' \vdash t_1 \cdot \sigma : \Pi(t \cdot \sigma).(t' \cdot \sigma)) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi \vdash t_1 : \Pi(t).t') \circ \mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma : \Phi)$, and that
$\mathsf{relevant}(\Psi;\ \Phi' \vdash t_2 \cdot \sigma : t \cdot \sigma) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi \vdash t_2 : t) \circ \mathsf{relevant}(\Psi;\ \Phi' \vdash \sigma : \Phi)$. Furthermore, we have that
$\mathsf{relevant}(\Psi;\ \Phi' \vdash t_1 \cdot \sigma\ t_2 \cdot \sigma : \lceil t' \cdot \sigma \rceil \cdot (\mathsf{id}_{\Phi'},\ t_2 \cdot \sigma))$
$= \mathsf{relevant}(\Psi;\ \Phi' \vdash t_1 \cdot \sigma : \Pi(t \cdot \sigma).(t' \cdot \sigma)) \circ \mathsf{relevant}(\Psi;\ \Phi' \vdash t_2 \cdot \sigma : t \cdot \sigma)$. The desired follows trivially, using the properties of join.

Case $t = X_i/\sigma'$ $\triangleright$ Trivial, using part 4(b).

Case (rest) $\triangleright$ Using similar techniques as above.

Part 4(b) By induction and use of part 4(a).

Lemma D.16 (Relevancy soundness) *1. If* $\Psi \vdash T : K$ *and* $\mathsf{relevant}(\Psi \vdash T : K) = \widehat{\Psi}$ *then* $\widehat{\Psi} \vdash T : K$.

2. If $\Psi \vdash \Phi$ *wf and* $\mathsf{relevant}(\Psi \vdash \Phi \text{ wf}) = \widehat{\Psi}$ *then* $\widehat{\Psi} \vdash \Phi$ *wf.*

3. If $\Psi;\ \Phi \vdash t : t'$ *and* $\mathsf{relevant}(\Psi;\ \Phi \vdash t : t') = \widehat{\Psi}$ *then* $\widehat{\Psi};\ \Phi \vdash t : t'$.

4. If $\Psi;\ \Phi \vdash \sigma : \Phi'$ *and* $\mathsf{relevant}(\Psi;\ \Phi \vdash \sigma : \Phi') = \widehat{\Psi}$ *then* $\widehat{\Psi};\ \Phi \vdash \sigma : \Phi'$.

Part 1 By induction on the derivation of $\Psi \vdash T : K$.

Case $T = [\Phi]t$ $\triangleright$ By part 3 we have that if $\mathsf{relevant}(\Psi;\ \Phi \vdash t : t') = \widehat{\Psi}$, then $\widehat{\Psi};\ \Phi \vdash t : t'$. From this we also get that $\widehat{\Psi};\ \Phi \vdash t' : s$, and thus it is trivial to construct a derivation of $\widehat{\Psi} \vdash [\Phi]t : [\Phi]t'$.

Case $T = [\Phi]\Phi'$ $\triangleright$ From part 2 we get that if $\mathsf{relevant}(\Psi \vdash \Phi,\ \Phi'\ \mathrm{wf}) = \widehat{\Psi}$, then $\widehat{\Psi} \vdash \Phi,\ \Phi'$ wf, thus the desired follows trivially.

Part 2 By induction on the derivation of $\Psi \vdash \Phi$ wf.

Case $\Phi = \bullet$ $\triangleright$ Trivially we have that $\mathsf{unspec}_\Psi \vdash \bullet$ wf.

Case $\Phi = \Phi,\ t$ $\triangleright$ We have that if $\mathsf{relevant}(\Psi;\ \Phi \vdash t : s) = \widehat{\Psi}$, then $\widehat{\Psi};\ \Phi \vdash t : s$ by part 3, and furthermore using the implicit requirement that Φ is well-formed, we also get that $\widehat{\Psi} \vdash \Phi$ wf. Thus using the appropriate typing rule we get $\widehat{\Psi} \vdash (\Phi,\ t)$ wf.

Case $\Phi = \Phi,\ X_i$ $\triangleright$ By induction we get that if $\mathsf{relevant}(\Psi \vdash \Phi\ \mathrm{wf}) = \widehat{\Psi}$, then $\widehat{\Psi} \vdash \Phi$ wf, and thus also $\widehat{\Psi} \circ (\Psi \widehat{@} i) \vdash \Phi$ wf. Furthermore, $(\widehat{\Psi} \circ (\Psi \widehat{@} i)).i = \Psi.i$. Thus using the appropriate well-formedness rule for Φ we get that $\widehat{\Psi} \vdash (\Phi,\ X_i)$ wf.

Part 3 By induction on the derivation of $\Psi;\ \Phi \vdash t : t'$.

Case $t = c$ $\triangleright$ Trivially we have that $\widehat{\Psi};\ \Phi \vdash c : t$ for any $\widehat{\Psi}$, Φ such that $\widehat{\Psi} \vdash \Phi$ wf, which holds for the corresponding $\widehat{\Psi}$ based on part 2.

Case $t = s$ $\triangleright$ Similarly as above.

Case $t = f_{\mathbf{I}}$ $\triangleright$ Again, as above.

Case $t = \Pi(t_1).t_2$ $\triangleright$ Simple by induction hypothesis for $\lceil t_2 \rceil$, and also from the fact that $\mathsf{relevant}(\Psi;\ \Phi \vdash t_1 : s) \sqsubseteq \mathsf{relevant}(\Psi \vdash (\Phi,\ t_1)\ \mathrm{wf}) \sqsubseteq \mathsf{relevant}(\Psi;\ \Phi,\ t_1 \vdash \lceil t_2 \rceil : s')$.

Case $t = \lambda(t_1).t_2$ $\triangleright$ By induction hypothesis for $\lceil t_2 \rceil$, if $\mathsf{relevant}(\Psi;\ \Phi,\ t_1 \vdash \lceil t_2 \rceil : s') = \widehat{\Psi}$, we get that $\widehat{\Psi};\ \Phi,\ t_1 \vdash \lceil t_2 \rceil : s'$. Thus we also have that $\widehat{\Psi};\ \Phi \vdash t_1 : s$, and also that either $t' = \mathsf{Type}'$ (which is an impossible case), or $\widehat{\Psi};\ \Phi,\ t_1 \vdash t' : s''$. By inversion of typing for $\Psi;\ \Phi \vdash \Pi(t_1).\lfloor t' \rfloor : s'$ we get that in fact $s'' = s'$, and thus it is easy to derive $\widehat{\Psi};\ \Phi,\ t_1 \vdash t' : s'$ and $\widehat{\Psi};\ \Phi \vdash \Pi(t_1).\lfloor t' \rfloor : s'$. From these we get the desired derivation for $\widehat{\Psi};\ \Phi \vdash \lambda(t_1).t_2 : \Pi(t_1).\lfloor t' \rfloor$.

Case $t = t_1\ t_2$ $\triangleright$ Trivial by induction hypothesis for t_1 and t_2.

Case $t = (t_1 = t_2)$ $\triangleright$ Again, trivial by induction hypothesis for t_1 and t_2, and also from the fact that $\widehat{\Psi}_1;\ \Phi \vdash t_1 : t$ implies $\widehat{\Psi}_1;\ \Phi \vdash t : \mathsf{Type}$.

Case $t = X_i/\sigma$ $\triangleright$ From the first part (relevancy of T under the prefix context), we get that $\vdash \widehat{\Psi}$ wf. Furthermore, using part 4 we get that $\widehat{\Psi};\ \Phi \vdash \sigma : \Phi'$. Last, it is trivial to derive $\widehat{\Psi};\ \Phi \vdash X_i/\sigma : t' \cdot \sigma$ using the same typing rule, since $\widehat{\Psi}.i = \Psi.i$.

Part 4 By induction on the derivation of $\Psi;\ \Phi \vdash \sigma : \Phi'$.

Case $\sigma = \bullet$ $\triangleright$ Trivial.

Case $\sigma = \sigma', t$ $\triangleright$ Trivial by induction hypothesis and use of part 3.

Case $\sigma = \sigma', \mathbf{id}(X_i)$ $\triangleright$ By induction hypothesis get $\widehat{\Psi};\ \Phi \vdash \sigma : \Phi'$. Furthermore, from $\widehat{\Psi} \vdash \Phi$ wf and the fact that $\Phi',\ X_i \subseteq \Phi$, we have by repeated typing inversions that $\Psi @ i \sqsubseteq \widehat{\Psi}$. Thus $\widehat{\Psi}.i = \Psi.i$, and we can construct a derivation for $\widehat{\Psi};\ \Phi \vdash (\sigma,\ \mathbf{id}(X_i)) : (\Phi',\ X_i)$.

Definition D.17 *Applying an extension substitution to a partial context is defined as follows, assuming that the partial context does not contain extension variables bigger than* $X_{|\Psi|-1}$.

$\boxed{\widehat{\Psi} \cdot \sigma_\Psi}$

$$
\begin{array}{lcl}
\bullet \cdot \sigma_\Psi & = & \bullet \\
(\widehat{\Psi},\ K) \cdot \sigma_\Psi & = & \widehat{\Psi} \cdot \sigma_\Psi,\ K \cdot (\sigma_\Psi,\ X_{|\Psi|},\ \cdots,\ X_{|\Psi|+|\widehat{\Psi}|}) \\
(\widehat{\Psi},\ ?) \cdot \sigma_\Psi & = & \widehat{\Psi} \cdot \sigma_\Psi,\ ?
\end{array}
$$

Lemma D.18 (Relevancy and extension substitution) *1. If* $\mathsf{unspec}_\Psi, \widehat{\Psi}_u \sqsubseteq \mathsf{relevant}(\Psi,\ \Psi_u \vdash T : K)$, $\Psi' \vdash \sigma_\Psi : \Psi$, *and* $\sigma'_\Psi = \sigma_\Psi, X_{|\Psi'|}, \cdots, X_{|\Psi'|+|\Psi_u|}$, *then* $\mathsf{unspec}_{\Psi'}, \widehat{\Psi}_u \cdot \sigma_\Psi \sqsubseteq \mathsf{relevant}(\Psi',\ \Psi_u \cdot \sigma_\Psi \vdash T \cdot \sigma'_\Psi : K \cdot \sigma'_\Psi)$.

2. If $\mathsf{unspec}_\Psi, \widehat{\Psi}_u \sqsubseteq \mathsf{relevant}(\Psi,\ \Psi_u \vdash \Phi\ wf)$, $\Psi' \vdash \sigma_\Psi : \Psi$, *and* $\sigma'_\Psi = \sigma_\Psi, X_{|\Psi'|}, \cdots, X_{|\Psi'|+|\Psi_u|}$, *then* $\mathsf{unspec}_{\Psi'}, \widehat{\Psi}_u \cdot \sigma_\Psi \sqsubseteq \mathsf{relevant}(\Psi',\ \Psi_u \cdot \sigma_\Psi \vdash \Phi \cdot \sigma'_\Psi\ wf)$.

3. If $\mathsf{unspec}_\Psi, \widehat{\Psi}_u \sqsubseteq \mathsf{relevant}(\Psi,\ \Psi_u;\ \Phi \vdash t : t')$, $\Psi' \vdash \sigma_\Psi : \Psi$, *and* $\sigma'_\Psi = \sigma_\Psi, X_{|\Psi'|}, \cdots, X_{|\Psi'|+|\Psi_u|}$, *then* $\mathsf{unspec}_{\Psi'}, \widehat{\Psi}_u \cdot \sigma_\Psi \sqsubseteq \mathsf{relevant}(\Psi',\ \Psi_u \cdot \sigma_\Psi;\ \Phi \cdot \sigma'_\Psi \vdash t \cdot \sigma'_\Psi : t' \cdot \sigma'_\Psi)$.

4. If $\mathsf{unspec}_\Psi, \widehat{\Psi}_u \sqsubseteq \mathsf{relevant}(\Psi,\ \Psi_u;\ \Phi \vdash \sigma : \Phi')$, $\Psi' \vdash \sigma_\Psi : \Psi$, *and* $\sigma'_\Psi = \sigma_\Psi, X_{|\Psi'|}, \cdots, X_{|\Psi'|+|\Psi_u|}$, *then* $\mathsf{unspec}_{\Psi'}, \widehat{\Psi}_u \cdot \sigma_\Psi \sqsubseteq \mathsf{relevant}(\Psi',\ \Psi_u \cdot \sigma_\Psi;\ \Phi \cdot \sigma'_\Psi \vdash \sigma \cdot \sigma'_\Psi : \Phi' \cdot \sigma'_\Psi)$.

Part 1 By induction on the typing derivation of T, and use of parts 2 and 3.

Part 2 By induction on the well-formedness derivation of Φ.

Case $\Phi = \bullet$ $\triangleright$ Trivial.

Case $\Phi = \Phi', t$ $\triangleright$ Using part 3 we get the desired result.

Case $\Phi = \Phi', X_i$ $\triangleright$
We have that $\mathsf{unspec}_\Psi,\ \widehat{\Psi}_u \sqsubseteq \mathsf{relevant}\,(\Psi,\ \Psi_u \vdash \Phi'\ \mathrm{wf}) \circ ((\Psi,\ \Psi_u)\,\widehat{@}\,i)$.
We split cases based on whether $i < |\Psi|$ or not.
In the first case:
We trivially have $\mathsf{unspec}_\Psi,\ \widehat{\Psi}_u \sqsubseteq \mathsf{relevant}\,(\Psi,\ \Psi_u \vdash \Phi'\ \mathrm{wf})$, thus directly by use of the induction hypothesis and the same rule for relevancy we get the desired.

In the second case:
Assume without loss of generality $\widehat{\Psi}'_u$ such that $\mathsf{unspec}_\Psi,\ \widehat{\Psi}'_u \sqsubseteq \mathsf{relevant}\,(\Psi,\ \Psi_u \vdash \Phi'\ \mathrm{wf})$, and $(\mathsf{unspec}_\Psi,\ \widehat{\Psi}_u) = (\mathsf{unspec}_\Psi,\ \widehat{\Psi}'_u) \circ ((\Psi,\ \Psi_u)\,\widehat{@}\,i)$.
Then by induction hypothesis get that $\mathsf{unspec}_{\Psi'},\ \widehat{\Psi}'_u \cdot \sigma_\Psi \sqsubseteq \mathsf{relevant}\,(\Psi',\ \Psi_u \cdot \sigma_\Psi \vdash \Phi' \cdot \sigma'_\Psi\ \mathrm{wf})$.
Now we have that $(\Phi',\ X_i) \cdot \sigma'_\Psi = \Phi' \cdot \sigma'_\Psi,\ X_{i-|\Psi|+|\Psi'|}$.
Thus $\mathsf{relevant}\,(\Psi',\ \Psi_u \cdot \sigma_\Psi \vdash (\Phi',\ X_i) \cdot \sigma'_\Psi\ \mathrm{wf}) = \mathsf{relevant}\left(\Psi',\ \Psi_u \cdot \sigma_\Psi \vdash (\Phi' \cdot \sigma'_\Psi,\ X_{i-|\Psi|+|\Psi'|})\ \mathrm{wf}\right) = \mathsf{relevant}\,(\Psi',\ \Psi_u \cdot \sigma_\Psi \vdash \Phi' \cdot \sigma'_\Psi\ \mathrm{wf}) \circ ((\Psi',\ \Psi_u \cdot \sigma_\Psi)\,\widehat{@}\,i - |\Psi| + |\Psi'|)$.
Thus we have that $(\mathsf{unspec}_{\Psi'},\ \widehat{\Psi}'_u \cdot \sigma_\Psi) \circ ((\Psi',\ \Psi_u \cdot \sigma_\Psi)\,\widehat{@}\,i - |\Psi| + |\Psi'|) \sqsubseteq \mathsf{relevant}\,(\Psi',\ \Psi_u \cdot \sigma_\Psi \vdash (\Phi',\ X_i) \cdot \sigma'_\Psi\ \mathrm{wf})$.
But $(\mathsf{unspec}_{\Psi'},\ \widehat{\Psi}'_u \cdot \sigma_\Psi) \circ ((\Psi',\ \Psi_u \cdot \sigma_\Psi)\,\widehat{@}\,i - |\Psi| + |\Psi'|) = \mathsf{unspec}_{\Psi'},\ \widehat{\Psi}_u \cdot \sigma_\Psi$.
This is because $(\mathsf{unspec}_\Psi,\ \widehat{\Psi}_u) = (\mathsf{unspec}_\Psi,\ \widehat{\Psi}'_u) \circ ((\Psi,\ \Psi_u)\,\widehat{@}\,i)$, so the i-th element is the only one where $\mathsf{unspec}_\Psi, \widehat{\Psi}'_u$ might differ from $\mathsf{unspec}_\Psi, \widehat{\Psi}_u$; this will be the $i - |\Psi| + |\Psi'|$-th element after σ'_Ψ is applied; and that element is definitely equal after the join.

Part 3 By induction on the typing derivation for t.

Case $t = c, s,$ or $v_{\mathbf{I}}$ $\triangleright$ Trivial using part 2.

Case $t = \Pi(t_1).t_2$ $\triangleright$ By induction hypothesis for $\lceil t_2 \rceil$.

Case $t = \lambda(t_1).t_2$ $\triangleright$ By induction hypothesis for $\lceil t_2 \rceil$.

Case $t = t_1\ t_2$ $\triangleright$ Assume $\widehat{\Psi}_1$ and $\widehat{\Psi}_2$ such that $\widehat{\Psi} = \widehat{\Psi}_1 \circ \widehat{\Psi}_2$. Then use induction hypothesis for t_1 and t_2. Last combine the results using join to get the desired, noticing that both $\widehat{\Psi}_1 \cdot \sigma_\Psi$ and $\widehat{\Psi}_2 \cdot \sigma_\Psi$ are $\sqsubseteq \widehat{\Psi} \cdot \sigma_\Psi$ (so join is defined between them), and also that $(\widehat{\Psi}_1 \circ \widehat{\Psi}_2) \cdot \sigma_\Psi = \widehat{\Psi}_1 \cdot \sigma_\Psi \circ \widehat{\Psi}_2 \circ \sigma_\Psi$.

Case $t = X_i/\sigma$ $\triangleright$
We split cases based on whether $i < |\Psi|$ or not. In case it is, the proof is trivial using part 4. We thus focus on the case where $i \geq |\Psi|$.
We have that $\mathsf{unspec}_\Psi,\ \widehat{\Psi} \sqsubseteq \mathsf{relevant}\,((\Psi, \Psi_u)\lfloor_i \vdash [\Phi']t : [\Phi']s) \circ \mathsf{relevant}\,(\Psi,\ \Psi_u;\ \Phi \vdash \sigma : \Phi') \circ ((\Psi,\ \Psi_u)\,\widehat{@}\,i)$.
Assume $\widehat{\Psi}^1_u, \widehat{\Psi}'_u, \widehat{\Psi}^2_u$ such that $(\widehat{\Psi}^1_u = \widehat{\Psi}'_u,\ \overbrace{?, \cdots, ?}^{|\Psi|+|\Psi_u|-i \text{ times}})$, $\mathsf{unspec}_\Psi,\ \widehat{\Psi}'_u \sqsubseteq \mathsf{relevant}\,((\Psi, \Psi_u)\lfloor_i \vdash [\Phi']t : [\Phi']s)$, $\mathsf{unspec}_\Psi,\ \widehat{\Psi}^2_u \sqsubseteq \mathsf{relevant}\,(\Psi,\ \Psi_u;\ \Phi \vdash \sigma : \Phi')$ and last that $\widehat{\Psi} = \widehat{\Psi}^1_u \circ \widehat{\Psi}^2_u \circ ((\Psi,\ \Psi_u\,\widehat{@}\,i))$.
By induction hypothesis for $[\Phi']t$ we get that:
$\mathsf{unspec}_{\Psi'},\ \widehat{\Psi}'_u \cdot \sigma_\Psi \sqsubseteq \mathsf{relevant}\,((\Psi', \Psi_u \cdot \sigma_\Psi)\lfloor_i \vdash [\Phi' \cdot \sigma'_\Psi]t \cdot \sigma'_\Psi : [\Phi' \cdot \sigma'_\Psi]s \cdot \sigma'_\Psi)$.
By induction hypothesis for σ we get that:
$\mathsf{unspec}_{\Psi'},\ \widehat{\Psi}^2_u \cdot \sigma_\Psi \sqsubseteq \mathsf{relevant}\,(\Psi',\ \Psi_u \cdot \sigma_\Psi;\ \Phi \cdot \sigma'_\Psi \vdash \sigma \cdot \sigma'_\Psi : \Phi' \cdot \sigma'_\Psi)$
We combine the above to get the desired, using the properties of join at $\widehat{@}$ as we did earlier.

Case (rest) $\triangleright$ Similar to the above cases.

Part 4 Similar as above.

D.4 Unification

Here, we are matching a term with some unification variables against a closed term. Therefore we will use typing judgements like $\Psi \vdash_p T : K$ instead of $\Psi',\ \Psi_u \vdash_p T : K$, as we did above. The single Ψ that we use actually corresponds to Ψ_u; the normal context Ψ' is empty.

First, we need to define the notion of partial substitutions, corresponding to substitutions for partial contexts as defined above.

Definition D.19 (Partial substitutions) *The syntax for partial substitutions follows.*

$\widehat{\sigma_\Psi} ::= \bullet \mid \widehat{\sigma_\Psi},\ T \mid \widehat{\sigma_\Psi},\ ?$

Definition D.20 *Joining two partial substitutions is defined below.*

$\boxed{\widehat{\sigma_\Psi} \circ \widehat{\sigma_\Psi}'}$

$$\begin{array}{lcl}
\bullet \circ \bullet & = & \bullet \\
(\widehat{\sigma_\Psi},\ T) \circ (\widehat{\sigma_\Psi}',\ T) & = & (\widehat{\sigma_\Psi} \circ \widehat{\sigma_\Psi}'),\ T \\
(\widehat{\sigma_\Psi},\ ?) \circ (\widehat{\sigma_\Psi}',\ T) & = & (\widehat{\sigma_\Psi} \circ \widehat{\sigma_\Psi}'),\ T \\
(\widehat{\sigma_\Psi},\ T) \circ (\widehat{\sigma_\Psi}',\ ?) & = & (\widehat{\sigma_\Psi} \circ \widehat{\sigma_\Psi}'),\ T \\
(\widehat{\sigma_\Psi},\ ?) \circ (\widehat{\sigma_\Psi}',\ ?) & = & (\widehat{\sigma_\Psi} \circ \widehat{\sigma_\Psi}'),\ ?
\end{array}$$

Definition D.21 *Comparing two partial substitutions is defined below.*

$\boxed{\widehat{\sigma_\Psi} \sqsubseteq \widehat{\sigma_\Psi}'}$

$$\begin{array}{lcl}
\bullet \sqsubseteq \bullet & & \\
(\widehat{\sigma_\Psi},\ T) \sqsubseteq (\widehat{\sigma_\Psi}',\ T) & \Leftarrow & \widehat{\sigma_\Psi} \sqsubseteq \widehat{\sigma_\Psi}' \\
(\widehat{\sigma_\Psi},\ ?) \sqsubseteq (\widehat{\sigma_\Psi}',\ T) & \Leftarrow & \widehat{\sigma_\Psi} \sqsubseteq \widehat{\sigma_\Psi}' \\
(\widehat{\sigma_\Psi},\ ?) \sqsubseteq (\widehat{\sigma_\Psi}',\ ?) & \Leftarrow & \widehat{\sigma_\Psi} \sqsubseteq \widehat{\sigma_\Psi}'
\end{array}$$

Definition D.22 *The fully unspecified substitution for a specific partial context is defined as:*

$\boxed{\mathsf{unspec}_{\widehat{\Psi}} = \widehat{\sigma_\Psi}}$

$$\begin{array}{lcl}
\mathsf{unspec}_{\bullet} & = & ? \\
\mathsf{unspec}_{\widehat{\Psi},\ ?} & = & \mathsf{unspec}_{\widehat{\Psi}},\ ? \\
\mathsf{unspec}_{\widehat{\Psi},\ K} & = & \mathsf{unspec}_{\widehat{\Psi}},\ ?
\end{array}$$

Definition D.23 *Applying a partial extension substitution to a term, a context, or a substitution is entirely identical to normal substitution. It fails when a metavariable that is left unspecified in the extension substitution gets used, something that already happens from the existing definition B.77.*

Definition D.24 *Replacing an unspecified element of a partial substitution with another works as follows.*

$\boxed{\widehat{\sigma_\Psi}[i \mapsto T] = \widehat{\sigma_\Psi}'}$

$$\begin{array}{lcl}
(\widehat{\sigma_\Psi},\ ?)[i \mapsto T] & = & \widehat{\sigma_\Psi},\ T \text{ when } i = |\widehat{\sigma_\Psi}| \\
(\widehat{\sigma_\Psi},\ T')[i \mapsto T] & = & \widehat{\sigma_\Psi}[i \mapsto T],\ T' \text{ when } i < |\widehat{\sigma_\Psi}| \\
(\widehat{\sigma_\Psi},\ ?)[i \mapsto T] & = & \widehat{\sigma_\Psi}[i \mapsto T],\ ? \text{ when } i < |\widehat{\sigma_\Psi}|
\end{array}$$

Definition D.25 *Limiting a partial substitution to a specific partial context works as follows; we assume* $|\widehat{\sigma_\Psi}| = |\widehat{\Psi}|$.

$\boxed{\widehat{\sigma_\Psi}|_{\widehat{\Psi}}}$

$$\begin{array}{lcl}
(\bullet)|_{\bullet} & = & \bullet \\
(\widehat{\sigma_\Psi},\ T)|_{\widehat{\Psi},\ ?} & = & \widehat{\sigma_\Psi}|_{\widehat{\Psi}},\ ? \\
(\widehat{\sigma_\Psi},\ T)|_{\widehat{\Psi},\ K} & = & \widehat{\sigma_\Psi}|_{\widehat{\Psi}},\ T \\
(\widehat{\sigma_\Psi},\ ?)|_{\widehat{\Psi},\ ?} & = & \widehat{\sigma_\Psi}|_{\widehat{\Psi}},\ ?
\end{array}$$

Definition D.26 *Typing for partial substitutions is defined below.*

$\boxed{\bullet \vdash \widehat{\sigma_\Psi} : \widehat{\Psi}}$

$$\frac{}{\bullet \vdash \bullet : \bullet} \qquad \frac{\bullet \vdash \widehat{\sigma_\Psi} : \widehat{\Psi} \qquad \bullet \vdash T : K \cdot \widehat{\sigma_\Psi}}{\bullet \vdash_p (\widehat{\sigma_\Psi},\ T) : (\widehat{\Psi},\ K)} \qquad \frac{\bullet \vdash_p \widehat{\sigma_\Psi} : \widehat{\Psi}}{\bullet \vdash_p (\widehat{\sigma_\Psi},\ ?) : (\widehat{\Psi},\ ?)}$$

Lemma D.27 *If* $\bullet \vdash \widehat{\sigma_{\Psi_1}} : \widehat{\Psi}_1$ *and* $\bullet \vdash \widehat{\sigma_{\Psi_2}} : \widehat{\Psi}_2$*, with* $\widehat{\Psi}_1 \circ \widehat{\Psi}_2$ *and* $\widehat{\sigma_{\Psi_1}} \circ \widehat{\sigma_{\Psi_2}}$ *defined, then* $\bullet \vdash \widehat{\sigma_{\Psi_1}} \circ \widehat{\sigma_{\Psi_2}} : \widehat{\Psi}_1 \circ \widehat{\Psi}_2$.

By induction on the derivation of $\widehat{\sigma_{\Psi_1}} \circ \widehat{\sigma_{\Psi_2}} = \widehat{\sigma_\Psi}'$.

Case $\bullet \circ \bullet \ \triangleright$ Trivial, since $\widehat{\Psi}_1 = \widehat{\Psi}_2 = \bullet$ by typing inversion.

Case $(\widehat{\sigma_{\Psi_1}}',\ T) \circ (\widehat{\sigma_{\Psi_2}}',\ T) \ \triangleright$ By typing inversion get $\widehat{\Psi}_1 = \widehat{\Psi}_1',\ K$ with $T : K$, and $\widehat{\Psi}_2 = \widehat{\Psi}_2',\ K$ with $T : K$. Thus $\widehat{\Psi}_1 \circ \widehat{\Psi}_2 = \widehat{\Psi}_1' \circ \widehat{\Psi}_2',\ K$, and by induction hypothesis for $\widehat{\sigma_{\Psi_1}}', \widehat{\sigma_{\Psi_2}}'$ and typing it is easy to prove the desired.

Case $\dfrac{(\widehat{\sigma_{\Psi_1}}',\ ?) \circ (\widehat{\sigma_{\Psi_2}}',\ T)}{B} \ \triangleright$ y typing inversion get $\widehat{\Psi}_1 = \widehat{\Psi}_1',\ ?$, and $\widehat{\Psi}_2 = \widehat{\Psi}_2',\ K$ with $T : K$. Thus $\widehat{\Psi}_1 \circ \widehat{\Psi}_2 = \widehat{\Psi}_1' \circ \widehat{\Psi}_2',\ K$, and by induction hypothesis for $\widehat{\sigma_{\Psi_1}}', \widehat{\sigma_{\Psi_2}}'$ and typing it is easy to prove the desired.

Case $\dfrac{(\widehat{\sigma_{\Psi_1}}',\ T) \circ (\widehat{\sigma_{\Psi_2}}',\ ?)}{S} \ \triangleright$ imilar to the above.

Case $\dfrac{(\widehat{\sigma_{\Psi_1}}',\ ?) \circ (\widehat{\sigma_{\Psi_2}}',\ ?)}{A} \ \triangleright$ gain by induction hypothesis and the fact that $\widehat{\Psi}_1 = \widehat{\Psi}_1', ?$ and $\widehat{\Psi}_2 = \widehat{\Psi}_2', ?$ by typing inversion.

Lemma D.28 *If* $\bullet \vdash \widehat{\sigma_\Psi} : \widehat{\Psi}$*,* $\bullet \vdash \widehat{\Psi}'$ *wf and* $\widehat{\Psi}' \sqsubseteq \widehat{\Psi}$*, then* $\widehat{\sigma_\Psi}|_{\widehat{\Psi}'} \sqsubseteq \widehat{\sigma_\Psi}$ *and* $\bullet \vdash \widehat{\sigma_\Psi}|_{\widehat{\Psi}'} : \widehat{\Psi}'$.

Trivial by induction on the derivation of $\widehat{\sigma_\Psi}|_{\widehat{\Psi}'}$.

Now we are ready to proceed to a proof about the fact that either a unique unification partial substitution exists for patterns and terms that are typed under the restrictive typing, or that no such substitution exists. The constructive content of this proof will be our unification procedure.

To prove the following theorem we assume that if $\Psi;\ \Phi \vdash_p t : t'$, with $t' \neq \mathsf{Type}'$, the derivation $\Psi;\ \Phi \vdash_p t' : s$ for a suitable s is a sub-derivation of the derivation $\Psi;\ \Phi \vdash_p t : t'$. The way we have written our rules this is actually not true, but an adaptation where the $t' : s$ derivation becomes part of the $t : t'$ derivation is possible, thanks to the theorem B.68.

Theorem D.29 (Decidability and determinism of unification) *1. If $\Psi \vdash_p \Phi$ wf, $\bullet \vdash_p \Phi'$ wf, $\mathsf{relevant}(\Psi \vdash_p \Phi\ wf) = \widehat{\Psi}$, then there either exists a unique substitution $\widehat{\sigma_\Psi}$ such that $\bullet \vdash \widehat{\sigma_\Psi} : \widehat{\Psi}$ and $\Phi \cdot \widehat{\sigma_\Psi} = \Phi'$, or no such substitution exists.*

2. If $\Psi;\ \Phi \vdash_p t : t_T$, $\bullet;\ \Phi' \vdash_p t' : t'_T$ and $\mathsf{relevant}(\Psi;\ \Phi \vdash t : t'_T) = \widehat{\Psi'}$, then:
assuming that $\Psi;\ \Phi \vdash_p t_T : s$, $\bullet;\ \Phi \vdash_p t'_T : s$, $\mathsf{relevant}(\Psi;\ \Phi \vdash_p t_T : s) = \widehat{\Psi}$ (or, if $t_T = \mathsf{Type}'$, that $\Psi \vdash_p \Phi$ wf, $\bullet \vdash_p \Phi$ wf, $\mathsf{relevant}(\Psi \vdash_p \Phi\ wf) = \widehat{\Psi}$) and there exists a unique substitution $\widehat{\sigma_\Psi}$ such that $\bullet \vdash \widehat{\sigma_\Psi} : \widehat{\Psi}$, $\Phi \cdot \widehat{\sigma_\Psi} = \Phi'$ and $t_T \cdot \widehat{\sigma_\Psi} = t'_T$,
then there either exists a unique substitution $\widehat{\sigma_\Psi}'$ such that $\bullet \vdash \widehat{\sigma_\Psi}' : \widehat{\Psi'}$, $\Phi \cdot \widehat{\sigma_\Psi}' = \Phi'$, $t_T \cdot \widehat{\sigma_\Psi}' = t'_T$ and $t \cdot \widehat{\sigma_\Psi}' = t'$, or no such substitution exists.

3. If $\Psi \vdash_p T : K$, $\bullet \vdash_p T' : K$ and $\mathsf{relevant}(\Psi;\ \Phi \vdash T : K) = \Psi$, then either there exists a unique substitution σ_Ψ such that $\bullet \vdash \sigma_\Psi : \Psi$ and $T \cdot \sigma_\Psi = T'$, or no such substitution exists.

Part 2 By induction on the typing derivation for t.

Case $\dfrac{c : t \in \Sigma}{\Psi;\ \Phi \vdash_p c : t_T}\ \triangleright$

We have $t \cdot \widehat{\sigma_\Psi}' = c \cdot \widehat{\sigma_\Psi}' = c$. So for any substitution to satisfy the desired properties we need to have that $t' = c$ also; if this isn't so, no $\widehat{\sigma_\Psi}'$ possibly exists. If we have that $t = t' = c$, then the desired is proved directly by assumption, considering that $\mathsf{relevant}(\Psi;\ \Phi \vdash_p c : t) = \mathsf{relevant}(\Psi;\ \Phi \vdash_p t_T : s) = \mathsf{relevant}(\Psi \vdash_p \Phi\ \mathrm{wf})$ (since t_T comes from the definitions context and can therefore not contain extension variables).

Case $\dfrac{\Phi.\mathbf{I} = t}{\Psi;\ \Phi \vdash_p f_{\mathbf{I}} : t_T}\ \triangleright$

Similarly as above. First, we need $t' = f_{\mathbf{I}'}$, otherwise no suitable $\widehat{\sigma_\Psi}'$ exists. From assumption we have a unique $\widehat{\sigma_\Psi}$ for $\mathsf{relevant}(\Psi;\ \Phi \vdash_p t_T : s)$. If $\mathbf{I} \cdot \widehat{\sigma_\Psi} = \mathbf{I}'$, then $\widehat{\sigma_\Psi}$ has all the desired properties for $\widehat{\sigma_\Psi}'$, considering the fact that $\mathsf{relevant}(\Psi;\ \Phi \vdash_p f_{\mathbf{I}} : t_T) = \mathsf{relevant}(\Psi \vdash_p \Phi\ \mathrm{wf})$ and $\mathsf{relevant}(\Psi;\ \Phi \vdash_p t_T : s) = \mathsf{relevant}(\Psi \vdash_p \Phi\ \mathrm{wf})$ (since $t_T = \Phi.i$). It is also unique, because an alternate $\widehat{\sigma_\Psi}'$ would violate the assumed uniqueness of $\widehat{\sigma_\Psi}$. If $\mathbf{I} \cdot \widehat{\sigma_\Psi} \neq \widehat{\sigma_\Psi}'$, no suitable substitution exists, because of the same reason.

Case $\dfrac{(s, s') \in \mathcal{A}}{\Psi;\ \Phi \vdash_p s : s'}\ \triangleright$

Entirely similar to the case for c.

Case $\dfrac{\Psi;\ \Phi \vdash_p t_1 : s \qquad \Psi;\ \Phi,\ t_1 \vdash_p \lceil t_2 \rceil_{|\Phi|} : s' \qquad (s, s', s'') \in \mathcal{R}}{\Psi;\ \Phi \vdash_p \Pi(t_1).t_2 : s''}\ \triangleright$

First, we have either that $t' = \Pi(t'_1).t'_2$, or no suitable $\widehat{\sigma_\Psi}'$ exists. Thus by inversion for t' we get:
$\bullet;\ \Phi' \vdash_p t'_1 : s_*$, $\bullet;\ \Phi',\ t'_1 \vdash_p \lceil t'_2 \rceil_{|\Phi'|} : s'_*$, $(s_*, s'_*, s'') \in \mathcal{R}$.
Now, we need $s = s_*$, otherwise no suitable $\widehat{\sigma_\Psi}'$ possibly exists. To see why this is so, assume that a $\widehat{\sigma_\Psi}'$ satisfying the necessary conditions exists, and $s \neq s_*$; then we have that $t_1 \cdot \widehat{\sigma_\Psi}' = t'_1$, which means that their types should also match, a contradiction.
We use the induction hypothesis for t_1 and t'_1. We are allowed to do so because $\mathsf{relevant}(\Psi;\ \Phi \vdash_p s'' : s''') = \mathsf{relevant}(\Psi;\ \Phi \vdash_p s : s'''')$, and the other properties for $\widehat{\sigma_\Psi}$ also hold trivially.

From that we either get a $\widehat{\sigma_\Psi}'$ such that: $\bullet \vdash \widehat{\sigma_\Psi}' : \widehat{\Psi}'$, where $\widehat{\Psi}' = \mathsf{relevant}(\Psi;\ \Phi \vdash_p t_1 : s)$ and $t_1 \cdot \widehat{\sigma_\Psi}' = t_1'$, $\Phi \cdot \widehat{\sigma_\Psi}' = \Phi'$. Since a partial substitution unifying t with t' will also include a substitution that only has to do with $\widehat{\Psi}'$, we see that if no $\widehat{\sigma_\Psi}'$ is returned by the induction hypothesis, no suitable substitution for t and t' actually exists.
We can now use the induction hypothesis for t_2 and $\widehat{\sigma_\Psi}'$, since $\mathsf{relevant}(\Psi;\ \Phi\, t_1 \vdash_p s' : s'''') = \mathsf{relevant}(\Psi;\ \Phi \vdash_p t_1 : s)$, and the other requirements trivially hold. Especially for s' and s'_* being equal, this is trivial since both need to be equal to s'' (because of the form of our rule set $\mathcal{R}$).
From that we either get a $\widehat{\sigma_\Psi}''$ such that, $\bullet \vdash \widehat{\sigma_\Psi}'' : \widehat{\Psi}''$, $\lceil t_2 \rceil_{|\Phi|} \cdot \widehat{\sigma_\Psi}'' = \lceil t_2' \rceil_{|\Phi'|}$, $\Phi \cdot \widehat{\sigma_\Psi}'' = \Phi$ and $t_1 \cdot \widehat{\sigma_\Psi}'' = t_1'$, or that such $\widehat{\sigma_\Psi}''$ does not exist. In the second case we proceed as above, so we focus in the first case.
By use of properties of freshening (like injectivity) we are led to the fact that $(\Pi(t_1).t_2) \cdot \widehat{\sigma_\Psi}'' = \Pi(t_1').(t_2')$, so the returned $\widehat{\sigma_\Psi}''$ has the desired properties, if we consider the fact that $\mathsf{relevant}(\Psi;\ \Phi \vdash_p \Pi(t_1).t_2 : s'') = \mathsf{relevant}\left(\Psi;\ \Phi, t_1 \vdash_p \lceil t_2 \rceil_{|\Phi|} : s'\right)$.

Case $\dfrac{\Psi;\ \Phi \vdash_p t_1 : s \qquad \Psi;\ \Phi,\ t_1 \vdash_p \lceil t_2 \rceil_{|\Phi|} : t_3 \qquad \Psi;\ \Phi \vdash_p \Pi(t_1).\lfloor t_3 \rfloor_{|\Phi|,\,.} : s'}{\Psi;\ \Phi \vdash_p \lambda(t_1).t_2 : \Pi(t_1).\lfloor t_3 \rfloor_{|\Phi|,\,.}}\ \triangleright$

We have that either $t' = \lambda(t_1').t_2'$, or no suitable $\widehat{\sigma_\Psi}'$ exists. Thus by typing inversion for t' we get:
$\bullet;\ \Phi' \vdash_p t_1' : s_*$, $\bullet;\ \Phi',\ t_1' \vdash_p \lceil t_2' \rceil_{|\Phi'|} : t_3'$, $\bullet;\ \Phi' \vdash_p \Pi(t_1').\lfloor t_3 \rfloor_{|\Phi'|,\,.} : s'_*$.
By assumption we have that there exists a unique $\widehat{\sigma_\Psi}$ such that $\mathsf{relevant}\left(\Psi;\ \Phi \vdash_p \Pi(t_1).\lfloor t_3 \rfloor_{|\Phi|,\,.} : s\right) = \widehat{\Psi}$, $\bullet \vdash \widehat{\sigma_\Psi} : \widehat{\Psi}$, $\Phi \cdot \widehat{\sigma_\Psi} = \Phi'$, $(\Pi(t_1).\lfloor t_3 \rfloor) \cdot \widehat{\sigma_\Psi} = \Pi(t_1').\lfloor t_3' \rfloor$, if $\mathsf{relevant}\left(\Psi;\ \Phi \vdash_p \Pi(t_1).\lfloor t_3 \rfloor_{|\Phi,\cdot|}\right) = \widehat{\Psi}$. From that we also get that $s' = s'_*$.
From the fact that $(\Pi(t_1).\lfloor t_3 \rfloor) \cdot \widehat{\sigma_\Psi} = \Pi(t_1').\lfloor t_3' \rfloor$, we get first of all that $t_1 \cdot \widehat{\sigma_\Psi} = t_1'$, and also that $t_3 \cdot \widehat{\sigma_\Psi} = t_3'$.
Furthermore, We have that $\mathsf{relevant}(\Psi;\ \Phi \vdash_p \Pi(t_1).\lfloor t_3 \rfloor : s') = \mathsf{relevant}(\Psi;\ \Phi,\ t_1 \vdash_p t_3 : s')$.
From that we understand that $\widehat{\sigma_\Psi}$ is a suitable substitution to use for the induction hypothesis for $\lceil t_2 \rceil$.
Thus from induction hypothesis we either get a unique $\widehat{\sigma_\Psi}'$ with the properties: $\bullet \vdash \widehat{\sigma_\Psi}' : \widehat{\Psi}'$, $\lceil t_2 \rceil \cdot \widehat{\sigma_\Psi}' = \lceil t_2' \rceil$, $(\Phi,\ t_1) \cdot \widehat{\sigma_\Psi} = \Phi'$, t_1', $t_3 \cdot \widehat{\sigma_\Psi} = t_3'$, if $\mathsf{relevant}\left(\Psi;\ \Phi,\ t_1 \vdash_p \lceil t_2 \rceil_{|\Phi|} : t_3\right) = \widehat{\Psi}$, or that no such substitution exists.
We focus on the first case; in the second case no unifying substitution for t and t' exists, otherwise the lack of existence of a suitable $\widehat{\sigma_\Psi}'$ would lead to a contradiction.
This substitution $\widehat{\sigma_\Psi}'$ has the desired properties with respect to unification of t against t' (again using the properties of freshening, like injectivity), and it is unique, because the existence of an alternate substitution with the same properties would violate the uniqueness assumption of the substitution returned by induction hypothesis.

Case $\dfrac{\Psi;\ \Phi \vdash_p t_1 : \Pi(t_a).t_b \qquad \Psi;\ \Phi \vdash_p t_2 : t_a}{\Psi;\ \Phi \vdash_p t_1\ t_2 : \lceil t_b \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2)}\ \triangleright$

Again we have that either $t' = t_1'\ t_2'$, or no suitable substitution possibly exists. Thus by inversion of typing for t' we get:
$\bullet;\ \Phi \vdash_p t_1' : \Pi(t_a').t_b'$, $\bullet;\ \Phi \vdash_p t_2' : t_a'$, $t_T' = \left\lceil t_b' \right\rceil_{|\Phi'|} \cdot (\mathsf{id}_{\Phi'},\ t_2')$.
Furthermore we have that $\Psi;\ \Phi \vdash_p \Pi(t_a).t_b : s$ and $\bullet;\ \Phi \vdash_p \Pi(t_a').t_b' : s'$ for suitable s, s'. We need $s = s'$, otherwise no suitable $\widehat{\sigma_\Psi}'$ exists (because if t_1 and t_1' were unifiable by substitution, their Π-types would match, and also their sorts, which is a contradiction).
We can use the induction hypothesis for $\Pi(t_a).t_b$ and $\Pi(t_a').t_b'$, with the partial substitution $\widehat{\sigma_\Psi}$ limited only to those variables relevant in $\Psi \vdash_p \Phi$ wf. In that case all of the requirements for $\widehat{\sigma_\Psi}$ hold (the uniqueness condition also holds for this substitution, using part 1 for the fact that Φ and Φ' only have a unique unification substitution), so we get from the induction hypothesis either a $\widehat{\sigma_\Psi}'$ for $\widehat{\Psi}' = \mathsf{relevant}(\Psi;\ \Phi \vdash_p \Pi(t_a).t_b : s)$ such that $\Phi \cdot \widehat{\sigma_\Psi} = \Phi'$ and $(\Pi(t_a).t_b) \cdot \widehat{\sigma_\Psi} = \Pi(t_a').t_b'$, or that no such $\widehat{\sigma_\Psi}'$ exists. In the second case, again we can show

that no suitable substitution for t and t' exists; so we focus in the first case.
We can now use the induction hypothesis for t_1, using this $\widehat{\sigma_\Psi}'$. From that, we get that either a $\widehat{\sigma_{\Psi_1}}$ exists for $\widehat{\Psi}_1 = \mathsf{relevant}(\Psi;\ \Phi \vdash_p t_1 : \Pi(t_a).t_b)$ such that $t_1 \cdot \widehat{\sigma_{\Psi_1}} = t_1'$ etc., or that no such $\widehat{\sigma_{\Psi_1}}$ exists, in which case we argue that no global $\widehat{\sigma_\Psi}'$ exists for unifying t and t' (because we could limit it to the $\widehat{\Psi}_1$ variables and yield a contradiction).
We now form $\widehat{\sigma_\Psi}''$ which is the limitation of $\widehat{\sigma_\Psi}'$ to the context $\widehat{\Psi}'' = \mathsf{relevant}(\Psi;\ \Phi \vdash_p t_a : s_*)$. For that, we have that $\bullet \vdash_p \widehat{\sigma_\Psi}'' : \widehat{\Psi}''$, $\Phi \cdot \widehat{\sigma_\Psi}'' = \Phi'$ and $t_a \cdot \widehat{\sigma_\Psi}'' = t_a$. Also it is the unique substitution with those properties, otherwise the induction hypothesis for t_a would be violated.
Using $\widehat{\sigma_\Psi}''$ we can allude to the induction hypothesis for t_2, which either yields a substitution $\widehat{\sigma_{\Psi_2}}$ for $\widehat{\Psi}_2 = \mathsf{relevant}(\Psi;\ \Phi \vdash_p t_2 : t_a)$, such that $t_2 \cdot \widehat{\sigma_{\Psi_2}} = t_2'$, etc., or that no such substitution exists, which we prove implies no global unifying substitution exists.
Having now the $\widehat{\sigma_{\Psi_1}}$ and $\widehat{\sigma_{\Psi_2}}$ specified above, we consider the substitution $\widehat{\sigma_{\Psi_r}} = \widehat{\sigma_{\Psi_1}} \circ \widehat{\sigma_{\Psi_2}}$. This substitution, if it exists, has the desired properties: we have that $\widehat{\Psi}_r = \mathsf{relevant}(\Psi;\ \Phi \vdash_p t_1\ t_2 : \lceil t_b \rceil \cdot (\mathsf{id}_\Phi,\ t_2)) = \mathsf{relevant}(\Psi;\ \Phi \vdash_p t_1 : \Pi(t_a).t_b) \circ \mathsf{relevant}(\Psi;\ \Phi \vdash_p t_2 : t_a)$, and thus $\bullet \vdash \widehat{\sigma_{\Psi_r}} : \widehat{\Psi}_r$. Also, $(t_1\ t_2) \cdot \widehat{\sigma_{\Psi_r}} = t_1'\ t_2'$, $t_T \cdot \widehat{\sigma_{\Psi_r}} = t_T'$ (because $t_b \cdot \widehat{\sigma_{\Psi_r}} = t_b'$ etc.), and $\Phi \cdot \widehat{\sigma_{\Psi_r}} = \Phi'$. It is also unique: if another substitution had the same properties, we could limit it to either the relevant variables for t_1 or t_2 and get a contradiction. Thus this is the desired substitution.
If $\widehat{\sigma_{\Psi_r}}$ does not exist, then no suitable substitution for unifying t and t' exists. This is again because we could limit any potential such substitution to two parts, $\widehat{\sigma_{\Psi_1}}'$ and $\widehat{\sigma_{\Psi_2}}'$ (for $\widehat{\Psi}_1$ and $\widehat{\Psi}_2$ respectively), violating the uniqueness of the substitutions yielded by the induction hypothesis.

Case $\dfrac{\Psi;\ \Phi \vdash_p t_1 : t_a \qquad \Psi;\ \Phi \vdash_p t_2 : t_a \qquad \Psi;\ \Phi \vdash_p t_a : \mathsf{Type}}{\Psi;\ \Phi \vdash_p t_1 = t_2 : \mathsf{Prop}}\ \triangleright$

Similarly as above. First assume that $t' = (t_1' = t_2')$, with $t_1' : t_a'$, $t_2' : t_a'$ and $t_a' : \mathsf{Type}$. Then, by induction hypothesis get a unifying substitution $\widehat{\sigma_\Psi}'$ for t_a and t_a'. Use that $\widehat{\sigma_\Psi}'$ in order to allude to the induction hypothesis for t_1 and t_2 independently, yielding substitutions $\widehat{\sigma_{\Psi_1}}$ and $\widehat{\sigma_{\Psi_2}}$. Last, claim that the globally required substitution must actually be equal to $\widehat{\sigma_{\Psi_1}} \circ \widehat{\sigma_{\Psi_2}}$.

Case $\dfrac{(\Psi).i = T \qquad T = [\Phi_*]t_T \qquad \Psi;\ \Phi \vdash_p \sigma : \Phi_* \qquad \Phi_* \subseteq \Phi \qquad \sigma = \mathsf{id}_{\Phi_*}}{\Psi;\ \Phi \vdash_p X_i/\sigma : t_T \cdot \sigma}\ \triangleright$

We trivially have $t_T \cdot \sigma = t_T$. We split cases depending on whether $\widehat{\sigma_\Psi}.i =?$ or not. If it is unspecified:
We split cases further depending on whether t' uses any variables higher than $|\Phi_* \cdot \widehat{\sigma_\Psi}| - 1$ or not. That is, if $t' <^f |\Phi_* \cdot \widehat{\sigma_\Psi}|$ or not. In the case where this doesn't hold, it is obvious that there is no possible $\widehat{\sigma_\Psi}'$ such that $(X_i/\sigma) \cdot \widehat{\sigma_\Psi}' = t'$, since $\widehat{\sigma_\Psi}'$ must include $\widehat{\sigma_\Psi}$, and the term $(X_i/\sigma) \cdot \widehat{\sigma_\Psi}'$ can therefore not include variables outside the prefix $\Phi_* \cdot \widehat{\sigma_\Psi}$ of $\Phi \cdot \widehat{\sigma_\Psi}$.
In the case where $t' <^f |\Phi_* \cdot \widehat{\sigma_\Psi}|$, we consider the substitution $\widehat{\sigma_\Psi}' = \widehat{\sigma_\Psi}[i \mapsto t']$. In that case we obviously have $\Phi \cdot \widehat{\sigma_\Psi}' = \Phi'$, $t_T \cdot \widehat{\sigma_\Psi}' = t_T$, and also $t \cdot \widehat{\sigma_\Psi} = t'$. Also, $\widehat{\Psi}' = \mathsf{relevant}(\Psi;\ \Phi \vdash_p X_i/\sigma : t_T \cdot \sigma) = (\mathsf{relevant}(\Psi\lfloor_i;\ \Phi_* \vdash_p t_T : s), ?, \cdots, ?) \circ \mathsf{relevant}(\Psi;\ \Phi \vdash_p \sigma : \Phi_*) \circ (\Psi \widehat{@} i)$.
We need to show that $\bullet \vdash \widehat{\sigma_\Psi}' : \widehat{\Psi}'$. First, we have that $\mathsf{relevant}(\Psi;\ \Phi \vdash_p \sigma : \Phi^*) = \mathsf{relevant}(\Psi \vdash_p \Phi\ \mathrm{wf})$ since $\Phi^* \subseteq \Phi$. Second, we have that $\mathsf{relevant}(\Psi;\ \Phi \vdash_p t_T : s) = (\mathsf{relevant}(\Psi\lfloor_i;\ \Phi_* \vdash_p t_T : s), ?, \cdots, ?) \circ \mathsf{relevant}(\Psi \vdash_p \Phi\ \mathrm{wf})$. Thus we have that $\widehat{\Psi}' = \widehat{\Psi} \circ (\Psi \widehat{@} i)$. It is now trivial to see that indeed $\bullet \vdash \widehat{\sigma_\Psi}' : \widehat{\Psi}'$.
If $\widehat{\sigma_\Psi}.i = t_*$, then we split cases on whether $t_* = t'$ or not. If it is, then obviously $\widehat{\sigma_\Psi}$ is the desired unifying substitution for which all the desired properties hold. If it is not, then no substitution with the desired properties possibly exists, because it would violate the uniqueness assumption for $\widehat{\sigma_\Psi}$.

Case *(rest)* $\triangleright$ Similar techniques as above.

Part 1 By induction on the well-formedness derivation for Φ.

Case $\dfrac{}{\Psi \vdash_p \bullet \text{ wf}}$ $\triangleright$
Trivially, we either have $\Phi' = \bullet$, in which case unspec_Ψ is the unique substitution with the desired properties, or no substitution possibly exists.

Case $\dfrac{\Psi \vdash_p \Phi \text{ wf} \qquad \Psi;\, \Phi \vdash_p t : s}{\Psi \vdash_p (\Phi,\, t) \text{ wf}}$ $\triangleright$
We either have that $\Phi' = \Phi',\, t'$ or no substitution possibly exists. By induction hypothesis get $\widehat{\sigma_\Psi}$ such that $\Phi \cdot \widehat{\sigma_\Psi} = \Phi'$ and $\bullet \vdash \widehat{\sigma_\Psi} : \widehat{\Psi}$ with $\widehat{\Psi} = \mathsf{relevant}\,(\Psi \vdash_p \Phi \text{ wf})$. Now we use part 2 to either get a $\widehat{\sigma_\Psi}'$ which is obviously the substitution that we want, since $(\Phi,\, t) \cdot \widehat{\sigma_\Psi}' = \Phi',\, t'$ and $\mathsf{relevant}\,(\Psi \vdash_p (\Phi,\, t) \text{ wf}) = \mathsf{relevant}\,(\Psi;\, \Phi \vdash_p t : s)$; or we get the fact that no such substitution possibly exists. In that case, we again conclude that no substitution for the current case exists either, otherwise it would violate the induction hypothesis.

Case $\dfrac{\bullet \vdash_p \Phi \text{ wf} \qquad (\Psi).i = [\Phi]\,\mathsf{ctx}}{\Psi \vdash_p \Phi,\, X_i \text{ wf}}$ $\triangleright$
We either have $\Phi' = \Phi,\, \Phi''$, or no substitution possibly exists (since Φ does not depend on unification variables, so we always have $\Phi \cdot \widehat{\sigma_\Psi} = \Phi$). We now consider the substitution $\widehat{\sigma_\Psi} = \mathsf{unspec}_\Psi[i \mapsto [\Phi]\Phi'']$. We obviously have that $(\Phi,\, X_i) \cdot \widehat{\sigma_\Psi} = \Phi,\, \Phi''$, and also that $\bullet \vdash \widehat{\sigma_\Psi} : \widehat{\Psi}$ with $\widehat{\Psi} = \Psi \widehat{@} i = \mathsf{relevant}\,(\Psi \vdash_p \Phi,\, X_i \text{ wf})$. Thus this substitution has the desired properties.

Part 3 By induction on the typing for T.

Case $\dfrac{\Psi;\, \Phi \vdash_p t : t_T \qquad \Psi;\, \Phi \vdash t_T : s}{\Psi \vdash_p [\Phi]t : [\Phi]t_T}$ $\triangleright$
By inversion of typing for T' we have: $T' = [\Phi]t',\, \bullet;\, \Phi \vdash_p t' : t_T,\, \bullet;\, \Phi \vdash_p t_T : s$.
We obviously have $\widehat{\Psi} = \mathsf{relevant}\,(\Psi;\, \Phi \vdash_p t_T : s) = \mathsf{unspec}_\Psi$, and the substitution $\widehat{\sigma_\Psi} = \mathsf{unspec}_\Psi$ is the unique substitution such that $\bullet \vdash \widehat{\sigma_\Psi} : \widehat{\Psi}$, $\Phi \cdot \widehat{\sigma_\Psi} = \Phi$ and $t_T \cdot \widehat{\sigma_\Psi} = t_T$. We can thus use part 2 for attempting unification between t and t', yielding a $\widehat{\sigma_\Psi}'$ such that $\bullet \vdash \widehat{\sigma_\Psi}' : \widehat{\Psi}'$ with $\widehat{\Psi}' = \mathsf{relevant}\,(\Psi;\, \Phi \vdash_p t : t_T)$ and $t \cdot \widehat{\sigma_\Psi}' = t'$. We have that $\mathsf{relevant}\,(\Psi;\, \Phi \vdash_p t : t_T) = \mathsf{relevant}\,(\Psi \vdash_p T : K)$, thus $\widehat{\Psi}' = \Psi$ by assumption. From that we realize that $\widehat{\sigma_\Psi}'$ is a fully-specified substitution since $\bullet \vdash \widehat{\sigma_\Psi}' : \Psi$, and thus this is the substitution with the desired properties.
If unification between t and t' fails, it is trivial to see that no substitution with the desired substitution exists, otherwise it would lead directly to a contradiction.

Case $\dfrac{\Psi \vdash_p \Phi,\, \Phi' \text{ wf}}{\Psi \vdash_p [\Phi]\Phi' : [\Phi]\,\mathsf{ctx}}$ $\triangleright$
By inversion of typing for T' we have: $T' = [\Phi]\Phi'',\, \bullet \vdash_p \Phi,\, \Phi'' \text{ wf},\, \bullet \vdash_p \Phi \text{ wf}$. From part 1 we get a $\widehat{\sigma_\Psi}$ unifying $\Phi,\, \Phi'$ and $\Phi,\, \Phi''$, or the fact that no such $\widehat{\sigma_\Psi}$ exists. In the first case, as above, it is easy to see that this is the fully-specified substitution that we desire. In the second case, no suitable substitution exists, otherwise we are led directly to a contradiction.

The above proof is constructive. Its computational content is actually a unification algorithm for our patterns. We illustrate the algorithm below by giving its unification rules; notice that it follows the inductive

structure of the proof (and makes the same assumption about types-of-types being subderivations). If a derivation according to the following rules is not possible, the algorithm returns failure.

Definition D.30 (Unification algorithm) *We give the rules for the unification algorithm below.*

$\boxed{(\Psi \vdash_p T : K) \sim (\bullet \vdash_p T' : K) \triangleright \widehat{\sigma_\Psi}}$

$$\frac{(\Psi;\, \Phi \vdash_p t : t_T) \sim \left(\bullet;\, \Phi \vdash_p t' : t_T\right) \triangleleft \mathsf{unspec}_\Psi \triangleright \widehat{\sigma_\Psi}}{(\Psi \vdash_p [\Phi]t : [\Phi]t_T) \sim \left(\bullet \vdash_p [\Phi]t' : [\Phi]t'_T\right) \triangleright \widehat{\sigma_\Psi}} \qquad \frac{\left(\Psi \vdash_p \Phi,\, \Phi' \text{ wf}\right) \sim \left(\bullet \vdash_p \Phi,\, \Phi'' \text{ wf}\right) \triangleright \widehat{\sigma_\Psi}}{\left(\Psi \vdash_p [\Phi]\Phi' : [\Phi]\,\mathsf{ctx}\right) \sim \left(\bullet \vdash_p [\Phi]\Phi'' : [\Phi]\,\mathsf{ctx}\right) \triangleright \widehat{\sigma_\Psi}}$$

$\boxed{(\Psi \vdash_p \Phi \text{ wf}) \sim (\bullet \vdash \Phi' \text{ wf}) \triangleright \widehat{\sigma_\Psi}}$

$$(\Psi \vdash_p \bullet \text{ wf}) \sim (\bullet \vdash \bullet \text{ wf}) \triangleright \mathsf{unspec}_\Psi$$

$$\frac{(\Psi \vdash_p \Phi \text{ wf}) \sim \left(\bullet \vdash_p \Phi' \text{ wf}\right) \triangleright \widehat{\sigma_\Psi} \qquad (\Psi;\, \Phi \vdash_p t : s) \sim \left(\bullet;\, \Phi' \vdash t' : s\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_\Psi}'}{(\Psi \vdash_p (\Phi,\, t) \text{ wf}) \sim \left(\bullet \vdash (\Phi',\, t') \text{ wf}\right) \triangleright \widehat{\sigma_\Psi}'}$$

$$(\Psi \vdash_p \Phi,\, X_i \text{ wf}) \sim \left(\bullet \vdash \Phi,\, \Phi' \text{ wf}\right) \triangleright \mathsf{unspec}_\Psi[i \mapsto \Phi']$$

$\boxed{(\Psi;\, \Phi \vdash_p t : t_T) \sim (\bullet;\, \Phi' \vdash t' : t'_T) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_\Psi}'}$

$$(\Psi;\, \Phi \vdash_p c : t) \sim \left(\bullet;\, \Phi' \vdash c : t'\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_\Psi}' \qquad \left(\Psi;\, \Phi \vdash_p s : s'\right) \sim \left(\bullet;\, \Phi' \vdash s : s'\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_\Psi}'$$

$$\frac{\mathbf{I} \cdot \widehat{\sigma_\Psi} = \mathbf{I}'}{(\Psi;\, \Phi \vdash_p f_{\mathbf{I}} : t) \sim \left(\bullet;\, \Phi' \vdash f_{\mathbf{I}'} : t'\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_\Psi}}$$

$$\frac{\begin{array}{c}(\Psi;\, \Phi \vdash_p t_1 : s) \sim \left(\bullet;\, \Phi' \vdash t'_1 : s\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_\Psi}' \\ \left(\Psi;\, \Phi,\, t_1 \vdash_p \lceil t_2 \rceil_{|\Phi|} : s'\right) \sim \left(\bullet;\, \Phi',\, t'_1 \vdash_p \lceil t'_2 \rceil_{|\Phi'|} : s'\right) \triangleleft \widehat{\sigma_\Psi}' \triangleright \widehat{\sigma_\Psi}''\end{array}}{\left(\dfrac{\Psi;\, \Phi \vdash_p t_1 : s \qquad \Psi;\, \Phi,\, t_1 \vdash_p \lceil t_2 \rceil_{|\Phi|} : s' \qquad (s, s', s'') \in \mathcal{R}}{\Psi;\, \Phi \vdash_p \Pi(t_1).t_2 : s''}\right) \sim \left(\dfrac{\bullet;\, \Phi \vdash t'_1 : s \qquad \bullet;\, \Phi',\, t'_1 \vdash \lceil t'_2 \rceil_{|\Phi'|} : s' \qquad (s, s', s'') \in \mathcal{R}}{\bullet;\, \Phi' \vdash \Pi(t'_1).t'_2 : s''}\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_\Psi}''}$$

$$\frac{\left(\Psi;\, \Phi,\, t_1 \vdash_p \lceil t_2 \rceil_{|\Phi|} : t'\right) \sim \left(\bullet;\, \Phi',\, t'_1 \vdash_p \lceil t'_2 \rceil_{|\Phi'|} : t''\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_\Psi}'}{\left(\dfrac{\Psi;\, \Phi \vdash_p t_1 : s \qquad \Psi;\, \Phi,\, t_1 \vdash_p \lceil t_2 \rceil_{|\Phi|} : t' \qquad \Psi;\, \Phi \vdash_p \Pi(t_1).\, \lfloor t' \rfloor_{|\Phi|,\,.} : s'}{\Psi;\, \Phi \vdash_p \lambda(t_1).t_2 : \Pi(t_1).\, \lfloor t' \rfloor_{|\Phi|,\,.}}\right) \sim \left(\dfrac{\bullet;\, \Phi' \vdash t'_1 : s \qquad \bullet;\, \Phi',\, t'_1 \vdash \lceil t'_2 \rceil_{|\Phi'|} : t'' \qquad \bullet;\, \Phi' \vdash_p \Pi(t'_1).\, \lfloor t'' \rfloor_{|\Phi'|,\,.} : s'}{\bullet;\, \Phi' \vdash \lambda(t'_1).t'_2 : \Pi(t'_1).\, \lfloor t'' \rfloor_{|\Phi'|,\,.}}\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_\Psi}'}$$

$$\frac{\begin{array}{c}\left(\Psi;\ \Phi \vdash_p \Pi(t_a).t_b : s'\right) \sim \left(\bullet;\ \Phi \vdash_p \Pi(t'_a).t'_b : s'\right) \triangleleft \widehat{\sigma_\Psi}|_{\mathsf{relevant}(\Psi \vdash_p \Phi\ \mathrm{wf})} \triangleright \widehat{\sigma_\Psi}' \\ (\Psi;\ \Phi \vdash_p t_1 : \Pi(t_a).t_b) \sim \left(\bullet;\ \Phi' \vdash_p t'_1 : \Pi(t'_a).t'_b\right) \triangleleft \widehat{\sigma_\Psi}' \triangleright \widehat{\sigma_{\Psi 1}} \\ (\Psi;\ \Phi \vdash_p t_2 : t_a) \sim \left(\bullet;\ \Phi' \vdash_p t'_2 : t'_a\right) \triangleleft \widehat{\sigma_\Psi}'|_{\mathsf{relevant}(\Psi;\ \Phi \vdash_p t_a : s)} \triangleright \widehat{\sigma_{\Psi 2}}\end{array}}{\left(\dfrac{\dfrac{\dfrac{\Psi;\ \Phi \vdash_p t_a : s}{\Psi;\ \Phi \vdash_p \Pi(t_a).t_b : s'}}{\Psi;\ \Phi \vdash_p t_1 : \Pi(t_a).t_b} \qquad \Psi;\ \Phi \vdash_p t_2 : t_a}{\Psi;\ \Phi \vdash_p t_1\ t_2 : \lceil t_b \rceil_{|\Phi|} \cdot (\mathsf{id}_\Phi, t_2)}\right) \sim \left(\dfrac{\dfrac{\dfrac{\bullet;\ \Phi' \vdash_p t'_a : s}{\bullet;\ \Phi \vdash_p \Pi(t'_a).t'_b : s}}{\bullet;\ \Phi' \vdash_p t'_1 : \Pi(t'_a).t'_b} \qquad \bullet;\ \Phi' \vdash_p t'_2 : t'_a}{\bullet;\ \Phi' \vdash_p t'_1\ t'_2 : \lceil t'_b \rceil_{|\Phi'|} \cdot (\mathsf{id}_{\Phi'}, t'_2)}\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_{\Psi 1}} \circ \widehat{\sigma_{\Psi 2}}}$$

$$\frac{\begin{array}{c}(\Psi;\ \Phi \vdash_p t : \mathsf{Type}) \sim \left(\bullet;\ \Phi' \vdash_p t' : \mathsf{Type}\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_\Psi}' \\ (\Psi;\ \Phi \vdash_p t_1 : t) \sim \left(\bullet;\ \Phi' \vdash_p t'_1 : t'\right) \triangleleft \widehat{\sigma_\Psi}' \triangleright \widehat{\sigma_{\Psi 1}} \qquad (\Psi;\ \Phi \vdash_p t_2 : t) \sim \left(\bullet;\ \Phi' \vdash_p t'_2 : t'\right) \triangleleft \widehat{\sigma_\Psi}' \triangleright \widehat{\sigma_{\Psi 2}}\end{array}}{\left(\dfrac{\Psi;\ \Phi \vdash_p t_1 : t \qquad \Psi;\ \Phi \vdash_p t_2 : t}{\Psi;\ \Phi \vdash_p t : \mathsf{Type}}{\Psi;\ \Phi \vdash_p t_1 = t_2 : \mathsf{Prop}}\right) \sim \left(\dfrac{\bullet;\ \Phi \vdash_p t'_1 : t' \qquad \bullet;\ \Phi \vdash_p t'_2 : t'}{\bullet;\ \Phi \vdash_p t' : \mathsf{Type}}{\bullet;\ \Phi \vdash_p t'_1 = t'_2 : \mathsf{Prop}}\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_{\Psi 1}} \circ \widehat{\sigma_{\Psi 2}}}$$

$$\frac{\widehat{\sigma_\Psi}.i = ? \qquad \Psi.i = [\Phi^*]t_T \qquad t' <^f |\Phi^* \cdot \widehat{\sigma_\Psi}|}{(\Psi;\ \Phi \vdash_p X_i/\sigma : t_T \cdot \sigma) \sim \left(\bullet;\ \Phi' \vdash t' : t'_T\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_\Psi}[i \mapsto [\Phi^*]t']}$$

$$\frac{\widehat{\sigma_\Psi}.i = [\Phi^*]t \qquad t = t'}{(\Psi;\ \Phi \vdash_p X_i/\sigma : t_T) \sim \left(\bullet;\ \Phi' \vdash t' : t'_T\right) \triangleleft \widehat{\sigma_\Psi} \triangleright \widehat{\sigma_\Psi}}$$

Lemma D.31 *The above rules are algorithmic.*

Proved by the fact that they obey structural induction on the typing derivations, and are deterministic; non-covered cases signify the unification failure result.

By mimicking the unification proof above, we could show independently that the above algorithm is sound – that is, that the $\widehat{\sigma_\Psi}'$ it returns if it is successful is actually a substitution that makes t and t' unify (as well as Φ and Φ', along with t_T and t'_T) and is of the right type, provided that the assumptions about the input substitution $\widehat{\sigma_\Psi}$ do hold. Furthermore, we could show completeness, the fact that if the algorithm fails, no such substitution actually exists.

D.5 Computational language

Here we will refine our results for progress and preservation from the previous section, using the above results.

Definition D.32 *We refine the typing rule for pattern matching from definition C.4 as shown below.*

$$\frac{\begin{array}{c}\Psi \vdash T : K \qquad \Psi,\ K;\ \Gamma \vdash \lceil \tau \rceil_{|\Psi|,1} : \star \qquad \Psi \vdash_p \lceil \Psi' \rceil_{|\Psi|}\ \mathrm{wf} \\ \Psi,\ \lceil \Psi' \rceil_{|\Psi|} \vdash_p \lceil T' \rceil_{|\Psi|,|\Psi'|} : K \qquad \mathsf{unspec}_\Psi,\ \lceil \Psi' \rceil_{|\Psi|} \sqsubseteq \mathsf{relevant}\left(\Psi,\ \lceil \Psi' \rceil_{|\Psi|} \vdash_p \lceil T' \rceil_{|\Psi|,|\Psi'|} : K\right) \\ \Psi,\ \lceil \Psi' \rceil_{|\Psi|};\ \Sigma;\ \Gamma \vdash \lceil e' \rceil_{|\Psi|,|\Psi'|} : \lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\ \lceil T' \rceil_{|\Psi|,|\Psi'|})\end{array}}{\Psi;\ \Sigma;\ \Gamma \vdash \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi'.T' \mapsto e') : (\lceil \tau \rceil_{|\Psi|,1} \cdot (\mathsf{id}_\Psi,\ T)) + \mathsf{unit}}$$

Lemma D.33 (Substitution) *Adaptation of the substitution lemma from C.13.*

All the cases are entirely identical to the previous proof, with the exception of the pattern matching construct which has a new typing rule. In that case, proceed similarly as before, with the use of the lemmas D.2, D.3 and D.18 proved above.

Theorem D.34 (Preservation) *Adaptation of theorem C.16 to the new rules.*

All the cases are entirely identical to the previous proof, with the exception of the pattern matching construct. In that case, we need to explicitly allude to the fact that if $\bullet \vdash_p \lceil \Psi' \rceil_{|\Psi|}$ wf, then obviously also $\bullet \vdash \lceil \Psi' \rceil_{|\Psi|}$ wf. Similarly we have that $\lceil \Psi' \rceil_0 \vdash_p \lceil T' \rceil_{0,|\Psi'|} : K$ implies $\lceil \Psi' \rceil_0 \vdash \lceil T' \rceil_{0,|\Psi'|} : K$.

Theorem D.35 (Progress) *Adaptation of theorem E.11 to the new rules.*

Again the only case that needs adaptation is the pattern matching case. In that case, we first note that if $\bullet \vdash T : K$ (as we have here), we also have $\bullet \vdash_p T : K$. Then, we allude to the theorem 3 to split cases depending on whether a suitable σ_Ψ exists or not. In both cases, one step rule is applicable – if a unique σ_Ψ exists, then it has the desired properties for the first pattern matching step rule to work; if it does not, the second pattern matching step rule is applicable.

D.6 Sketch: practical pattern matching

The unification algorithm presented above requires full typing derivations for terms, something that is unrealistic to keep around as part of the runtime representation of terms. Here we will present an informal refinement of the above algorithm, that works on suitably annotated terms, instead of full typing derivations. The annotations are the minimal possible extra information needed to simulate the above algorithm.

Definition D.36 *We define a notion of annotated terms, for which we reuse the t syntactic class; it will be apparent from the context whether we mean a normal or an annotated term.*

$$t ::= c \mid s \mid f_{\mathbf{I}} \mid b_i \mid \lambda(t_1).t_2 \mid \Pi_s(t_1).t_2 \mid (t_1 : t)\, t_2 \mid t_1 =_t t_2 \mid X_i/\sigma$$

Lemma D.37 *1. If t is an unannotated term with $\bullet$, Ψ_u; $\Phi \vdash_p t : t'$ then there exists a derivation for Ψ_u; $\Phi \vdash_p t : t'$ where all terms are annotated terms.*

2. The inverse is also true.

These are trivial to prove by structural induction on the typing derivations.

Definition D.38 *The unification procedure is defined through the following judgement. It gets Ψ as a global parameter, which we omit here.*

$\boxed{(T) \sim (T')}$

$$\frac{(t) \sim \left(t'\right) \triangleleft \mathsf{unspec}_\Psi \triangleright \widehat{\sigma_\Psi}}{([\Phi]\, t) \sim \left([\Phi]\, t'\right) \triangleright \widehat{\sigma_\Psi}} \qquad \frac{\left(\Phi,\, \Phi'\right) \sim \left(\Phi,\, \Phi''\right) \triangleright \widehat{\sigma_\Psi}}{\left([\Phi]\, \Phi'\right) \sim \left([\Phi]\, \Phi''\right) \triangleright \widehat{\sigma_\Psi}}$$

$\boxed{(\Phi') \sim (\Phi'')}$

$$\frac{}{(\bullet) \sim (\bullet) \triangleright \mathsf{unspec}_{\widehat{\Psi}}} \qquad \frac{(\Phi) \sim \left(\Phi'\right) \triangleright \widehat{\sigma_\Psi} \qquad (t) \sim \left(t'\right) \triangleleft \widehat{\sigma_\Psi}\widehat{\sigma_\Psi}'}{((\Phi,\, t)) \sim \left((\Phi',\, t')\right) \triangleright \widehat{\sigma_\Psi}'}$$

$$\frac{}{(\Phi,\, X_i) \sim \left(\Phi, \Phi'\right) = \mathsf{unspec}_{\widehat{\Psi}}[i \mapsto [\Phi]\Phi']}$$

$\boxed{(t) \sim (t')}$

$$\frac{}{(c) \sim (c) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}} \qquad \frac{}{(s) \sim (s) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}} \qquad \frac{\mathbf{I} \cdot \widehat{\sigma_\Psi} = \mathbf{I}'}{(f_{\mathbf{I}}) \sim \left(f'_{\mathbf{I}}\right) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}}$$

$$\frac{s = s' \qquad (t_1) \sim \left(t'_1\right) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}' \qquad (\lceil t_2 \rceil) \sim \left(\lceil t'_2 \rceil\right) \lhd \widehat{\sigma_\Psi}' \rhd \widehat{\sigma_\Psi}''}{(\Pi_s(t_1).t_2) \sim \left(\Pi_{s'}(t'_1).t'_2\right) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}''} \qquad \frac{(\lceil t_2 \rceil) \sim \left(\lceil t'_2 \rceil\right) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}'}{(\lambda(t_1).t_2) \sim \left(\lambda(t'_1).t'_2\right) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}'}$$

$$\frac{(t) \sim \left(t'\right) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}' \qquad (t_1) \sim \left(t'_1\right) \lhd \widehat{\sigma_\Psi}' \rhd \widehat{\sigma_{\Psi 1}} \qquad (t_2) \sim \left(t'_2\right) \lhd \widehat{\sigma_\Psi}' \rhd \widehat{\sigma_{\Psi 2}} \qquad \widehat{\sigma_{\Psi 1}} \circ \widehat{\sigma_{\Psi 2}} = \widehat{\sigma_\Psi}''}{((t_1 : t)\ t_2) \sim \left((t'_1 : t')\ t'_2\right) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}''}$$

$$\frac{(t) \sim \left(t'\right) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}' \qquad (t_1) \sim \left(t'_1\right) \lhd \widehat{\sigma_\Psi}' \rhd \widehat{\sigma_{\Psi 1}} \qquad (t_2) \sim \left(t'_2\right) \lhd \widehat{\sigma_\Psi}' \rhd \widehat{\sigma_{\Psi 2}} \qquad \widehat{\sigma_{\Psi 1}} \circ \widehat{\sigma_{\Psi 2}} = \widehat{\sigma_\Psi}''}{(t_1 =_t t_2) \sim \left(t'_1 =_{t'} t'_2\right) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}''}$$

$$\frac{\widehat{\sigma_\Psi}.i = ? \qquad t' <^f |\sigma \cdot \widehat{\sigma_\Psi}|}{(X_i/\sigma) \sim \left(t'\right) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}[i \mapsto t']} \qquad \frac{\widehat{\sigma_\Psi}.i = t'}{(X_i/\sigma) \sim \left(t'\right) \lhd \widehat{\sigma_\Psi} \rhd \widehat{\sigma_\Psi}}$$

E. Simple staging support

Here we will add a light-weight staging support to the computational language. We extend the computational language as follows.

Definition E.1 *The syntax of the computational language is extended below.*

$$\begin{aligned} e &::= \cdots \mid \mathsf{letstatic}\ x\ =\ e\ \mathsf{in}\ e' \\ \Gamma &::= \cdots \mid \Gamma,\ x :_s \tau \end{aligned}$$

Definition E.2 *Freshening and binding for computational types and terms are extended as follows.*

$\boxed{\lceil e \rceil^M_{N,K}}$

$$\lceil \mathsf{letstatic}\ x\ =\ e\ \mathsf{in}\ e' \rceil^M_{N,K} \quad = \quad \mathsf{letstatic}\ x\ =\ \lceil e \rceil^M_{N,K}\ \mathsf{in}\ \lceil e' \rceil^{M+1}_{N,K}$$

$\boxed{\lfloor e \rfloor^M_{N,K}}$

$$\lfloor \mathsf{letstatic}\ x\ =\ e\ \mathsf{in}\ e' \rfloor^M_{N,K} \quad = \quad \mathsf{letstatic}\ x\ =\ \lfloor e \rfloor^M_{N,K}\ \mathsf{in}\ \lfloor e' \rfloor^{M+1}_{N,K}$$

Definition E.3 *Extension substitution application to computational-level types and terms.*

$\boxed{e \cdot \sigma_\Psi}$

$$\mathsf{letstatic}\ x\ =\ e\ \mathsf{in}\ e' \cdot \sigma_\Psi \quad = \quad \mathsf{letstatic}\ x\ =\ e \cdot \sigma_\Psi\ \mathsf{in}\ e' \cdot \sigma_\Psi$$

Definition E.4 *Limiting a context to the static types is defined as follows.*

$\boxed{\Gamma|_{\text{static}}}$

$$\begin{array}{lcl} \bullet|_{\text{static}} & = & \bullet \\ (\Gamma,\, x :_s t)|_{\text{static}} & = & \Gamma|_{\text{static}},\, x : t \\ (\Gamma,\, x : t)|_{\text{static}} & = & \Gamma|_{\text{static}} \\ (\Gamma,\, \alpha : k)|_{\text{static}} & = & \Gamma|_{\text{static}} \end{array}$$

Definition E.5 *The typing judgements for the computational language are extended below.*

$\boxed{\Psi;\, \Sigma;\, \Gamma \vdash e : \tau}$

$$\frac{\bullet;\, \Sigma;\, \Gamma|_{\text{static}} \vdash e : \tau \qquad \Psi;\, \Sigma;\, \Gamma, x :_s \tau \vdash e' : \tau}{\Psi;\, \Sigma;\, \Gamma \vdash \mathsf{letstatic}\, x \,=\, e\, \mathsf{in}\, e' : \tau} \qquad \frac{x :_s \tau \in \Gamma}{\Psi;\, \Sigma;\, \Gamma \vdash x : \tau}$$

Definition E.6 *Small-step operational semantics for the language are extended below.*

$$\begin{array}{rcl} e & ::= & \Lambda(K).e \mid e\ T \mid \mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ e \mid \mathsf{unpack}\ e\ (.)x.(e') \\ & & \mid () \mid \mathsf{error} \mid \lambda x : \tau.e \mid e\ e' \mid x \mid (e,\, e') \mid \mathsf{proj}_i\ e \mid \mathsf{inj}_i\ e \mid \mathsf{case}(e, x.e', x.e'') \mid \mathsf{fold}\ e \mid \mathsf{unfold}\ e \mid \mathsf{ref}\ e \\ & & \mid e := e' \mid\ !e \mid l \mid \Lambda\alpha : k.e \mid e\ \tau \mid \mathsf{fix}\ x : \tau.e \\ & & \mid \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi.T' \mapsto e') \mid \mathsf{letstatic}\, x \,=\, e\, \mathsf{in}\, e' \\ v & ::= & \Lambda(K).e_d \mid \mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ v \mid () \mid \lambda x : \tau.e_d \mid (v,\, v') \mid \mathsf{inj}_i\ v \mid \mathsf{fold}\ v \mid l \mid \Lambda\alpha : k.e_d \\ e_d & ::= & \Lambda(K).e_d \mid e_d\ T \mid \mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ e_d \mid \mathsf{unpack}\ e_d\ (.)x.(e'_d) \\ & & \mid () \mid \mathsf{error} \mid \lambda x : \tau.e_d \mid e_d\ e'_d \mid x \mid (e_d,\, e'_d) \mid \mathsf{proj}_i\ e_d \mid \mathsf{inj}_i\ e_d \mid \mathsf{case}(e_d, x.e'_d, x.e''_d) \mid \mathsf{fold}\ e_d \mid \mathsf{unfold}\ e_d \\ & & \mid \mathsf{ref}\ e_d \mid e_d := e'_d \mid\ !e_d \mid l \mid \Lambda\alpha : k.e_d \mid e_d\ \tau \mid \mathsf{fix}\ x : \tau.e_d \\ & & \mid \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi.T' \mapsto e'_d) \\ \mathcal{S} & ::= & \mathsf{letstatic}\, x \,=\, \bullet\, \mathsf{in}\, e' \mid \mathsf{letstatic}\, x \,=\, \mathcal{S}\, \mathsf{in}\, e' \mid \Lambda(K).\mathcal{S} \mid \lambda x : \tau.\mathcal{S} \mid \mathsf{unpack}\ e_d\ (.)x.(\mathcal{S}) \\ & & \mid \mathsf{case}(e_d, x.\mathcal{S}, x.e_2) \mid \mathsf{case}(e_d, x.e_d, x.\mathcal{S}) \mid \Lambda\alpha : k.\mathcal{S} \mid \mathsf{fix}\ x : \tau.\mathcal{S} \mid \mathsf{unify}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ (\Psi.T' \mapsto \mathcal{S}) \\ & & \mid \mathcal{E}_s[\mathcal{S}] \\ \mathcal{E}_s & ::= & \mathcal{E}_s\ T \mid \mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ \mathcal{E}_s \mid \mathsf{unpack}\ \mathcal{E}_s\ (.)x.(e') \mid \mathcal{E}_s\ e' \mid e_d\ \mathcal{E}_s \mid (\mathcal{E}_s,\, e) \mid (e_d,\, \mathcal{E}_s) \mid \mathsf{proj}_i\ \mathcal{E}_s \\ & & \mid \mathsf{inj}_i\ \mathcal{E}_s \mid \mathsf{case}(\mathcal{E}_s, x.e_1, x.e_2) \mid \mathsf{fold}\ \mathcal{E}_s \mid \mathsf{unfold}\ \mathcal{E}_s \mid \mathsf{ref}\ \mathcal{E}_s \mid \mathcal{E}_s := e' \mid e_d := \mathcal{E}_s \mid\ !\mathcal{E}_s \mid \mathcal{E}_s\ \tau \\ \mathcal{E} & ::= & \bullet \mid \mathcal{E}\ T \mid \mathsf{pack}\ T\ \mathsf{return}\ (.\tau)\ \mathsf{with}\ \mathcal{E} \mid \mathsf{unpack}\ \mathcal{E}\ (.)x.(e_d) \mid \mathcal{E}\ e_d \mid v\ \mathcal{E} \mid (\mathcal{E},\, e_d) \mid (v,\, \mathcal{E}) \mid \mathsf{proj}_i\ \mathcal{E} \mid \mathsf{inj}_i\ \mathcal{E} \\ & & \mid \mathsf{case}(\mathcal{E}, x.e'_d, x.e''_d) \mid \mathsf{fold}\ \mathcal{E} \mid \mathsf{unfold}\ \mathcal{E} \mid \mathsf{ref}\ \mathcal{E} \mid \mathcal{E} := e_d \mid v := \mathcal{E} \mid\ !\mathcal{E} \mid \mathcal{E}\ \tau \\ \mu & ::= & \bullet \mid \mu,\, l \mapsto v \end{array}$$

$\boxed{(\,\mu\,,\, e\,) \longrightarrow_s ((\,\mu\,,\, e'\,)|\mathsf{error})}$

$$\frac{(\,\mu\,,\, e_d\,) \longrightarrow (\,\mu'\,,\, e'_d\,)}{(\,\mu\,,\, \mathcal{S}[e_d]\,) \longrightarrow_s (\,\mu'\,,\, \mathcal{S}[e'_d]\,)} \qquad \frac{(\,\mu\,,\, e_d\,) \longrightarrow \mathsf{error}}{(\,\mu\,,\, \mathcal{S}[e_d]\,) \longrightarrow_s \mathsf{error}} \qquad (\,\mu\,,\, \mathcal{S}[\mathsf{letstatic}\, x \,=\, v\, \mathsf{in}\, e]\,) \longrightarrow_s (\,\mu\,,\, \mathcal{S}[e[v/x]]\,)$$

$$(\,\mu\,,\, \mathsf{letstatic}\, x \,=\, v\, \mathsf{in}\, e\,) \longrightarrow_s (\,\mu\,,\, e[v/x]\,)$$

Most lemmas are trivial to adapt. We adapt the substitution lemma for computational terms below.

Lemma E.7 (Substitution) *1. If* $\Psi,\, \Psi';\, \Gamma,\, \alpha' : k',\, \Gamma' \vdash \tau : k$ *and* $\Psi;\, \Gamma \vdash \tau' : k'$ *then* $\Psi,\, \Psi';\, \Gamma,\, \Gamma'[\tau'/\alpha'] \vdash \tau[\tau'/\alpha'] : k$.

2. *If* $\Psi,\ \Psi';\ \Sigma\ \Gamma,\ \alpha' : k',\ \Gamma' \vdash e : \tau$ *and* $\Psi;\ \Gamma \vdash \tau' : k'$ *then* $\Psi,\ \Psi';\ \Sigma;\ \Gamma,\ \Gamma'[\tau'/\alpha'] \vdash e[\tau'/\alpha'] : \tau[\tau'/\alpha']$.

3. *If* $\Psi,\ \Psi';\ \Sigma\ \Gamma,\ x' : \tau',\ \Gamma' \vdash e_d : \tau$ *and* $\Psi;\ \Sigma;\ \Gamma \vdash e'_d : \tau'$ *then* $\Psi,\ \Psi';\ \Sigma;\ \Gamma,\ \Gamma' \vdash e_d[e'_d/x'] : \tau$.

4. *If* $\Psi;\ \Gamma,\ x :_s \tau,\ \Gamma' \vdash e : \tau'$ *and* $\bullet;\ \Sigma;\ \bullet \vdash v : \tau$ *then* $\Psi;\ \Sigma;\ \Gamma,\ \Gamma' \vdash e[v/x] : \tau'$.

5. *If* $\Psi;\ \Gamma,\ x : \tau,\ \Gamma' \vdash e : \tau'$ *and* $\bullet;\ \Sigma;\ \bullet \vdash v : \tau$ *then* $\Psi;\ \Sigma;\ \Gamma,\ \Gamma' \vdash e[v/x] : \tau'$.

Easy proof by structural induction on the typing derivation for e. We prove the interesting cases below:

Part 3. **Case** $\dfrac{x :_s \tau \in \Gamma}{\Psi;\ \Sigma;\ \Gamma \vdash x : \tau}\ \triangleright$

We have that $e_d[e'_d/x] = e'_d$, and $\Psi;\ \Sigma;\ \Gamma \vdash e'_d : \tau$, which is the desired.

Case $\dfrac{\bullet;\ \Sigma;\ \Gamma|_{\text{static}} \vdash e : \tau \qquad \Psi;\ \Sigma;\ \Gamma,\ x :_s \tau \vdash e' : \tau'}{\Psi;\ \Sigma;\ \Gamma \vdash \mathsf{letstatic}\ x\ =\ e\ \mathsf{in}\ e' : \tau'}\ \triangleright$

Impossible case, because the theorem only has to do with e_d cases.

Part 4. Most cases are trivial. The only interesting case follows.

Case $\dfrac{\bullet;\ \Sigma;\ \Gamma|_{\text{static}},\ x : \tau,\ \Gamma'|_{\text{static}} \vdash e : \tau \qquad \Psi;\ \Sigma;\ \Gamma,\ x :_s \tau,\ \Gamma',\ x' :_s \tau'' \vdash e' : \tau'}{\Psi;\ \Sigma;\ \Gamma,\ x :_s \tau,\ \Gamma' \vdash \mathsf{letstatic}\ x'\ =\ e\ \mathsf{in}\ e' : \tau'}\ \triangleright$

We use part 5 for e to get that $\bullet;\ \Sigma;\ \Gamma|_{\text{static}},\ \Gamma'|_{\text{static}} \vdash e[v/x] : \tau$.
By induction hypothesis for e' we get $\Psi;\ \Sigma;\ \Gamma,\ \Gamma', x' :_s \tau'' \vdash e'[v/x] : \tau'$.
Thus using the same typing rule we get the desired result.

Part 5. Trivial by structural induction.

Lemma E.8 (Types of decompositions) *1. If* $\Psi;\ \Sigma;\ \Gamma \vdash \mathcal{S}[e] : \tau$ *with* $\Gamma|_{static} = \bullet$*, then there exists* τ' *such that* $\bullet;\ \Sigma;\ \bullet \vdash e : \tau'$ *and for all* e' *such that* $\bullet;\ \Sigma;\ \bullet \vdash e' : \tau'$*, we have that* $\Psi;\ \Sigma;\ \Gamma \vdash \mathcal{S}[e'] : \tau$.

2. *If* $\Psi;\ \Sigma;\ \Gamma \vdash \mathcal{E}_s[e] : \tau$ *then there exists* τ' *such that* $\Psi;\ \Sigma;\ \Gamma \vdash e : \tau'$ *and for all* e' *such that* $\Psi;\ \Sigma;\ \Gamma \vdash e' : \tau'$*, we have that* $\Psi;\ \Sigma;\ \Gamma \vdash \mathcal{E}_s[e'] : \tau$.

Part 1. By structural induction on $\mathcal{S}$.

Case $\mathcal{S} = \mathsf{letstatic}\ x\ =\ \bullet\ \mathsf{in}\ e'\ \triangleright$ By inversion of typing we get that $\bullet;\ \Sigma;\ \Gamma|_{\text{static}} \vdash e : \tau$. We have $\Gamma|_{\text{static}}$, thus we get $\bullet;\ \Sigma;\ \bullet \vdash e : \tau'$. Using the same typing rule we get the desired result for $\mathcal{S}[e']$.

Case $\mathcal{S} = \mathsf{letstatic}\ x\ =\ \mathcal{S}'\ \mathsf{in}\ e''\ \triangleright$ By inductive hypothesis for $\mathcal{S}'$ we get the desired directly.

Case $\mathcal{S} = \Lambda(K).\mathcal{S}'\ \triangleright$ We have that $\lceil \mathcal{S}'[e_d] \rceil = \mathcal{S}''[\lceil e \rceil]$ with $\mathcal{S}'' = \lceil \mathcal{S}'[\bullet] \rceil$. By inductive hypothesis for $\Psi,\ K;\ \Sigma;\ \Gamma \vdash \mathcal{S}''[\lceil e \rceil] : \tau$ we get that $\bullet;\ \Sigma;\ \bullet \vdash \lceil e \rceil : \tau'$. From this we directly get $\lceil e \rceil = e$, and the desired follows immediately (using the rest of the inductive hypothesis).

Case $\mathcal{S} = \mathcal{E}_s[\mathcal{S}]\ \triangleright$ We have that $\Psi;\ \Sigma;\ \Gamma \vdash \mathcal{E}_s[\mathcal{S}[e_d]] : \tau$. Using part 2 for $\mathcal{E}_s$ and $\mathcal{S}[e_d]$ we get that $\Psi;\ \Sigma;\ \Gamma \vdash \mathcal{S}[e_d] : \tau'$ for some τ' and also that for all e' such that $\Psi;\ \Sigma;\ \Gamma \vdash e : \tau'$, $\Psi;\ \Sigma;\ \Gamma \vdash \mathcal{E}_s[e'] : \tau$. Then using induction hypothesis we get a τ'' such that $\bullet;\ \Sigma;\ \bullet \vdash e_d : \tau''$. For this type, we also have that $\bullet;\ \Sigma;\ \bullet \vdash e'_d : \tau''$ implies $\Psi;\ \Sigma;\ \Gamma \vdash \mathcal{S}[e'_d] : \tau'$, which further implies $\Psi;\ \Sigma;\ \Gamma \vdash \mathcal{E}_s[\mathcal{S}[e'_d]] : \tau$.

The rest of the cases follow similar ideas.

Part 2. By induction on the structure of $\mathcal{E}_s$. In each case, use inversion of typing to get the type for e, and then use the same typing rule to get the derivation for $\mathcal{E}_s[e']$.

Theorem E.9 (Preservation) *1. If* $\bullet; \Sigma; \bullet \vdash e : \tau$, $\mu \sim \Sigma$, $(\mu, e) \longrightarrow_s (\mu', e')$ *then there exists* Σ' *such that* $\Sigma \subseteq \Sigma'$, $\mu' \sim \Sigma'$ *and* $\bullet; \Sigma'; \bullet \vdash e' : \tau$.

2. If $\bullet; \Sigma; \bullet \vdash e_d : \tau$, $\mu \sim \Sigma$, $(\mu, e_d) \longrightarrow_s (\mu', e'_d)$ *then there exists* Σ' *such that* $\Sigma \subseteq \Sigma'$, $\mu' \sim \Sigma'$ *and* $\bullet; \Sigma'; \bullet \vdash e'_d : \tau$.

Part 1 We proceed by induction on the derivation of $(\mu, e) \longrightarrow (\mu', e')$.

Case $\dfrac{(\mu, e_d) \longrightarrow (\mu', e'_d)}{(\mu, \mathcal{S}[e_d]) \longrightarrow_s (\mu', \mathcal{S}[e'_d])}$ $\triangleright$
Using the lemma E.8 we get $\bullet; \Sigma; \bullet \vdash e_d : \tau'$. Using part 2, we get that $\bullet; \Sigma; \bullet \vdash e'_d : \tau'$. Thus, using again the same lemma we get the desired.

Case $(\mu, \mathcal{S}[\mathsf{letstatic}\ x\ =\ v\ \mathsf{in}\ e]) \longrightarrow_s (\mu, \mathcal{S}[e[v/x]])$ $\triangleright$
Using the lemma E.8 we get $\bullet; \Sigma; \bullet \vdash \mathsf{letstatic}\ x\ =\ v\ \mathsf{in}\ e : \tau'$. By typing inversion we get that $\bullet; \Sigma; \bullet \vdash v : \tau''$, and also that $\bullet; \Sigma; x :_s \tau'' \vdash e : \tau'$. Using the substitution lemma E.7 we get the desired result.

The rest of the cases are trivial.

Part 2 Proceeds exactly as before, as e_d entirely matches the definition of expressions prior to the extension.

Theorem E.10 (Unique decomposition) *1. For every expression e, we have:*

(a) Either e is a dynamic expression e_d*, in which case there is no way to write* e_d *as* $\mathcal{S}[e']$ *for any* e'.

(b) Or there is a unique decomposition of e into $\mathcal{S}[e_d]$.

2. For every expression e_d*, we have:*

(a) Either it is a value v and the decomposition $v = \mathcal{E}[e]$ *implies* $\mathcal{E} = \bullet$ *and* $e = v$.

(b) Or, there is a unique decomposition of e_d *into* $e_d = \mathcal{E}[v]$.

Part 1. Proceed by induction on the structure of the expression e.

Case $\Lambda(K).e'$ $\triangleright$ By induction hypothesis on the structure of e'. If we have $e' = e_d$, then this is a dynamic expression already. In the other cases, we get a unique decomposition of e' into $\mathcal{S}'[e'']$. The original expression e can be uniquely decomposed using $\mathcal{S} = \Lambda(K).\mathcal{S}'$, with $e = \mathcal{S}[e'']$. This decomposition is unique because the outer frame is uniquely determined; if the inner frames or the expression filling the hole could be different, we would violate the uniqueness part of the decomposition returned by induction hypothesis.

Case $e'\ T$ $\triangleright$ By induction hypothesis we get that either $e' = e'_d$, or there is a unique decomposition of e' into $\mathcal{S}'[e'']$. In that case, e is uniquely decomposed using $\mathcal{S} = \mathcal{E}_s[\mathcal{S}']$ with $\mathcal{E}_s = \bullet\ T$, into $e = \mathcal{S}'[e'']\ T$.

Case $\mathsf{unpack}\ x\ (.)e'.(e'')$ $\triangleright$ By induction hypothesis on e'; if it is a dynamic expression, then by induction hypothesis on e''; if that too is a dynamic expression, then the original expression is too. Otherwise, use the unique decomposition of $e'' = \mathcal{S}'[e''']$ to get the unique decomposition $e = \mathsf{unpack}\ x\ (.)e_d.(\mathcal{S}'[e'''])$. If e' is not a dynamic expression, use the unique decomposition of $e' = \mathcal{S}''[e'''']$ to get the unique decomposition $e = \mathsf{unpack}\ x\ (.)\mathcal{S}''[e''''].(e'')$.

Case $\mathsf{letstatic}\ x\ =\ e'\ \mathsf{in}\ e''\ \triangleright$ By induction hypothesis on e'.
In the case that $e' = e_d$, then trivially we have the unique decomposition $e = (\mathsf{letstatic}\ x\ =\ \bullet\ \mathsf{in}\ e'')[e_d]$.
In the case where $e' = \mathcal{S}[e_d]$, we have the unique decomposition $e = (\mathsf{letstatic}\ x\ =\ \mathcal{S}\ \mathsf{in}\ e'')[e_d]$.

The rest of the cases are similar.

Part 2. Trivial by induction on the structure of the dynamic expression e_d.

Theorem E.11 (Progress) *1. If $\bullet; \Sigma; \bullet \vdash e : \tau$ and $\mu \sim \Sigma$, then either $\mu,\ e \longrightarrow_s \mathsf{error}$, or e is a dynamic expression e_d, or there exist μ' and e' such that $\mu,\ e \longrightarrow_s \mu',\ e'$.*

2. If $\bullet; \Sigma; \bullet \vdash e_d : \tau$ and $\mu \sim \Sigma$, then either $\mu,\ e_d \longrightarrow \mathsf{error}$, or e_d is a value v, or there exist μ' and e'_d such that $\mu,\ e_d \longrightarrow \mu',\ e'_d$.

Part 1 First, we use the unique decomposition lemma E.10, we get that either e is a dynamic expression, in which case we are done; or a decomposition into $\mathcal{S}[e'_d]$. In that case, we use the lemma E.8 and part 2 to get that either e'_d is a value or that some progress can be made: either by failing or getting a μ', e''_d, in which case we use the appropriate rule for $\longrightarrow_s$ either to fail or to progress to $\mu', \mathcal{S}[e''_d]$. If e'_d is a value, then we split cases depending on $\mathcal{S}$ – if it is simply $\mathsf{letstatic}\ x\ =\ \bullet\ \mathsf{in}\ e$ or it is nested. In both cases we make progress using the appropriate step rule.

Part 2 Identical as before.

F. Collapsing terms with extension variables into terms with normal variables

Definition F.1 *A decidable judgement for deciding whether a term t, a context Φ, etc. are collapsable to a normal logical term is given below.*

Intuitively, it defines as collapsable terms where all context Φ involved (even inside extension variable types) are subcontexts of a single context Φ'' (which is the result of the procedure), and all extension variables are used with identity substitutions of that context.

$\boxed{\mathsf{collapsible}\,(\Psi) \triangleleft \Phi' \triangleright \Phi''}$

$$\frac{}{\mathsf{collapsible}\,(\bullet) \triangleleft \Phi' \triangleright \Phi'} \qquad \frac{\mathsf{collapsible}\,(\Psi) \triangleleft \Phi' \triangleright \Phi'' \qquad \mathsf{collapsible}\,(K) \triangleleft \Phi'' \triangleright \Phi'''}{\mathsf{collapsible}\,(\Psi,\ K) \triangleleft \Phi' \triangleright \Phi'''}$$

$\boxed{\mathsf{collapsible}\,(K) \triangleleft \Phi' \triangleright \Phi''}$

$$\frac{K = T \qquad \mathsf{collapsible}\,(T) \triangleleft \Phi' \triangleright \Phi''}{\mathsf{collapsible}\,(K) \triangleleft \Phi' \triangleright \Phi''} \qquad \frac{\mathsf{collapsible}\,(\Phi) \triangleleft \Phi' \triangleright \Phi''}{\mathsf{collapsible}\,([\Phi]\,\mathsf{ctx}) \triangleleft \Phi' \triangleright \Phi''}$$

$\boxed{\mathsf{collapsible}\,(T) \triangleleft \Phi' \triangleright \Phi''}$

$$\frac{\mathsf{collapsible}\,(\Phi) \triangleleft \Phi' \triangleright \Phi'' \qquad \mathsf{collapsible}\,(t) \triangleleft \Phi''}{\mathsf{collapsible}\,([\Phi]\,t) \triangleleft \Phi' \triangleright \Phi''} \qquad \frac{\mathsf{collapsible}\,(\Phi_1,\ \Phi_2) \triangleleft \Phi' \triangleright \Phi''}{\mathsf{collapsible}\,([\Phi_1]\Phi_2) \triangleleft \Phi' \triangleright \Phi''}$$

$\boxed{\mathsf{collapsible}\,(\Phi) \lhd \Phi' \rhd \Phi''}$

$$\frac{}{\mathsf{collapsible}\,(\bullet) \lhd \Phi' \rhd \Phi'} \qquad \frac{\mathsf{collapsible}\,(\Phi) \lhd \Phi' \rhd \Phi'' \qquad \Phi = \Phi'' \qquad \mathsf{collapsible}\,(t) \lhd \Phi}{\mathsf{collapsible}\,(\Phi,\, t) \lhd \Phi' \rhd (\Phi,\, t)}$$

$$\frac{\mathsf{collapsible}\,(\Phi) \lhd \Phi' \rhd \Phi'' \qquad \Phi \subset \Phi'' \qquad \mathsf{collapsible}\,(t) \lhd \Phi''}{\mathsf{collapsible}\,(\Phi,\, t) \lhd \Phi' \rhd \Phi''} \qquad \frac{\mathsf{collapsible}\,(\Phi) \lhd \Phi' \rhd \Phi'' \qquad \Phi = \Phi''}{\mathsf{collapsible}\,(\Phi,\, X_i) \lhd \Phi' \rhd (\Phi,\, X_i)}$$

$$\frac{\mathsf{collapsible}\,(\Phi) \lhd \Phi' \rhd \Phi'' \qquad \Phi \subset \Phi''}{\mathsf{collapsible}\,(\Phi,\, X_i) \lhd \Phi' \rhd \Phi''}$$

$\boxed{\mathsf{collapsible}\,(t) \lhd \Phi'}$

$$\frac{}{\mathsf{collapsible}\,(s) \lhd \Phi'} \qquad \frac{}{\mathsf{collapsible}\,(c) \lhd \Phi'} \qquad \frac{}{\mathsf{collapsible}\,(f_{\mathbf{I}}) \lhd \Phi'}$$

$$\frac{\mathsf{collapsible}\,(t_1) \lhd \Phi' \qquad \mathsf{collapsible}\,(\lceil t_2 \rceil) \lhd \Phi'}{\mathsf{collapsible}\,(\lambda(t_1).t_2) \lhd \Phi'} \qquad \frac{\mathsf{collapsible}\,(t_1) \lhd \Phi' \qquad \mathsf{collapsible}\,(\lceil t_2 \rceil) \lhd \Phi'}{\mathsf{collapsible}\,(\Pi(t_1).t_2) \lhd \Phi'}$$

$$\frac{\mathsf{collapsible}\,(t_1) \lhd \Phi' \qquad \mathsf{collapsible}\,(t_2) \lhd \Phi'}{\mathsf{collapsible}\,(t1\ t_2) \lhd \Phi'} \qquad \frac{\mathsf{collapsible}\,(t_1) \lhd \Phi' \qquad \mathsf{collapsible}\,(t_2) \lhd \Phi'}{\mathsf{collapsible}\,(t1 = t_2) \lhd \Phi'}$$

$$\frac{\sigma \subseteq \mathsf{id}\Phi'}{\mathsf{collapsible}\,(X_i/\sigma) \lhd \Phi'}$$

$\boxed{\mathsf{collapsible}\,(\Psi \vdash T : K) \rhd \Phi'}$

$$\frac{\mathsf{collapsible}\,(\Psi) \lhd \bullet \rhd \Phi' \qquad \mathsf{collapsible}\,(K) \lhd \Phi' \rhd \Phi'' \qquad \mathsf{collapsible}\,(T) \lhd \Phi'' \rhd \Phi''}{\mathsf{collapsible}\,(\Psi \vdash T : K) \rhd \Phi''}$$

$\boxed{\mathsf{collapsible}\,(\Psi \vdash \Phi\ \mathrm{wf}) \rhd \Phi'}$

$$\frac{\mathsf{collapsible}\,(\Psi) \lhd \bullet \rhd \Phi' \qquad \mathsf{collapsible}\,(\Phi) \lhd \Phi' \rhd \Phi''}{\mathsf{collapsible}\,(\Psi \vdash \Phi\ \mathrm{wf}) \rhd \Phi''}$$

Lemma F.2 *1. If* $\mathsf{collapsible}\,(\Phi) \lhd \Phi' \rhd \Phi''$ *then either* $\Phi' \subseteq \Phi$ *and* $\Phi'' = \Phi$*, or* $\Phi \subseteq \Phi'$ *and* $\Phi'' = \Phi'$*.*

2. If $\mathsf{collapsible}\,(\Psi \vdash [\Phi]t : [\Phi]t_T) \rhd \Phi'$ *then* $\Phi \subseteq \Phi'$*.*

3. If $\mathsf{collapsible}\,(\Psi \vdash [\Phi_0]\,\Phi_1 : [\Phi_0]\,\Phi_1) \rhd \Phi'$ *then* $\Phi_0, \Phi_1 \subseteq \Phi'$*.*

Trivial by structural induction.

Lemma F.3 *1. If* $\mathsf{collapsible}\,(\Psi) \lhd \bullet \rhd \Phi$ *and* $\Phi \subseteq \Phi'$ *then* $\mathsf{collapsible}\,(\Psi) \lhd \Phi' \rhd \Phi'$*.*

2. If $\mathsf{collapsible}\,(K) \lhd \bullet \rhd \Phi$ *and* $\Phi \subseteq \Phi'$ *then* $\mathsf{collapsible}\,(K) \lhd \Phi' \rhd \Phi'$*.*

3. If $\mathsf{collapsible}\,(T) \lhd \bullet \rhd \Phi$ *and* $\Phi \subseteq \Phi'$ *then* $\mathsf{collapsible}\,(T) \lhd \Phi' \rhd \Phi'$*.*

4. If $\mathsf{collapsible}\,(\Phi_0) \lhd \bullet \rhd \Phi$ *and* $\Phi \subseteq \Phi'$ *then* $\mathsf{collapsible}\,(\Phi_0) \lhd \Phi' \rhd \Phi'$*.*

Trivial by structural induction on the collapsing relation.

Lemma F.4 *If* $\vdash \Psi$ *wf and* $\mathsf{collapsible}\,(\Psi) \lhd \bullet \rhd \Phi^0$*, then there exist* Ψ^1*,* σ^1_Ψ*,* σ^2_Ψ*,* Φ^1*,* σ^1 *and* σ^{-1} *such that:*
$\Psi^1 \vdash \sigma^1_\Psi : \Psi$*,*
$\bullet \vdash \sigma^2_\Psi : \Psi^1$*,*
$\Psi^1 \vdash \Phi^1$ *wf,*
$\Psi^1;\, \Phi^1 \vdash \sigma^1 : \Phi^0 \cdot \sigma^1_\Psi$*,*
$\Psi;\, \Phi^0 \vdash \sigma^{-1} : \Phi^1 \cdot \sigma^2_\Psi$*,*
for all t *such that* $\Psi;\, \Phi^0 \vdash t : t'$*, we have* $t \cdot \sigma^1_\Psi \cdot \sigma^1 \cdot \sigma^2_\Psi \cdot \sigma^{-1} = t$*, and all members of* Ψ^1 *are of the form* $[\Phi^*]t$ *where* $\Phi^* \subseteq \Phi^1$*.*

By induction on the derivation of the relation $\mathsf{collapsible}\,(\Psi) \lhd \bullet \rhd \Phi^0$.

Case $\Psi = \bullet\ \rhd$
We choose $\Psi^1 = \bullet$; $\sigma^1_\Psi = \sigma^2_\Psi = \bullet$; $\Phi^1 = \bullet$; $\sigma^1 = \bullet$; $\sigma^{-1} = \bullet$ $\Phi^1 = \bullet$ and the desired trivially hold.

Case $\Psi = \Psi',\ [\Phi]\,\mathsf{ctx}\ \rhd$
From the collapsable relation, we get: $\mathsf{collapsible}\,(\Psi') \lhd \bullet \rhd \Phi'^0$, $\mathsf{collapsible}\,([\Phi]\,\mathsf{ctx}) \lhd \Phi'^0 \rhd \Phi^0$. By induction hypothesis for Ψ', get:
$\Psi'^1 \vdash \sigma'^1_\Psi : \Psi'$,
$\bullet \vdash \sigma'^2_\Psi : \Psi'^1$,
$\Psi'^1 \vdash \Phi'^1$ wf,
$\Psi'^1;\, \Phi'^1 \vdash \sigma'^1 : \Phi'^0 \cdot \sigma'^1_\Psi$,
$\Psi';\, \Phi'^0 \vdash \sigma'^{-1} : \Phi'^1 \cdot \sigma'^2_\Psi$,
for all t such that $\Psi';\, \Phi'^0 \vdash t : t'$, we have $t \cdot \sigma'^1_\Psi \cdot \sigma'^1 \cdot \sigma'^2_\Psi \cdot \sigma'^{-1} = t$, and all members of Ψ'^1 are of the form $[\Phi^*]t$ where $\Phi^* \subseteq \Phi'^1$.
By inversion of typing for $[\Phi]\,\mathsf{ctx}$ we get that $\Psi' \vdash \Phi$ wf.
We fix $\sigma^1_\Psi = \sigma'^1_\Psi,\ [\Phi'^0 \cdot \sigma'^1_\Psi]$ which is a valid choice as long as we select Ψ^1 so that $\Psi'^1 \subseteq \Psi^1$. This substitution has correct type by taking into account the substitution lemma for Φ'^0 and σ'^1_Ψ.
For choosing the rest, we proceed by induction on the derivation of $\Phi'^0 \subseteq \Phi^0$.
If $\Phi^0 = \Phi'^0$, then:
We have $\Phi \subseteq \Phi'^0$ because of the previous lemma.
Choose $\Psi^1 = \Psi'^1$; $\sigma^2_\Psi = \sigma'^2_\Psi$; $\Phi^1 = \Phi'^1$; $\sigma^1 = \sigma'^1$; $\sigma^{-1} = \sigma'^{-1}$.
Everything holds trivially, other than σ^1_Ψ typing. This too is easy to prove by taking into account the substitution lemma for Φ and σ'^1_Ψ. Also, σ'^{-1} typing uses extension variable weakening. Last, for the cancellation part, terms that are typed under Ψ are also typed under Ψ' so this part is trivial too.
If $\Phi^0 = \Phi'^0,\ t$, then: (here we abuse things slightly – by identifying the context and substitutions from induction hypothesis with the ones we already have: their properties are the same for the new Φ'^0)
We have $\Phi = \Phi^0 = \Phi'^0,\ t$ because of the previous lemma (Φ^0 is not Φ'^0 thus $\Phi^0 = \Phi$).
First, choose $\Phi^1 = \Phi'^1,\ t \cdot \sigma'^1_\Psi \cdot \sigma'^1$. This is a valid choice, because $\Psi';\, \Phi'^0 \vdash t : s$; by applying σ'^1_Ψ we get $\Psi'^1;\, \Phi'^0 \cdot \sigma'^1_\Psi \vdash t \cdot \sigma'^1_\Psi : s$; by applying σ'^1 we get $\Psi'^1;\, \Phi'^1 \vdash t \cdot \sigma'^1_\Psi \cdot \sigma'^1 : s$.
Thus $\Psi'^1 \vdash \Phi'^1,\ t \cdot \sigma'^1_\Psi \cdot \sigma'^1$ wf (and the Ψ^1 we will choose is supercontext of Ψ'^1).
Now, choose $\Psi^1 = \Psi'^1,\ [\Phi_1]\, t \cdot \sigma'^1_\Psi \cdot \sigma'^1$. This is well-formed because of what we proved above about the substituted t, taking weakening into account. Also, the condition for the contexts in Ψ^1 being subcontexts of Φ^1 obviously holds.

Choose $\sigma^2_\Psi = \sigma'^2_\Psi,\ [\Phi_1] f_{|\Phi'^1|}$. We have $\bullet \vdash \sigma^2_\Psi : \Psi^1$ directly by our construction.
Choose $\sigma^1 = \sigma'^1,\ f_{|\Phi'^1|}$. We have that this latter term can be typed as $\Psi^1;\ \Phi^1 \vdash f_{|\Phi'^1|} : t \cdot \sigma'^1_\Psi \cdot \sigma'^1$, and thus we have $\Psi^1;\ \Phi^1 \vdash \sigma^1 : \Phi'^0 \cdot \sigma'^1_\Psi,\ t \cdot \sigma'^1_\Psi$.
Choose $\sigma^{-1} = \sigma'^{-1},\ f_{|\Phi'^0|}$, which is typed correctly since $t \cdot \sigma'^1_\Psi \cdot \sigma'^1 \cdot \sigma'^2_\Psi \cdot \sigma^{-1} = t$. Last, assume $\Psi;\ \Phi'^0,\ t \vdash t_* : t'_*$. We prove $t_* \cdot \sigma^1_\Psi \cdot \sigma^1 \cdot \sigma^2_\Psi \cdot \sigma^{-1} = t_*$.

> First, t_* is also typed under Ψ' because t_* cannot use the newly-introduced variable directly (even in the case where it would be part of Φ_0, there's still no extension variable that has $X_{|\Psi'|}$ in its context).
> Thus it suffices to prove $t_* \cdot \sigma'^1_\Psi \cdot \sigma^1 \cdot \sigma^2_\Psi \cdot \sigma^{-1} = t_*$.
> Then proceed by structural induction on t_*. The only interesting case occurs when $t_* = f_{|\Phi'^0|}$, in which case we have:
> $f_{|\Phi'^0|} \cdot \sigma'^1_\Psi \cdot \sigma^1 \cdot \sigma^2_\Psi \cdot \sigma^{-1} = f_{|\Phi'^0 \cdot \sigma'^1_\Psi|} \cdot \sigma^1 \cdot \sigma^2_\Psi \cdot \sigma^{-1} = f_{|\Phi'^1|} \cdot \sigma^2_\Psi \cdot \sigma^{-1} = f_{|\Phi'^1 \cdot \sigma^2_\Psi|} \cdot \sigma^{-1} = f_{|\Phi'^0|}$

If $\Phi^0 = \Phi'^0,\ X_i$:
By well-formedness inversion we get that $\Psi.i = [\Phi_*]\,\mathsf{ctx}$, and by repeated inversions of the collapsable relation we get $\Phi_* \subseteq \Phi'^0$.
Choose $\Phi^1 = \Phi'^1\ ;\ \Psi^1 = \Psi'^1\ ;\ \sigma^2_\Psi = \sigma'^2_\Psi;\ \sigma^1 = \sigma'^1;\ \sigma^{-1} = \sigma'^{-1}$.
Most desiderata are trivial. For σ^1, note that $(\Phi'^1,\ X_i) \cdot \sigma'^1_\Psi = \Phi'^1 \cdot \sigma'^1_\Psi$ since by construction we have that σ'^1_Ψ always substitutes parametric contexts by the empty context.
For cancellation, we need to prove that for all t such that $\Psi;\ \Phi'^0,\ X_i \vdash t_* : t'_*$, we have $t_* \cdot \sigma^1_\Psi \cdot \sigma^1 \cdot \sigma^2_\Psi \cdot \sigma^{-1} = t_*$. This is proved directly by noticing that t_* is typed also under Ψ' (if X_i was the just-introduced variable, it wouldn't be able to refer to itself).

Case $\Psi = \Psi',\ [\Phi] t\ \triangleright$
From the collapsable relation, we get: $\mathsf{collapsible}\,(\Psi') \triangleleft \bullet \triangleright \Phi'^0$, $\mathsf{collapsible}\,(\Phi) \triangleleft \Phi'^0 \triangleright \Phi^0$, $\mathsf{collapsible}\,(t) \triangleleft \Phi^0$. By induction hypothesis for Ψ', get:
$\Psi'^1 \vdash \sigma'^1_\Psi : \Psi'$,
$\bullet \vdash \sigma'^2_\Psi : \Psi'^1$,
$\Psi'^1 \vdash \Phi'^1$ wf,
$\Psi'^1;\ \Phi'^1 \vdash \sigma'^1 : \Phi'^0 \cdot \sigma'^1_\Psi$,
$\Psi';\ \Phi'^0 \vdash \sigma'^{-1} : \Phi'^1 \cdot \sigma'^2_\Psi$,
for all t such that $\Psi';\ \Phi'^0 \vdash t : t'$, we have $t \cdot \sigma'^1_\Psi \cdot \sigma'^1 \cdot \sigma'^2_\Psi \cdot \sigma'^{-1} = t$, and all members of Ψ'^1 are of the form $[\Phi^*] t$ where $\Phi^* \subseteq \Phi'^1$.
Also from typing inversion we get: $\Psi' \vdash \Phi$ wf and $\Psi';\ \Phi \vdash t : s$.
We proceed similarly as in the previous case, by induction on $\Phi'^0 \subseteq \Phi^0$, in order to redefine $\Psi'^1, \sigma'^1_\Psi, \sigma'^2_\Psi, \Phi'^1, \sigma'^1, \sigma^{-1}$ with the properties:
$\Psi'^1 \vdash \sigma'^1_\Psi : \Psi'$,
$\bullet \vdash \sigma'^2_\Psi : \Psi'^1$,
$\Psi'^1 \vdash \Phi'^1$ wf,
$\Psi'^1;\ \Phi'^1 \vdash \sigma'^1 : \Phi^0 \cdot \sigma'^1_\Psi$,
$\Psi';\ \Phi^0 \vdash \sigma'^{-1} : \Phi'^1 \cdot \sigma'^2_\Psi$,
for all t such that $\Psi';\ \Phi^0 \vdash t : t'$, we have $t \cdot \sigma'^1_\Psi \cdot \sigma'^1 \cdot \sigma'^2_\Psi \cdot \sigma'^{-1} = t$, and all members of Ψ'^1 are of the form $[\Phi^*] t$ where $\Phi^* \subseteq \Phi'^1$.
Now we have $\Phi \subseteq \Phi^0$ thus $\Psi';\ \Phi^0 \vdash t : s$.
By applying σ'^1_Ψ and then σ'^1 to t we get $\Psi'^1;\ \Phi'^1 \vdash t \cdot \sigma'^1_\Psi \cdot \sigma'^1 : s$. We can now choose $\Phi^1 = \Phi'^1,\ t \cdot \sigma'^1_\Psi \cdot \sigma'^1$.
Choose $\Psi^1 = \Psi'^1,\ [\Phi^1] t \cdot \sigma'^1_\Psi \cdot \sigma'^1$. It is obviously well-formed.
Now, will choose σ^1_Ψ:

Need to choose t^1 such that $\Psi^1;\ \Phi\cdot\sigma'^1_\Psi \vdash t^1 : t\cdot\sigma'^1_\Psi$.
Assuming $t^1 = X_{|\Phi'^1|}/\sigma$, we need $t\cdot\sigma'^1_\Psi\cdot\sigma'^1\cdot\sigma = t\cdot\sigma'^1_\Psi$ and $\Psi^1;\ \Phi\cdot\sigma'^1_\Psi \vdash \sigma : \Phi^1$.
Thus, what we require is the inverse of σ^1. By construction, there exists such a σ, because σ^1 is just a variable renaming. (Note that this is different from σ^{-1}.)
Therefore, set $\sigma^1_\Psi = \sigma'^1_\Psi,\ [\Phi]X_{|\Phi'^1|}/\sigma$, which has the desirable properties.
Choose $\sigma^2_\Psi = \sigma'^2_\Psi,\ [\Phi^1]f_{|\Phi'^1|}$. We trivially have $\bullet \vdash \sigma^2_\Psi : \Psi^1$.
Choose $\sigma^1 = \sigma'^1$, with typing holding obviously.
Choose $\sigma^{-1} = \sigma'^{-1},\ X_{|\Psi'|}/\mathsf{id}\Phi$. Consider the cancellation fact; typing is then possible.
It remains to prove that for all t_* such that $\Psi', [\Phi]t;\ \Phi^0 \vdash t_* : t'_*$, we have $t\cdot\sigma^1_\Psi\cdot\sigma^1\cdot\sigma^2_\Psi\cdot\sigma^{-1} = t$.
This is done by structural induction on t_*, with the interesting case being $t_* = X_{|\Psi'|}/\sigma_*$. By inversion of collapsable relation, we get that $\sigma_* = \mathsf{id}\Phi$.
Thus $(X_{|\Psi'|}/\mathsf{id}\Phi)\cdot\sigma^1_\Psi\cdot\sigma^1\cdot\sigma^2_\Psi\cdot\sigma^{-1} = (X_{|\Phi'^1|}/\sigma)\cdot(\mathsf{id}\Phi\cdot\sigma^1_\Psi)\cdot\sigma^1\cdot\sigma^2_\Psi\cdot\sigma^{-1} = (X_{|\Phi'^1|}/\sigma)\cdot(\mathsf{id}\Phi\cdot\sigma^1_\Psi)\cdot\sigma^1\cdot\sigma^2_\Psi\cdot\sigma^{-1} = (X_{|\Phi'^1|}/\sigma)\cdot\sigma^1\cdot\sigma^2_\Psi\cdot\sigma^{-1} = (X_{|\Phi'^1|}/(\sigma\cdot\sigma^1))\cdot\sigma^2_\Psi\cdot\sigma^{-1} = (X_{|\Phi'^1|}/(\mathsf{id}\Phi^1))\cdot\sigma^2_\Psi\cdot\sigma^{-1} = (f_{|\Phi'^1|}\cdot(\mathsf{id}\Phi^1\cdot\sigma^2_\Psi))\cdot\sigma^{-1} = (f_{|\Phi'^1|}\cdot\mathsf{id}\Phi^1\cdot\sigma^2_\Psi)\cdot\sigma^{-1} = f_{|\Phi'^1|\cdot\sigma^2_\Psi|}\cdot\sigma^{-1} = X_{|\Psi'|}/\mathsf{id}\Phi$.

Theorem F.5 *If $\Psi \vdash [\Phi]t : [\Phi]t_T$ and $\mathsf{collapsible}(\Psi \vdash [\Phi]t : [\Phi]t_T) = \Phi_*$, then there exist Φ', t', t'_T and σ such that $\bullet \vdash \Phi'$ wf, $\bullet \vdash [\Phi']t' : [\Phi']t'_T$, $\Psi;\ \Phi \vdash \sigma : \Phi'$, $t'\cdot\sigma = t$ and $t'_T\cdot\sigma = t_T$.*

Easy to prove using above lemma. Set $\Phi' = \Phi^1\cdot\sigma^2_\Psi$, $t' = t\cdot\sigma^1_\Psi\cdot\sigma^1\cdot\sigma^2_\Psi$, $t'_T = t_T\cdot\sigma^1_\Psi\cdot\sigma^1\cdot\sigma^2_\Psi$, and also set $\sigma = \sigma^{-1}$.