A

# Affine Refinement Types for Secure Distributed Programming (Long Version)

MICHELE BUGLIESI and STEFANO CALZAVARA, Università Ca' Foscari Venezia
FABIENNE EIGNER and MATTEO MAFFEI, CISPA, Saarland University

Recent research has shown that it is possible to leverage general-purpose theorem proving techniques to develop powerful type systems for the verification of a wide range of security properties on application code. Although successful in many respects, these type systems fall short of capturing resource-conscious properties that are crucial in large classes of modern distributed applications. In this paper, we propose the first type system that statically enforces the safety of cryptographic protocol implementations with respect to authorization policies expressed in affine logic. Our type system draws on a novel notion of "exponential serialization" of affine formulas, a general technique to protect affine formulas from the effect of duplication. This technique allows to formulate an expressive logical encoding of the authentication mechanisms underpinning distributed resource-aware authorization policies. We discuss the effectiveness of our approach on two case studies: the EPMO e-commerce protocol and the Kerberos authentication protocol. We finally devise a sound and complete type-checking algorithm, which is the key to achieving an efficient implementation of our analysis technique.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification; F.3.1 [**Theory of Computation**]: Specifying and Verifying and Reasoning about Programs

General Terms: Security, Theory, Verification

Additional Key Words and Phrases: Substructural logics, Type systems, Analysis of security protocols

**ACM Reference Format:**
Michele Bugliesi, Stefano Calzavara, Fabienne Eigner and Matteo Maffei. 2014. Affine Refinement Types for Secure Distributed Programming. *ACM Trans. Program. Lang. Syst.* V, N, Article A (January YYYY), 72 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

## 1. INTRODUCTION

Verifying the security of modern distributed applications is an important and complex challenge, which has attracted the interest of a growing research community audience over the last decade. Recent research has shown that it is possible to leverage general-purpose theorem proving techniques to develop powerful type systems for the verification of a wide range of security properties on application code, thus narrowing the gap between the formal model designed for the analysis and the actual implementation of the protocols [Bengtson et al. 2011; Backes et al. 2011; Swamy et al. 2011]. The integration between type systems and theorem proving is achieved by resorting to a form of dependent types, known as *refinement* types. A refinement type $\{x : T \mid F(x)\}$ qualifies the structural information of the type $T$ with a property specified by the logical formula $F$: a value $M$ of this type is a value of type $T$ such that $F(M)$ holds true.

...


DOI:http://dx.doi.org/10.1145/0000000.0000000

Authorization systems based on refinement types use the refinement formulas to express (and gain static control of) the credentials associated with the data and the cryptographic keys involved in the authorization checks. Clearly, the expressiveness of the resulting analysis hinges on the choice of the underlying logic, and indeed several logics have been proposed for the specification and verification of security properties [Chapin et al. 2008]. A number of proposals have thus set logic *parametricity* as a design goal, to gain modularity and scalability of the resulting systems. Though logic parametricity is in principle a sound and wise design choice, current attempts in this direction draw primarily (if not exclusively) on classical (or intuitionistic) logical frameworks. That, in turn, is a choice that makes the resulting systems largely ineffective on large classes of resource-aware authorization policies, such as those based on consumable credentials, or predicating over access counts and/or usage bounds.

The natural choice for expressing and reasoning about such classes of policies are instead *substructural* logics, such as linear and affine logic [Girard 1995; Troelstra 1992]. On the other hand, integrating substructural logics with existing refinement type systems for distributed authorization is challenging, as one must build safeguards against the ability of an attacker to duplicate the data exchanged over the network, and correspondingly duplicate the associated credentials, thus undermining their bounded nature [Bugliesi et al. 2011].

*Contributions.* In this paper, we present an *affine refinement type system* for RCF [Bengtson et al. 2011], a concurrent $\lambda$-calculus which can be directly mapped to a large subset of a real functional programming language like F#. The type system guarantees that well-typed programs comply with any given authorization policy expressed in affine logic, even in the presence of an active opponent.

This type system draws on the novel concept of *exponential serialization*, a general technique to protect affine formulas from the effect of duplication. This technique makes it possible to factor the authorization-relevant invariants of the analysis out of the type system, and to characterize them directly as proof obligations for the underlying affine logical system. This leads to a rather general and modular design of our proposal, and sheds new light on the logical foundations of standard cryptographic patterns underpinning distributed authorization frameworks. Furthermore, the concept of serialization enhances the expressiveness of the type system, capturing programming patterns out of the scope of many substructural type systems.

The clean separation between typing and logical entailment has the additional advantage of enabling the formulation of an algorithmic version of our system, in which the non-deterministic proof search distinctive of substructural type systems can be dispensed with. Intuitively, we can shift all the burden related to substructural resource management into a single proof obligation to be discharged to an external theorem prover. This proof obligation can be efficiently generated from a program in a syntax-directed way: this is the key to achieve a practical implementation of our framework.

We show the effectiveness of our approach on two case studies, namely the *EPMO* e-commerce protocol [Guttman et al. 2004] and the *Kerberos* authentication protocol [Steiner et al. 1988]. For both case studies we discuss the advantages in expressiveness enabled by the adoption of an underlying substructural logic.

*Structure of the paper.* Section 2 overviews the challenges and the most important aspects of our theory on a simple example. Section 3 reviews intuitionistic affine logic. Section 4 presents the meta-theory of exponential serialization. Section 5 reviews RCF and defines our notion of safety. Section 6 outlines the type system. Section 7 discusses encodings of network communication and our treatment of formal cryptography. Sections 8-9 present the case studies. Section 10 discusses the algorithmic formulation of our type system. Section 11 overviews the related work. Section 12 concludes.

Proofs are provided in the appendixes: Appendix A establishes the soundness of exponential serialization; Appendix B details a soundness proof for our type system; Appendix C provides proofs of the soundness and completeness of the algorithmic type system.

*Unpublished content.* The present work extends and revises a conference paper published at POST 2013 [Bugliesi et al. 2013], which received the EATCS award for the best theory paper at ETAPS. In this extended version we present full details of the formalization, including a complete presentation of the type system, its algorithmic variant, and complete soundness proofs for our main results: all this material was not published before, due to space constraints. Moreover, the Kerberos case study in Section 9 is new and required us to define an encoding of "self-dependent key types" in our type system, which we believe to be of independent interest.

## 2. OVERVIEW OF THE FRAMEWORK

Our protocol specification language is an affine variant of RCF, a concurrent $\lambda$-calculus with message passing and refinement types originally introduced in [Bengtson et al. 2011]. We anticipate that RCF is very expressive and can be mapped to a large subset of F#. For better readability, in the examples we use F#-like syntax with polymorphic types: our theoretical framework lacks full-fledged polymorphism, but that can be recovered by duplicating definitions at multiple monomorphic types when needed.

### 2.1. Protocol verification with (affine) refinement types

Verifying distributed authorization protocols with refinement types presupposes that protocols be annotated with security *assumptions* and *assertions*. The former are formulas that are assumed to hold at a given point in time, and they are employed to specify authorization policies and to encode the credentials available to request authorization. In contrast, assertions act as guards defining the properties to be entailed by the assumptions and the underlying policy, to grant authorization [Fournet et al. 2005; 2007; Bengtson et al. 2011].

An example will help in making the discussion concrete. We introduce a system to place and ship orders in a distributed online service governed by a simple authorization policy, establishing that an order can be cleared for shipping to a user only if that user has indeed placed the order. For example, we could start by assuming the authorization policy encoded by the first-order formula: $\mathcal{P} \triangleq \forall x, y.(\mathsf{Order}(x, y) \Rightarrow \mathsf{Ship}(x, y))$. The security-annotated code corresponding to the online service scenario is given below:

```
let place_order = fun ch id item skey →
  assume Order(id,item);
  let pkt = sign skey (id,item) in send ch pkt

let ship_order = fun ch vkey →
  let pkt = recv ch in
    let (xc, xit) = verify vkey pkt in
      assert Ship(xc,xit)
```

The assumption Order*(id,item)* makes the required credential available to the *place_order* function, enabling the subsequent code to sign a request with the key *skey* and send it off over channel *ch*. Upon receiving the message, *ship_order* verifies the signature using the verification key *vkey*, retrieves the two components *xc* and *xit* of the request and asserts the formula Ship*(xc,xit)*.

A client and a server will execute the two functions, communicating on a shared channel *ch* and using a pair of corresponding signing and verification keys, as shown below (the server runs *ship_order* recursively to serve multiple requests):

```
let prot_spec ch =
  assume P;
  let sk = mksigkey () in
    let vk = mkverkey sk in
      let client = (place_order ch "alice" "book" sk) in
        let rec server = (ship_order ch vk) ↱ (server ch vk) in
          client ↱ server
```

The protocol specification given above may be proved *robustly* safe by existing refinement type systems: this ensures that the conjunction of all the assertions which will become active at runtime (i.e., $\mathsf{Ship}(\textit{"alice"}, \textit{"book"})$), is entailed by the active assumptions (i.e., $\mathcal{P}$, $\mathsf{Order}(\textit{"alice"}, \textit{"book"})$), despite the best efforts of an arbitrary opponent. Unfortunately, a closer look reveals that the authorization policy $\mathcal{P}$ is too weak to enforce desirable *resource-aware* access constraints: for instance, in our example the online service is presumably interested in ensuring that each user's order can be cleared and shipped only *once*, but in first-order logic we can prove:

$$\forall x, y.(\mathsf{Order}(x, y) \Rightarrow \mathsf{Ship}(x, y)), \mathsf{Order}(id, item) \vdash \mathsf{Ship}(id, item) \wedge \mathsf{Ship}(id, item),$$

i.e., a single payment by the user can lead to the same order being shipped twice, without violating the previous authorization policy and (robust) safety.

Remarkably, the desired resource-aware authorization policy can be naturally encoded in affine logic by assuming the formula: $\mathcal{P}_{okay} \triangleq\ !\forall x, y.(\mathsf{Order}(x, y) \multimap \mathsf{Ship}(x, y))$, where the bang modality (!) allows using the authorization policy arbitrarily many times in a proof, while the multiplicative implication ($\multimap$) ensures that formulas of the form $\mathsf{Order}(id, item)$ are *consumed* when proving $\mathsf{Ship}(id, item)$. Verifying the desired *injective* correspondence between placed and shipped orders amounts then just to reinterpreting the standard notion of (robust) safety by taking into account the *multiplicative* conjunction ($\otimes$) of the top-level assertions rather than the standard conjunction of first-order logic: roughly, this ensures that the (multi-)set of assumptions can be partitioned in different (multi-)sets, each proving one specific assertion, hence the same assumption is never used in the proof of two different assertions.

Extending refinement type systems to show compliance with respect to affine logic policies like $\mathcal{P}_{okay}$ is challenging. Technically, these type systems support a form of compositional reasoning enabled by the structure of the cryptographic key types, and the typing discipline enforced on them. Briefly, cryptographic key types are associated with refinement types of the form $\mathsf{Key}(\{x : T \mid F\})$, enforcing the following invariants: $(i)$ to package a value $M : T$ with a key of this type, one must be able to prove $F(M)$ and consequently, $(ii)$ upon extracting a value $w : T$ packaged under a key of this type, one may in turn assume the formula $F(w)$ to hold. These two invariants are enough to derive static proofs of robust safety in traditional refinement type systems drawing on classical and intuitionistic logics, but they fall short of providing the necessary guarantees in resource-conscious settings such as the one we consider here.

### 2.2. Exponential serialization for protecting affine formulas

Given the nature of affine formulas as consumable resources, an affine refinement type system must additionally provide protection against an unconstrained assumption of the refinement formulas conveyed by the key types [Bugliesi et al. 2011]. For instance, when receiving a packet signed with a key of type $\mathsf{SigKey}(x : T, \{x : U \mid \mathsf{Order}(x, y)\})$, we must ensure that *each time* we verify the signature (and assume $\mathsf{Order}(x, y)$) at the receiver side, a corresponding assumption has indeed been introduced at the sender side.

Ensuring this kind of injective correspondence in distributed settings is known to require some protective measures, as an adversary may easily break it by mounting a replay attack and fool a receiver into deriving multiple assertions corresponding to one single assumption. We can see that in our running example: given the protocol specification defined above, assume we let it run over an untrusted network by passing the function *prot_spec* as a parameter to the function *adversary* defined below, which intercepts the message by the client and sends it twice to the server:

```
let adversary prot =
  let ch = mkchan () in
    prot ch;
    let m = recv ch in (send ch m) ↱ (send ch m)
```

The replay attack mounted by the adversary breaks the desired injective correspondence between assumptions and assertions, since the system admits a run in which the adversary intercepts the message exchanged on *ch* and duplicates it, leading to two assertions $\mathsf{Ship}(\textit{"alice"}, \textit{"book"})$ being made against just one assumption $\mathsf{Order}(\textit{"alice"}, \textit{"book"})$. More technically, in affine logic we have:

$$!\forall x, y.(\mathsf{Order}(x, y) \multimap \mathsf{Ship}(x, y)), \mathsf{Order}(id, item) \not\vdash \mathsf{Ship}(id, item) \otimes \mathsf{Ship}(id, item),$$

hence the protocol above is *not* robustly safe in our affine setting.

The problem we just outlined is, in fact, rather general and may be stated as follows: data exchanged over the network is inherently exposed to replays, hence their credentials, occurring as refinements of cryptographic key types, must be protected so that replicating the data does not duplicate the credentials. In the type system, this may be achieved by guarding the refinements of the key types with *control* formulas, which are guaranteed to be assumed in at most one point of the protocol code.

The resulting typing discipline leverages the underlying computational measures to counter replay attacks. Though the details vary for the different computational mechanisms, the intuition applies uniformly. The types of cryptographic keys are built around *guarded* refinements of the form:

$$\{\tilde{w} : \tilde{T}, \tilde{x} : \tilde{U} \mid !(C(\tilde{w}) \multimap F(\tilde{x}))\},$$

protecting the credential $F(\tilde{x})$ with the control formula $C(\tilde{w})$. In a nonce-handshake protocol, for instance, $\tilde{w}$ may represent a challenger-generated nonce, call it $n$, and $C(n)$ may be the corresponding guard assumed by the challenger, modeling that the nonce has been freshly generated. Upon receiving the nonce, a responder willing to transmit $M$ will package the pair $(n, M)$ under a key with the above type as a payload: intuitively, the receiver can then open the cryptographic packet to assume the implication above and derive the desired formula $F(M)$ by *consuming* the formula $C(n)$, which was never sent on the network and remained thus under the control of the challenger.

Notice that guarded refinement types as the one above contain an *exponential* formula prefixed by the bang modality, hence opening messages packaged under a key with this type more than once does not really provide additional information to the receiver and is perfectly safe. We call this packaging technique *exponential serialization*, as it provides us with a safe way to transmit payload with affine refinement types over an untrusted network, using an encoding based on exponential formulas.

### 2.3. Serializers for security type-checking

There is one problem left with the intuition above. A responder possessing the credential $F(M)$ and willing to prove it to the challenger will not be able to do so, as in affine logic an assumption $F(M)$ does not entail the guarded exponential formula $!(C(n) \multimap F(M))$, which the responder would need to prove to type-check the response. To close this gap, each affine assumption in the code must be associated with a corresponding *serializer*, to enable its use in the guarded refinements of the key types. Serializers have the general form:

$$!\forall \tilde{x}.\tilde{w}(F(\tilde{x}) \multimap !(C(\tilde{w}) \multimap F(\tilde{x})))),$$

and explicitly enable the transformation of the credential $F(\tilde{M})$ into its serialized form $!(C(\tilde{n} \multimap F(\tilde{M})))$, for appropriate terms $\tilde{n}$ and $\tilde{M}$.

Back to our example, assume we extend the protocol to include the nonce-handshake mentioned above:

```
let place_order' = fun ch1 ch2 id item skey →
  let nonce = recv ch1 in
    assume Order(id, item);
    let pkt = sign skey (id,item,nonce) in send ch2 pkt

let ship_order' = fun ch1 ch2 vkey →
  let mknonce = (fun () → let x = mkfresh () in assume N(x); x) in
    let nonce = mknonce () in
      send ch1 nonce;
      let pkt = recv ch2 in
        let (xc,xit,xn) = verify vkey pkt in
          if (xn = nonce) then
            assert Ship(xc, xit)
          else
            failwith "unauthorized"
```

We assume to be given access to a library function *mkfresh* : $\mathsf{unit} \rightarrow \mathsf{bytes}$, which generates fresh bit-strings. The function *mknonce*: $\mathsf{unit} \rightarrow \{x : \mathsf{bytes} \mid \mathsf{N}(x)\}$ is a wrapper around *mkfresh*, which additionally assumes the control formula $\mathsf{N}(x)$ over the returned value $x$. The new assumption is reflected by the *refined* return type of *mknonce*. Then, the typing of the signing and verification keys may be structured as follows:

$$\begin{aligned} skey &: \mathsf{SigKey}(\{x : \mathsf{string}, y : \mathsf{string}, z : \mathsf{bytes} \mid !(\mathsf{N}(z) \multimap \mathsf{Order}(x, y))\}) \\ vkey &: \mathsf{VerKey}(\{x : \mathsf{string}, y : \mathsf{string}, z : \mathsf{bytes} \mid !(\mathsf{N}(z) \multimap \mathsf{Order}(x, y))\}) \end{aligned}$$

conveying the affine formula $\mathsf{Order}(xc, xit)$ conditionally to the *guard* $\mathsf{N}(nonce)$ assumed by *ship_order'*. If the guard can be proved only once, $\mathsf{Order}(xc, xit)$ can also be retrieved only once, irrespectively of the number of signature verifications performed. To type-check the protocol, we need to assume the expected serializer:

$$\mathcal{S} \triangleq !\forall x, y.z.(\mathsf{Order}(x, y) \multimap !(\mathsf{N}(z) \multimap \mathsf{Order}(x, y))).$$

Overall, we get the following revised protocol:

```
let prot_spec' ch1 ch2 =
  assume P_okay;
  assume S;
  let sk = mksigkey () in
    let vk = mkverkey sk in
      let client' = (place_order' ch1 ch2 "alice" "book" sk) in
        let rec server' = (ship_order' ch1 ch2 vk) ↱ (server' ch vk) in
          client' ↱ server'
```

We briefly discuss how the two protocol components type-check. We start from the server. Upon creating *nonce*, *server'* assumes the control formula $\mathsf{N}(nonce)$ based on the return type of the function *mknonce* by calling *ship_order'*. Upon verifying the received signature, it extracts the refinement $!(\mathsf{N}(xn) \multimap \mathsf{Order}(xc, xit))$ based on the type of the verification key. Then, from the assumption $\mathsf{N}(nonce)$ and the nonce-checking test $xn = nonce$ that protects the assertion, it derives $\mathsf{N}(xn)$. Now, with two $\multimap$ elimination steps, using the refinement above and the policy $\mathcal{P}_{okay}$, it derives the asserted formula $\mathsf{Ship}(xc, xit)$. As to the client, upon receiving the challenge, by calling the function *place_order'*, *client'* assumes the formula $\mathsf{Order}(\textit{"alice"}, \textit{"book"})$ and then signs the triple *(cid,item,nonce)* with *vk*. Typing the signature requires the serializer, which provides a direct way to prove the desired formula.

We notice here that serializers may be generated automatically for any given affine formula, and we prove that introducing them as additional assumptions is sound, in that it does not affect the set of entailed assertions, under the sufficient conditions discussed in Section 4. Furthermore, serializers capture a rather general class of mechanisms for ensuring timely communications, like session keys or timestamps, which are all based on the consumption of an affine resource to assess the freshness of an exchange. We discuss these patterns in our case studies in Sections 8-9.

## 3. REVIEW: AFFINE LOGIC

In our framework we focus on a simple, yet expressive, fragment of intuitionistic affine logic [Troelstra 1992]. We presuppose an underlying signature $\Sigma$ of predicate symbols, ranged over by $p$, and function symbols, ranged over by $f$. The syntax of terms $t$ and formulas $F$ is defined by the following productions:

$$\begin{array}{rcll} t & ::= & x \mid f(t_1, \ldots, t_n) & \text{terms } (f \text{ of arity } n \text{ in } \Sigma) \\ A & ::= & p(t_1, \ldots, t_n) \mid t = t' & \text{atoms } (p \text{ of arity } n \text{ in } \Sigma) \\ F & ::= & A \mid F \otimes F \mid F \multimap F \mid \forall x.F \mid !F \mid \mathbf{0} & \text{formulas} \end{array}$$

This is the multiplicative fragment of affine logic with conjunction ($\otimes$) and implication ($\multimap$), the universal quantifier ($\forall$), the exponential modality (!) to express persistent truths, logical falsity ($\mathbf{0}$) to express negation, and syntactic equality. The logical truth is written $\mathbf{1}$ and encoded as $() = ()$, where $()$ is the nullary function symbol encoding the RCF "unit" value[1]. The negation of $F$, written $F^\perp$, is encoded as $F \multimap \mathbf{0}$, while inequality, written $t \neq t'$, is encoded as $(t = t')^\perp$. For simplicity, we do not consider disjunction and existential quantification: the logic considered here suffices for our purposes and we leave further extensions as future work.

The entailment relation $\Delta \vdash F$ from multiset of formulas to formulas is given in Table I. Observe that, in affine logic, rule (WEAK) can be liberally applied to disregard formulas along a proof derivation, while rule (CONTR) is restricted to exponential formulas, allowing for their unbounded duplication. Intuitively, the combination of the two rules enforces the following usage policy for formulas: "every formula must be used *at most* once in a proof, with the exception of exponential formulas, which can be used arbitrarily many times". This is in contrast with linear logic, where each formula must be used *exactly* once [Girard 1995].

As informally discussed before, affine logic provides multiplicative counterparts of standard logical connectives: for instance, rule ($\otimes$-RIGHT) states that to prove the multiplicative conjunction $F_1 \otimes F_2$ from the hypotheses $\Delta = \Delta_1, \Delta_2$, we have to prove $F_1$ from $\Delta_1$ and $F_2$ from $\Delta_2$, thus each affine hypothesis in $\Delta$ is used either to prove $F_1$ or to prove $F_2$. Analogously, rule ($\multimap$-LEFT) formalizes the intuition that the multi-

[1]We mention here that RCF terms can be encoded into the logic using the locally nameless representation of syntax with binders [de Bruijn 1972], as shown in [Bengtson et al. 2011].

**Table I** The entailment relation $\Delta \vdash F$

$$\text{(IDENT)}\ F \vdash F \qquad \text{(WEAK)}\ \frac{\Delta \vdash F'}{\Delta, F \vdash F'} \qquad \text{(CONTR)}\ \frac{\Delta, !F, !F \vdash F'}{\Delta, !F \vdash F'} \qquad (\otimes\text{-LEFT})\ \frac{\Delta, F_1, F_2 \vdash F'}{\Delta, F_1 \otimes F_2 \vdash F'} \qquad (\otimes\text{-RIGHT})\ \frac{\Delta_1 \vdash F_1 \qquad \Delta_2 \vdash F_2}{\Delta_1, \Delta_2 \vdash F_1 \otimes F_2}$$

$$(\multimap\text{-LEFT})\ \frac{\Delta_1 \vdash F_1 \qquad \Delta_2, F_2 \vdash F'}{\Delta_1, F_1 \multimap F_2, \Delta_2 \vdash F'} \qquad (\multimap\text{-RIGHT})\ \frac{\Delta, F_1 \vdash F_2}{\Delta \vdash F_1 \multimap F_2} \qquad (\forall\text{-LEFT})\ \frac{\Delta, F\{t/x\} \vdash F'}{\Delta, \forall x.F \vdash F'} \qquad (\forall\text{-RIGHT})\ \frac{\Delta \vdash F \qquad x \notin fv(\Delta)}{\Delta \vdash \forall x.F}$$

$$(!\text{-LEFT})\ \frac{\Delta, F \vdash F'}{\Delta, !F \vdash F'} \qquad (!\text{-RIGHT})\ \frac{!\Delta \vdash F}{!\Delta \vdash !F} \qquad \text{(FALSE)}\ \mathbf{0} \vdash F \qquad (=\text{-SUBST})\ \frac{\exists \sigma = mgu(t, t') \Rightarrow \Delta\sigma \vdash F\sigma}{\Delta, t = t' \vdash F} \qquad (=\text{-REFL})\ \Delta \vdash t = t$$

plicative implication $F_1 \multimap F_2$ acts as a sort of reaction, which consumes the resources needed to prove the premise $F_1$ to produce the conclusion $F_2$.

Rule (!-LEFT) is often called the *dereliction* rule and allows exponential assumptions to be degraded to affine assumptions, which can be used at most once. Rule (!-RIGHT), instead, is typically referred to as the *promotion* rule, which allows one to prove exponential formulas starting from the proof of an affine formula: the notation $!\Delta$ means that every formula in $\Delta$ must be of the form $!F$. The two rules for equality (=-SUBST) and (=-REFL) are borrowed from [Tiu and Momigliano 2012]; in rule (=-SUBST), if the terms $t$ and $t'$ are not unifiable, then we consider the premise as trivially fulfilled.

## 4. METATHEORY OF EXPONENTIAL SERIALIZATION

Recall from Section 2.3 that we had to explicitly assume a serializer $\mathcal{S}$ to make our example protocol type-check. In principle, the introduction of this serializer among the assumed hypotheses could alter the intended semantics of the authorization policy $\mathcal{P}_{okay}$, due to the subtle interplay of formulas through the entailment relation defined in Table I. Here, we isolate sufficient conditions under which exponential serialization leads to a sound protection mechanism for affine formulas.

We presuppose that the signature $\Sigma$ of predicate symbols is partitioned in two sets $\Sigma_A$ and $\Sigma_C$. Atomic formulas $A$ have the form $p(t_1, \ldots, t_n)$ for some $p \in \Sigma_A$; control formulas $C$ have the same form, though with $p \in \Sigma_C$. We identify various categories of formulas defined by the following productions:

$$\begin{array}{lcll} B & ::= & A \mid B \otimes B \mid B \multimap B \mid \forall x.B \mid !B & \text{base formulas} \\ P & ::= & B \mid C \mid P \otimes P & \text{payload formulas} \\ G & ::= & C \multimap P \mid !G & \text{guarded formulas} \end{array}$$

Base formulas $B$ are formulas of an authorization policy, built from atomic formulas using logical connectives. We use base formulas as security annotations in the application code. For simplicity, we dispense in this section with equalities and $\mathbf{0}$ to ensure logical consistency: these elements are used in our typed analysis, but we stipulate that they are never directly assumed in the protocol code (and thus never serialized).

Payload formulas $P$ are formulas which we want to serialize for communication over the untrusted network. Importantly, payload formulas comprise both base formulas and control formulas, which allows, e.g., for the transmission of fresh nonces to remote verifiers: this pattern is present in several authentication protocols [Gordon and Jeffrey 2003]. Finally, guarded formulas $G$ are used to model the serialized version of payload formulas, suitable for transmission. Notice also that serializers are not generated by any of the previous productions, so we let $S$ stand for any serializer of the form $!\forall \tilde{x}.(P \multimap !(C \multimap P))$. We write $\Delta \vdash F^n$ for $\Delta \vdash F \otimes \ldots \otimes F$ ($n$ times), with the proviso that $\Delta \vdash F^0$ stands for $\Delta \nvdash F$.

Intuitively, given a multiset of assumptions $\Delta$, the extension of $\Delta$ with the serializers $S_1, \ldots, S_n$ is sound if $\Delta$ and its extension derive the same payload formulas. As it turns out, this is only true when $\Delta$ satisfies additional conditions, which we formalize next.

*Definition* 4.1 (*Rank*). Let $\textit{rk} : \Sigma_C \to \mathbb{N}$ be a total, injective function. Given a formula $F$, we define the *rank* of $F$ with respect to *rk*, denoted by $\textit{rk}(F)$, as follows:

$$\begin{array}{lll} \textit{rk}(p(t_1, \ldots, t_n)) & = \textit{rk}(p) & \text{if } p \in \Sigma_C \\ \textit{rk}(F_1 \otimes F_2) & = \textit{min}\, \{\textit{rk}(F_1), \textit{rk}(F_2)\} & \\ \textit{rk}(F) & = +\infty & \text{otherwise} \end{array}$$

*Definition* 4.2 (*Stratification*). A formula $F$ is *stratified* with respect to a rank function *rk* if and only if: $(i)$ $F = C \multimap P$ implies $\textit{rk}(C) < \textit{rk}(P)$; $(ii)$ $F = P \multimap G$ implies that $G$ is stratified; $(iii)$ $F = \forall x.F'$ implies that $F'$ is stratified; $(iv)$ $F = !F'$ implies that $F'$ is stratified. We assume $F$ to be stratified in all the other cases. We say that a multiset of formulas $\Delta$ is stratified if and only if there exists a rank function *rk* such that each formula in $\Delta$ is stratified with respect to *rk*.

For instance, the multiset $C_1 \multimap C_2, C_2 \multimap C_3$, where $C_1, C_2, C_3$ are built over distinct predicate symbols, is stratified, given an appropriate choice of a rank function, while the multiset $C_1 \multimap C_2, C_2 \multimap C_1$ is not stratified. Stratification is required precisely to disallow these circular dependencies among control formulas and simplify the proof of our soundness result, Theorem 4.4 below. To prove that result, we need a further definition:

*Definition* 4.3 (*Controlled Multiset*). Let $\Delta = P_1, \ldots, P_m, S_1, \ldots, S_n$ be a stratified multiset of formulas. We say that $\Delta$ is *controlled* if and only if $\Delta \vdash C^k$ implies $k \leq 1$ for any control formula $C$.

The intuition underlying the definition may be explained as follows. Consider a multiset $\Delta$, a payload formula $P$ such that $\Delta \vdash P$ and let $S = !\forall\tilde{x}.(P \multimap !(C \multimap P))$ be a serializer for $P$. Now, the only way that $S$ may affect derivability is by allowing for the duplication of the payload formula $P$ via the exponential implication $!(C \multimap P)$, since the latter can be used arbitrarily often in a proof derivation. However, this effect is prevented if we are guaranteed that the control formula $C$ guarding $P$ is derived at most once in $\Delta$: that is precisely what the condition above ensures.

THEOREM 4.4 (SOUNDNESS OF SERIALIZATION). *Let $\Delta = P_1, \ldots, P_m$. If $\Delta' = \Delta, S_1, \ldots, S_n$ is controlled and $\Delta' \vdash P$, then $\Delta \vdash P$ for all payload formulas $P$.*

PROOF. See Appendix A. □

Notice that checking if a multiset of formulas is controlled may be difficult, since this depends on logical entailment, hence it may be not obvious when the theorem above can be applied. Fortunately, however, we can isolate a sufficient criterion to decide whether a multiset of formulas is controlled, based on a simple syntactic check.

PROPOSITION 4.5 (CHECKING CONTROL). *If $\Delta = P_1, \ldots, P_m, S_1, \ldots, S_n$ is stratified and the control formulas occurring in $P_1, \ldots, P_m$ are pairwise distinct, then $\Delta$ is controlled.*

PROOF. See Appendix A. □

## 5. RCF AND SAFETY

We now review RCF [Bengtson et al. 2011], a concurrent $\lambda$-calculus with message passing primitives, which provides the core language around which our theory is developed.

**Table II** Syntax of RCF expressions

| | | |
|---|---|---|
| $M, N ::=$ | | *values* |
| | $x$ | variable |
| | $()$ | unit |
| | $(M, N)$ | pair |
| | $\lambda x.\, E$ | function |
| | $h\ M$ | construction ($h \in \{\mathsf{inl}, \mathsf{inr}, \mathsf{fold}\}$) |
| $D, E ::=$ | | *expressions* |
| | $M$ | value |
| | $M\ N$ | application |
| | $M = N$ | syntactic equality |
| | $\mathsf{let}\ x = E\ \mathsf{in}\ E'$ | let (scope of $x$ is $E'$) |
| | $\mathsf{let}\ (x, y) = M\ \mathsf{in}\ E$ | pair split (scope of $x, y$ is $E$) |
| | $\mathsf{match}\ M\ \mathsf{with}\ h\ x\ \mathsf{then}\ E\ \mathsf{else}\ E'$ | match (scope of $x$ is $E$) |
| | $(\nu a)E$ | restriction (scope of $a$ is $E$) |
| | $E \upharpoonright E'$ | fork |
| | $a!M$ | message send |
| | $a?$ | message receive |
| | $\mathsf{assume}\ F$ | assumption |
| | $\mathsf{assert}\ F$ | assertion |

We also formally introduce the resource-aware variant of the standard notion of safety for RCF, which we have been mentioning.

### 5.1. Review of RCF

We assume collections of names ($a, b, c, m, n$) and variables ($x, y, z$). The syntax of values and expressions of RCF is introduced in Table II. The notions of free names and free variables arise as expected, according to the scope defined in the table.

Values include variables, unit, pairs, functions and constructions; constructors account for the creation of standard tagged unions and iso-recursive types. We also encode the boolean values $\mathsf{true} \triangleq \mathsf{inl}()$ and $\mathsf{false} \triangleq \mathsf{inr}()$. Expressions of RCF include standard $\lambda$-calculus constructs like values, applications, equality checks, lets, pair splits, and pattern matching, as well as primitives for concurrent, message-passing computations in the style of process algebras.

The semantics is mostly standard. The function application $(\lambda x.\, E)\ N$ evaluates to $E\{N/x\}$; the syntactic equality check $M = N$ evaluates to $\mathsf{true}$ when $M$ is equal to $N$ and to $\mathsf{false}$ otherwise; the let expression $\mathsf{let}\ x = E\ \mathsf{in}\ E'$ first evaluates $E$ to a value $N$ and then behaves as $E'\{N/x\}$; the pair splitting $\mathsf{let}\ (x, y) = (M, N)\ \mathsf{in}\ E$ evaluates to $E\{M/x\}\{N/y\}$; and the pattern matching $\mathsf{match}\ M\ \mathsf{with}\ h\ x\ \mathsf{then}\ E\ \mathsf{else}\ E'$ evaluates to $E\{N/x\}$ when $M$ is equal to $h\ N$ for some $N$, while it evaluates to $E'$ otherwise. We then have some constructs reminiscent of process algebras: expression $(\nu a)E$ generates a globally fresh channel name $a$ and then behaves as $E$. Expression $E \upharpoonright E'$ evaluates $E$ and $E'$ in parallel, and returns the result of $E'$. Expression $a!M$ asynchronously outputs $M$ on channel $a$ and returns $()$. Expression $a?$ waits until a term $N$ is available on channel $a$ and returns $N$. These message-passing expressions can be used to model the sending and receiving functions "*send*" and "*recv*" that are used in the code of our examples and that we further explain in Section 7.1. Assumptions and assertions are stuck expressions, which are just needed to state our safety notion (see below). The formal semantics of RCF expressions is defined by the reduction rules in Table III.

The reduction semantics depends upon the heating relation $E \Rrightarrow E'$, an asymmetric version of the standard structural congruence, to perform some syntactic rearrangements of expressions and allow reductions. We write $E \equiv E'$ to denote that both $E \Rrightarrow E'$ and $E' \Rrightarrow E$. The definition of the heating relation is presented in Table IV, the only

**Table III** Reduction semantics for RCF

| | |
|---|---|
| $(\lambda x.\, E)\ N \to E\{N/x\}$ | (RED FUN) |
| let $(x,y) = (M,N)$ in $E \to E\{M/x\}\{N/y\}$ | (RED SPLIT) |
| match $M$ with $h\ x$ then $E$ else $E' \to \begin{cases} E\{N/x\} & \text{if } M = h\ N \text{ for some } N \\ E' & \text{otherwise} \end{cases}$ | (RED MATCH) |
| $M = N \to \begin{cases} \mathsf{true} & \text{if } M = N \\ \mathsf{false} & \text{otherwise} \end{cases}$ | (RED EQ) |
| $a!M \upharpoonright a? \to M$ | (RED COMM) |
| let $x = M$ in $E \to E\{M/x\}$ | (RED LET VAL) |
| let $x = E$ in $E'' \to$ let $x = E'$ in $E''$ if $E \to E'$ | (RED LET) |
| $(\nu a)E \to (\nu a)E'$ if $E \to E'$ | (RED RES) |
| $E \upharpoonright E'' \to E' \upharpoonright E''$ if $E \to E'$ | (RED FORK 1) |
| $E'' \upharpoonright E \to E'' \upharpoonright E'$ if $E \to E'$ | (RED FORK 2) |
| $E \to E'$ if $E \Rrightarrow D, D \to D', D' \Rrightarrow E'$ | (RED HEAT) |

difference with respect to the original RCF presentation is the introduction of the rule (HEAT ASSERT ()), which simplifies our definition of safety.

**Table IV** Heating relation for RCF

| | |
|---|---|
| $E \Rrightarrow E$ | (HEAT REFL) |
| $E \Rrightarrow E''$ if $E \Rrightarrow E'$ and $E' \Rrightarrow E''$ | (HEAT TRANS) |
| let $x = E$ in $E'' \Rrightarrow$ let $x = E'$ in $E''$ if $E \Rrightarrow E'$ | (HEAT LET) |
| $(\nu a)E \Rrightarrow (\nu a)E'$ if $E \Rrightarrow E'$ | (HEAT RES) |
| $E \upharpoonright E'' \Rrightarrow E' \upharpoonright E''$ if $E \Rrightarrow E'$ | (HEAT FORK 1) |
| $E'' \upharpoonright E \Rrightarrow E'' \upharpoonright E'$ if $E \Rrightarrow E'$ | (HEAT FORK 2) |
| $() \upharpoonright E \equiv E$ | (HEAT FORK ()) |
| $a!M \Rrightarrow a!M \upharpoonright ()$ | (HEAT MSG ()) |
| assume $F \Rrightarrow$ assume $F \upharpoonright ()$ | (HEAT ASSUME ()) |
| assert $F \Rrightarrow$ assert $F \upharpoonright ()$ | (HEAT ASSERT ()) |
| $E' \upharpoonright (\nu a)E \Rrightarrow (\nu a)(E' \upharpoonright E)$ if $a \notin fn(E')$ | (HEAT RES FORK 1) |
| $(\nu a)E \upharpoonright E' \Rrightarrow (\nu a)(E \upharpoonright E')$ if $a \notin fn(E')$ | (HEAT RES FORK 2) |
| let $x = (\nu a)E$ in $E' \Rrightarrow (\nu a)($let $x = E$ in $E')$ if $a \notin fn(E')$ | (HEAT RES LET) |
| $(E \upharpoonright E') \upharpoonright E'' \equiv E \upharpoonright (E' \upharpoonright E'')$ | (HEAT FORK ASSOC) |
| $(E \upharpoonright E') \upharpoonright E'' \Rrightarrow (E' \upharpoonright E) \upharpoonright E''$ | (HEAT FORK COMM) |
| let $x = (E \upharpoonright E')$ in $E'' \equiv E \upharpoonright ($let $x = E'$ in $E'')$ | (HEAT FORK LET) |

### 5.2. Resource-aware safety

We are now ready to adapt the formal notion of safety defined for RCF expressions to our resource-aware setting. Intuitively, an expression $E$ is safe if, for all runs, the *multiplicative* conjunction of the top-level assertions is entailed by the top-level assumptions. Giving a precise definition, however, is somewhat tricky and it is convenient to introduce the notion of *structure* for this purpose.

Let $e$ denote an *elementary* expression, i.e., any expression that is not an assumption, assertion, restriction, let, fork, or send. Structures formalize the idea that a computation state has four components: (1) a multiset of assumed formulas $F_i$; (2) a multiset of asserted formulas $F'_j$; (3) a series of messages $M_k$ sent on channels but not yet received; and (4) a series of elementary expressions $e_\ell$ being evaluated in parallel contexts. The definition of a structure **S** is given in Table V. Structures are convenient, since their syntactic form already exhibits all the necessary ingredients to state a simple notion of *static safety*, the basic building block for safety.

We can prove that every expression $E$ can be transformed into a structure by heating, hence we can define a suitable notion of safety for any expression.

**Table V** Structures and static safety

$$\Pi_{i\in[1,n]}E_i \triangleq () \upharpoonright E_1 \upharpoonright \cdots \upharpoonright E_n$$
$$\mathcal{L}[e] ::= e \mid \mathsf{let}\ x = \mathcal{L}[e]\ \mathsf{in}\ E$$
$$\mathbf{S} ::= (\nu\widetilde{a})((\Pi_{i\in[1,m]}\mathsf{assume}\ F_i) \upharpoonright (\Pi_{j\in[1,n]}\mathsf{assert}\ F'_j) \upharpoonright (\Pi_{k\in[1,o]}c_k!M_k) \upharpoonright (\Pi_{\ell\in[1,p]}\mathcal{L}_\ell[e_\ell]))$$

The structure S above is *statically safe* if and only if $F_1, \ldots, F_m \vdash F'_1 \otimes \ldots \otimes F'_n$.

LEMMA 5.1 (STRUCTURE). *For every expression $E$, there exists a structure* S *such that $E \Rrightarrow$* S.

PROOF. By induction on the structure of $E$. □

*Definition* 5.2 (*Safety*). A closed expression $E$ is *safe* if and only if, for all $E'$ and S, if $E \rightarrow^* E'$ and $E' \Rrightarrow$ S, then S is statically safe.

The real property of interest, however, is stronger than the previous one: we desire protection despite the best efforts of an active opponent. We let an *opponent* be any closed expression of RCF which does not contain any assumption or assertion. The latter is a standard restriction, since opponents containing arbitrary assertions could vacuously falsify the property we target; this does not involve any loss of generality in practice, since we want to verify application code with respect to the security annotations placed therein. We note that security annotations are simply considered a tool for verification but that they hold no semantic meaning and are thus not necessary for the opponent code.

*Definition* 5.3 (*Robust Safety*). A closed expression $E$ is *robustly safe* if and only if, for any opponent $O$, the application $O\ E$ is safe[2].

## 6. THE TYPE SYSTEM

Our refinement type system builds on previous work by Bengtson et al. [Bengtson et al. 2011], extending it to guarantee the correct usage of affine formulas and to enforce our revised notion of (robust) safety.

### 6.1. Types, typing environments, and base judgements

The syntax of types is defined in Table VI. Again the notions of free names and free variables arise as expected, according to the scope defined in the table.

**Table VI** Syntax of types

| | | |
|---|---|---|
| $T, U, V$ ::= | | *types* |
| | unit | unit type |
| | $x : T \rightarrow U$ | dependent function type (scope of $x$ is $U$) |
| | $x : T * U$ | dependent pair type (scope of $x$ is $U$) |
| | $T + U$ | sum type |
| | $\mu\alpha. T$ | iso-recursive type (scope of $\alpha$ is $T$) |
| | $\alpha$ | type variable |
| | $\{x : T \mid F\}$ | refinement type (scope of $x$ is $F$) |

The unit value () is given type unit. Sum types have the form $T + U$, iso-recursive types are denoted by $\mu\alpha. T$, and type variables are denoted by $\alpha$. There exist various forms of dependent types: a function of type $x : T \rightarrow U$ takes as an input a value $M$ of type $T$ and returns a value of type $U\{M/x\}$; a pair $(M, N)$ has type $x : T * U$ if $M$ has type $T$ and $N$ has type $U\{M/x\}$; a value $M$ has a refinement type $\{x : T \mid F\}$ if $M$

[2]Here, we use the standard syntactic sugar $O\ E$ for the expression $\mathsf{let}\ x = O\ \mathsf{in}\ \mathsf{let}\ y = E\ \mathsf{in}\ x\ y$.

has type $T$ and the formula $F\{M/x\}$ holds true. We use type $\mathsf{Un} \triangleq \mathsf{unit}$ to model data that may come from, or be sent to the opponent, as it is customary for security type systems.[3] Type $\mathsf{bool} \triangleq \mathsf{unit} + \mathsf{unit}$ is inhabited by $\mathsf{true} \triangleq \mathsf{inl}()$ and $\mathsf{false} \triangleq \mathsf{inr}()$.

The type system comprises several typing judgements of the form $\Gamma; \Delta \vdash \mathcal{J}$, where $\Gamma; \Delta$ is a typing environment collecting all the information which can be used to derive $\mathcal{J}$. In particular, $\Gamma$ contains the type bindings, while $\Delta$ comprises logical formulas that are supposed to hold at run-time. Formally, we let $\Gamma$ be an ordered list of entries $\mu_1, \ldots, \mu_n$ and $\Delta$ be a multiset of affine logic formulas. Each entry $\mu_i$ in $\Gamma$ denotes either a type variable ($\alpha$), a kinding annotation ($\alpha :: k$), or a type binding for channels ($a \updownarrow T$) or variables ($x : T$). We let $\varepsilon$ denote the empty list and $\emptyset$ the empty multiset. The *domain* of $\Gamma$, written $dom(\Gamma)$, is defined as follows: $dom(\alpha) = \{\alpha\}$; $dom(\alpha :: k) = \{\alpha\}$; $dom(a \updownarrow T) = \{a\}$; $dom(x : T) = \{x\}$; and $dom(\mu_1, \ldots, \mu_n) = dom(\mu_1) \cup \ldots \cup dom(\mu_n)$. The set of *free* variables and free names is denoted by $\mathit{fnfv}$. The definition is standard.

We first discuss the base judgements of the type system. We use the judgement $\Gamma; \Delta \vdash \diamond$ to denote that the typing environment $\Gamma; \Delta$ is well-formed, i.e., it satisfies some standard syntactic conditions (for instance, it does not contain duplicate type bindings for the same variable). The only remarkable point in the definition of $\Gamma; \Delta \vdash \diamond$ is that we forbid variables in $\Gamma$ to be mapped to a refinement type: indeed, when extending a typing environment with a new type binding $x : T$, we will use the function $\psi$ to place the structural type information in $\Gamma$ and the function $\mathit{forms}$ to place the associated refinements in $\Delta$. We also write $\Gamma; \Delta \vdash T$ to denote that type $T$ is well-formed in $\Gamma; \Delta$ and $\Gamma; \Delta \vdash F$ when the formulas in $\Delta$ entail the formula $F$. We often abuse notation and write $\Gamma; \Delta \vdash F_1, \ldots, F_n$ to stand for $\Gamma; \Delta \vdash F_1 \otimes \ldots \otimes F_n$, with the proviso that $\Gamma; \Delta \vdash \emptyset$ is equivalent to $\Gamma; \Delta \vdash \mathbf{1}$. A complete formal definition of the described elements is given in Table VII below.

**Table VII** Auxiliary functions and base judgements

$$\psi(U) = \begin{cases} \psi(T) & \text{if } U = \{x : T \mid F\} \\ U & \text{otherwise} \end{cases} \qquad \mathit{forms}(y : U) = \begin{cases} F\{y/x\}, \mathit{forms}(y : T) & \text{if } U = \{x : T \mid F\} \\ \emptyset & \text{otherwise} \end{cases}$$

(ENV EMPTY)
$$\varepsilon; \emptyset \vdash \diamond$$

(TYPE ENV ENTRY)
$$\frac{\Gamma; \Delta \vdash \diamond \qquad dom(\mu) \cap dom(\Gamma) = \emptyset \qquad \mu = x : T \Rightarrow T = \psi(T) \wedge \mathit{fnfv}(T) \subseteq dom(\Gamma)}{\Gamma, \mu; \Delta \vdash \diamond}$$

(FORM ENV ENTRY)
$$\frac{\Gamma; \Delta \vdash \diamond \qquad \mathit{fnfv}(F) \subseteq dom(\Gamma)}{\Gamma; \Delta, F \vdash \diamond}$$

(TYPE)
$$\frac{\Gamma; \Delta \vdash \diamond \qquad \mathit{fnfv}(T) \subseteq dom(\Gamma)}{\Gamma; \Delta \vdash T}$$

(DERIVE)
$$\frac{\Gamma; \Delta \vdash \diamond \qquad \mathit{fnfv}(F) \subseteq dom(\Gamma) \qquad \Delta \vdash F}{\Gamma; \Delta \vdash F}$$

### 6.2. Environment rewriting

We stipulate that all the type information stored in $\Gamma$ can be used arbitrarily often in the derivation of any judgement of our type system, hence we dispense with affine types[4]. The treatment of the formulas in $\Delta$ is subtler, since affine resources must be

[3]Note that other types built over Un are available to the opponent through subtyping.
[4]In Section 6.8 we thoroughly discuss why this does not involve any loss in expressiveness, by showing an encoding of affine types through exponential serialization.

used at most once during type-checking: in particular, we need to split the environment $\Delta$ among subderivations to avoid the unbounded duplication of the formulas therein. However, a simple splitting of the formulas in $\Delta$ would lead to a very restrictive type system. To illustrate, let $\Delta \triangleq A, A \multimap\, !B$: if we just distributed the formulas $A$ and $A \multimap\, !B$ between two distinct subderivations, then the formula $!B$ would be available only in (at most) one subderivation, despite it being an exponential formula, which we may want to use arbitrarily often during type-checking.

The general structure of the rules of our system then looks as follows:

$$\frac{\Gamma; \Delta_1 \vdash \mathcal{J}_1 \qquad \ldots \qquad \Gamma; \Delta_n \vdash \mathcal{J}_n \qquad \Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \ldots, \Delta_n}{\Gamma; \Delta \vdash \mathcal{J}}$$

where $\Gamma; \Delta \hookrightarrow \Gamma; \Delta'$ denotes the *environment rewriting* of $\Gamma; \Delta$ to $\Gamma; \Delta'$. This relation is defined by rule (REWRITE) below:

$$\text{(REWRITE)}\quad \frac{\Delta \vdash \Delta' \qquad \Gamma; \Delta \vdash \diamond \qquad \Gamma; \Delta' \vdash \diamond}{\Gamma; \Delta \hookrightarrow \Gamma; \Delta'}$$

where we write $\Delta \vdash F_1, \ldots, F_n$ to denote that $\Delta \vdash F_1 \otimes \ldots \otimes F_n$, again with the proviso that $\Delta \vdash \emptyset$ stands for $\Delta \vdash \mathbf{1}$. Coming back to our previous example, notice that we have $A, A \multimap\, !B \vdash\, !B \otimes !B$ in affine logic, hence we can obtain two copies of $!B$ upon rewriting and distribute them between two distinct subderivations upon type-checking. As we will explain in Section 6.5, for soundness reasons we will often rely on rewriting of the form $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$, where $!\Delta'$ is a so-called *exponential* environment, i.e., an environment of the form $!F_1, \ldots, !F_n$.

The adoption of the environment rewriting relation as an house-keeping device for the formulas in $\Delta$ greatly improves the expressiveness of the type system in a very natural way. This idea of extending to the typing environment a number of context manipulation rules from the underlying substructural logic was first proposed by Mandelbaum et al. [Mandelbaum et al. 2003], even though their solution is technically different from ours. Namely, the authors of [Mandelbaum et al. 2003] allow for applications of arbitrary left rules from the logic inside the typing environment, while our proposal is reminiscent of the (CUT) rule typical of sequent calculi. We find this solution simpler to present and more convenient to prove sound.

Interestingly, all the non-determinism introduced by the application of the rewriting rules and the splitting of the logical formulas among the premises of the type rules can be effectively tamed by the algorithmic type system discussed in Section 10.

### 6.3. Kinding

Security type systems often rely on a kinding relation to discriminate whether or not messages of a specific type may be sent to the attacker or generated by it. The kinding judgement $\Gamma; \Delta \vdash T :: k$ denotes that type $T$ is of kind $k$. We distinguish between two kinds: kind $k = \mathsf{pub}$ denotes that the inhabitants of a given type are public and may be sent to the attacker, while kind $k = \mathsf{tnt}$ denotes that the inhabitants of a given type are tainted and may come from the attacker. We let $\overline{\mathsf{pub}} \triangleq \mathsf{tnt}$ and $\overline{\mathsf{tnt}} \triangleq \mathsf{pub}$.

The complete kinding relation is given in Table VIII. Most of the rules resemble those presented in other security type systems [Bengtson et al. 2011; Backes et al. 2011] and only differ in the treatment of affine formulas, which is similar to the one we employ for typing values and expressions. We postpone the discussion on this point until the next section, where it will be easier to provide an intuitive understanding. Here, we just point out some simple observations, which should hopefully guide the reader in understanding a few important aspects.

**Table VIII** Kinding rules

(KIND VAR)
$$\frac{\Gamma; \Delta \vdash \diamond \qquad (\alpha :: k) \in \Gamma}{\Gamma; \Delta \vdash \alpha :: k}$$

(KIND UNIT)
$$\frac{\Gamma; \Delta \vdash \diamond}{\Gamma; \Delta \vdash \mathsf{unit} :: k}$$

(KIND FUN)
$$\frac{\Gamma; !\Delta_1 \vdash T :: \overline{k} \qquad \Gamma, x : \psi(T); !\Delta_2 \vdash U :: k \qquad \Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2}{\Gamma; \Delta \vdash x : T \to U :: k}$$

(KIND PAIR)
$$\frac{\Gamma; !\Delta_1 \vdash T :: k \qquad \Gamma, x : \psi(T); !\Delta_2 \vdash U :: k \qquad \Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2}{\Gamma; \Delta \vdash x : T * U :: k}$$

(KIND SUM)
$$\frac{\Gamma; !\Delta_1 \vdash T :: k \qquad \Gamma; !\Delta_2 \vdash U :: k \qquad \Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2}{\Gamma; \Delta \vdash T + U :: k}$$

(KIND REC)
$$\frac{\Gamma, \alpha :: k; !\Delta' \vdash T :: k \qquad \Gamma; \Delta \hookrightarrow \Gamma; !\Delta'}{\Gamma; \Delta \vdash \mu\alpha. T :: k}$$

(KIND REFINE PUBLIC)
$$\frac{\Gamma; \Delta \vdash \{x : T \mid F\} \qquad \Gamma; \Delta \vdash T :: \mathsf{pub}}{\Gamma; \Delta \vdash \{x : T \mid F\} :: \mathsf{pub}}$$

(KIND REFINE TAINTED)
$$\frac{\Gamma; \Delta_1 \vdash \psi(T) :: \mathsf{tnt} \qquad \Gamma, y : \psi(T); \Delta_2 \vdash \mathit{forms}(y : T) \qquad \Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2 \qquad T \text{ refined}}{\Gamma; \Delta \vdash T :: \mathsf{tnt}}$$

The type unit is assumed to be both public and tainted by (KIND UNIT). According to (KIND PAIR), a pair type is public if both its components are public and can be disclosed to the opponent. Conversely, by the same rule, a pair type is tainted if both its components are tainted, since, if even a single component of the pair is untainted, then the pair cannot come from the opponent. The kinding of sum types (KIND SUM) behaves analogously. By rule (KIND FUN) a function type is public (thus available to the attacker) only if its return type is public (otherwise $\lambda x. M_{\text{secret}}$ could be public and leak a secret to the attacker) and its argument type is tainted such that it can be called by the attacker. The treatment of tainted function types is dual. To give kind $k$ to an iso-recursive type with a bound variable $\alpha$, the rule (KIND REC) proceeds recursively and extends the typing environment in the premise with the kinding annotation $\alpha :: k$. These kinding annotations are used when kinding a type variable (KIND VAR). By (KIND REFINE PUBLIC) a refinement type is public if the structural type it refines is public, while by (KIND REFINE TAINTED) it is tainted if its structural information is tainted and its refinements are entailed by the typing environment.

### 6.4. Subtyping

The subtyping judgment $\Gamma; \Delta \vdash T <: U$ expresses the fact that $T$ is a subtype of $U$ and, thus, values of type $T$ can be safely used in place of values of type $U$. The complete presentation of the subtyping relation can be found in Table IX.

We first note that subtyping is reflexive by (SUB REFL). Furthermore, the subtyping judgment makes public types subtype of tainted types through rule (SUB PUB TNT), and further describes standard subtyping relations for types sharing the same structure: for instance, pair and sum types are covariant (cf. (SUB PAIR) and (SUB SUM)), while function types are contravariant in their arguments and covariant in their return types (cf. (SUB FUN)). Intuitively, this means that a function can safely replace another function if it is "more liberal" in the types it accepts and "more conservative" in the types it returns.

The rule for iso-recursive types (SUB POS REC) is borrowed from [Backes et al. 2011] and it differs from the standard Amber rule proposed in the original presentation of RCF: the rule we consider here is easier to prove sound and the loss of expressiveness is very mild. We refer the interested reader to [Backes et al. 2011] for further discussion on this technical point.

The most interesting subtyping rule in Table IX is (SUB REFINE), which subsumes the rules (SUB REFINE LEFT) and (SUB REFINE RIGHT) from the original presentation

**Table IX** Subtyping rules

$$\frac{\Gamma;\Delta \vdash T}{\Gamma;\Delta \vdash T <: T}\ (\textsc{Sub Refl})$$

$$\frac{\Gamma;\Delta_1 \vdash T :: \mathsf{pub} \qquad \Gamma;\Delta_2 \vdash U :: \mathsf{tnt} \qquad \Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2}{\Gamma;\Delta \vdash T <: U}\ (\textsc{Sub Pub Tnt})$$

$$\frac{\Gamma;!\Delta_1 \vdash T' <: T \qquad \Gamma, x:\psi(T');!\Delta_2 \vdash U <: U' \qquad \Gamma;\Delta \hookrightarrow \Gamma;!\Delta_1,!\Delta_2}{\Gamma;\Delta \vdash x:T \to U <: x:T' \to U'}\ (\textsc{Sub Fun})$$

$$\frac{\Gamma;!\Delta_1 \vdash T <: T' \qquad \Gamma, x:\psi(T);!\Delta_2 \vdash U <: U' \qquad \Gamma;\Delta \hookrightarrow \Gamma;!\Delta_1,!\Delta_2}{\Gamma;\Delta \vdash x:T * U <: x:T' * U'}\ (\textsc{Sub Pair})$$

$$\frac{\Gamma;!\Delta_1 \vdash T <: T' \qquad \Gamma;!\Delta_2 \vdash U <: U' \qquad \Gamma;\Delta \hookrightarrow \Gamma;!\Delta_1,!\Delta_2}{\Gamma;\Delta \vdash T + U <: T' + U'}\ (\textsc{Sub Sum})$$

$$\frac{\Gamma,\alpha;!\Delta' \vdash T <: T' \qquad \alpha \text{ occurs only positively in } T \text{ and } T' \qquad \Gamma;\Delta \hookrightarrow \Gamma;!\Delta'}{\Gamma;\Delta \vdash \mu\alpha.T <: \mu\alpha.T'}\ (\textsc{Sub Pos Rec})$$

$$\frac{\Gamma;\Delta_1 \vdash \psi(T) <: \psi(U) \qquad \Gamma, y:\psi(T);\Delta_2, \mathit{forms}(y:T) \vdash \mathit{forms}(y:U) \qquad \Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2 \qquad T \text{ and/or } U \text{ refined}}{\Gamma;\Delta \vdash T <: U}\ (\textsc{Sub Refine})$$

of RCF, which are shown below:

$$\frac{\Gamma \vdash \{x:T \mid F\} \qquad \Gamma \vdash T <: U}{\Gamma \vdash \{x:T \mid F\} <: U}\ (\textsc{Sub Refine Left}) \qquad \frac{\Gamma \vdash T <: U \qquad \Gamma, x:T \vdash F}{\Gamma \vdash T <: \{x:U \mid F\}}\ (\textsc{Sub Refine Right})$$

The first rule allows discarding unneeded logical formulas and conforms to the core idea of "refinement" typing: values of type $\{x : T \mid F\}$ can be safely replaced for values of type $T$, since they are just values of type $T$ further qualified by the information encoded by the formula $F$. The second rule, instead, generalizes the substitution principle underlying subtyping to the refinement formulas: for instance, we have $\emptyset;\varepsilon \vdash \{x : \mathsf{Un} \mid x = 5\} <: \{x : \mathsf{Un} \mid x > 0\}$, since the logical condition $x = 5$ is stronger than the condition $x > 0$.

A natural adaptation of (SUB REFINE RIGHT) to our affine setting would be:

$$\frac{\Gamma;\Delta_1 \vdash T <: U \qquad \Gamma, x:\psi(T);\Delta_2, \mathit{forms}(x:T) \vdash F \qquad \Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2}{\Gamma;\Delta \vdash T <: \{x:U \mid F\}}\ (\textsc{Sub Refine Wrong})$$

Unfortunately, this rule is unsound, since the affine formulas of $T$ could actually be duplicated and we could prove, for instance: $\emptyset;\varepsilon \vdash \{x : \mathsf{Un} \mid F\} <: \{z : \{x : \mathsf{Un} \mid F\} \mid F\}$ by using (SUB REFL) in the left premise of (SUB REFINE WRONG). This cannot happen with our new rule, since $F \nvdash F \otimes F$ in affine logic.

While it is in principle possible to find out other sound counterparts of (SUB REFINE RIGHT) in an affine setting, previous work [Bugliesi et al. 2011] highlighted that the technical treatment of these rules is rather complicated, and we find rule (SUB REFINE) more convenient for proofs. The previous discussion should have also provided an intuition on the reasons behind a slightly more restrictive treatment for subtyping pairs and functions with respect to the original RCF paper, i.e., we must take care in applying the refinement stripping function $\psi$ before extending the typing environment in the second premise of the corresponding rules.

### 6.5. Typing values

The typing judgement $\Gamma; \Delta \vdash M : T$ denotes that value $M$ is given type $T$ under environment $\Gamma; \Delta$. The typing rules for values are given in Table X.

**Table X** Typing rules for values

(VAL VAR)
$$\frac{\Gamma; \Delta \vdash \diamond \qquad (x : T) \in \Gamma}{\Gamma; \Delta \vdash x : T}$$

(VAL UNIT)
$$\frac{\Gamma; \Delta \vdash \diamond}{\Gamma; \Delta \vdash () : \mathsf{unit}}$$

(VAL FUN)
$$\frac{\Gamma, x : \psi(T); !\Delta', \mathit{forms}(x : T) \vdash E : U \qquad \Gamma; \Delta \hookrightarrow \Gamma; !\Delta'}{\Gamma; \Delta \vdash \lambda x. E : x : T \to U}$$

(VAL PAIR)
$$\frac{\Gamma; !\Delta_1 \vdash M : T \qquad \Gamma; !\Delta_2 \vdash N : U\{M/x\} \qquad \Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2}{\Gamma; \Delta \vdash (M, N) : x : T * U}$$

(VAL REFINE)
$$\frac{\Gamma; \Delta_1 \vdash M : T \qquad \Gamma; \Delta_2 \vdash F\{M/x\} \qquad \Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2}{\Gamma; \Delta \vdash M : \{x : T \mid F\}}$$

(VAL INL)
$$\frac{\Gamma; !\Delta' \vdash M : T \qquad \Gamma; !\Delta' \vdash U \qquad \Gamma; \Delta \hookrightarrow \Gamma; !\Delta'}{\Gamma; \Delta \vdash \mathsf{inl}\ M : T + U}$$

(VAL INR)
$$\frac{\Gamma; !\Delta' \vdash M : U \qquad \Gamma; !\Delta' \vdash T \qquad \Gamma; \Delta \hookrightarrow \Gamma; !\Delta'}{\Gamma; \Delta \vdash \mathsf{inr}\ M : T + U}$$

(VAL FOLD)
$$\frac{\Gamma; !\Delta' \vdash M : T\{\mu a. T/\alpha\} \qquad \Gamma; \Delta \hookrightarrow \Gamma; !\Delta'}{\Gamma; \Delta \vdash \mathsf{fold}\ M : \mu\alpha. T}$$

The rules for variable and unit typing are standard: variables are typed by looking up their type binding in the typing environment $\Gamma$ using (VAL VAR); the unit value can be given type unit under any well-formed environment using (VAL UNIT). Rule (VAL REFINE) is a natural adaptation to an affine setting of the standard rule for refinement types: a value $M$ has type $\{x : T \mid F\}$ if $M$ has type $T$ and the formula $F\{M/x\}$ holds true. Rules (VAL FUN) and (VAL PAIR) are more interesting: recall, in fact, that our type system does not include affine types, since the type information in $\Gamma$ is propagated to all the premises of a typing rule. It is then crucial for soundness that both pairs and functions are type-checked in an exponential environment, i.e., an environment of the form $!F_1, \ldots, !F_n$. Indeed, using an affine formula $F$ from the typing environment to give a pair $(M, N)$ type $x : T * \{y : U \mid F\}$ would lead to an unbounded duplication of $F$ upon repeated pair splitting operations on $(M, N)$. Similar restrictions apply also to sum types (cf. (VAL INL) and (VAL INR)) and iso-recursive types (cf. (VAL FOLD)).

Notice that allowing for affine refinements, but forbidding affine types, confines the problem of resource management to the formula environment $\Delta$, thus simplifying the technical development of the type system, as well as its algorithmic variant. In Section 6.8 we explain how our exponential serialization technique can be leveraged to encode affine types in our framework, hence our choice does not lead to any loss of expressiveness.

### 6.6. Typing expressions

The typing judgement $\Gamma; \Delta \vdash E : T$ denotes that expression $E$ is given type $T$ under environment $\Gamma; \Delta$. The typing rules for expressions are given in Table XI.

Several typing rules make use of the *extraction* relation $E \rightsquigarrow [\Delta \mid D]$ that destructively collects all the assumed formulas $\Delta$ from the expression $E$ and returns the expression $D$ obtained by purging $E$ of its assumptions. The relation is defined in Table XII and will be explained further in the context of rule EXP FORK.

Rule (EXP SUBSUM) is a standard subsumption rule for expressions: if $E$ can be given type $T$, then it can be conservatively given any supertype of $T$. The rule for typing function applications (EXP APPL) divides the formula environment $\Delta$ among its premises and checks that the type of the argument corresponds to the expected

**Table XI** Typing rules for expressions

(EXP SUBSUM)
$$\frac{\Gamma;\Delta_1 \vdash E : T \qquad \Gamma;\Delta_2 \vdash T <: T' \qquad \Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2}{\Gamma;\Delta \vdash E : T'}$$

(EXP APPL)
$$\frac{\Gamma;\Delta_1 \vdash M : x : T \to U \qquad \Gamma;\Delta_2 \vdash N : T \qquad \Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2}{\Gamma;\Delta \vdash M\ N : U\{N/x\}}$$

(EXP LET)
$$\frac{E \rightsquigarrow^{\emptyset} [\Delta' \mid D] \qquad \Gamma;\Delta_1 \vdash D : T \qquad \Gamma, x : \psi(T);\Delta_2, \mathit{forms}(x:T) \vdash E' : U \qquad x \notin \mathit{fv}(U) \qquad \Gamma;\Delta,\Delta' \hookrightarrow \Gamma;\Delta_1,\Delta_2}{\Gamma;\Delta \vdash \mathsf{let}\ x = E\ \mathsf{in}\ E' : U}$$

(EXP SPLIT)
$$\frac{\Gamma;\Delta_1 \vdash M : x : T * U \qquad \Gamma, x:\psi(T), y:\psi(U);\Delta_2, \mathit{forms}(x:T), \mathit{forms}(y:U), !((x,y) = M) \vdash E : V \qquad \{x,y\} \cap \mathit{fv}(V) = \emptyset \qquad \Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2}{\Gamma;\Delta \vdash \mathsf{let}\ (x,y) = M\ \mathsf{in}\ E : V}$$

(EXP MATCH)
$$\frac{\Gamma;\Delta_1 \vdash M : T \qquad \Gamma, x:\psi(H);\Delta_2, \mathit{forms}(x:H), !(h\ x = M) \vdash E : U \qquad \Gamma;\Delta_2 \vdash E' : U \qquad (h,H,T) \in \{(\mathsf{inl}, T_1, T_1 + T_2), (\mathsf{inr}, T_2, T_1+T_2), (\mathsf{fold}, T'\{\mu\alpha.T'/\alpha\}, \mu\alpha.T')\} \qquad \Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2}{\Gamma;\Delta \vdash \mathsf{match}\ M\ \mathsf{with}\ h\ x\ \mathsf{then}\ E\ \mathsf{else}\ E' : U}$$

(EXP EQ)
$$\frac{\Gamma;\Delta_1 \vdash M : T \qquad \Gamma;\Delta_2 \vdash N : U \qquad x \notin \mathit{fv}(M) \cup \mathit{fv}(N) \qquad \Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2}{\Gamma;\Delta \vdash M = N : \{x : \mathsf{bool} \mid !(x = \mathsf{true} \multimap M = N)\}}$$

(EXP ASSUME)
$$\frac{\Gamma;\Delta, F \vdash \mathsf{assume}\ \mathbf{1} : T \qquad F \neq \mathbf{1}}{\Gamma;\Delta \vdash \mathsf{assume}\ F : T}$$

(EXP TRUE)
$$\frac{\Gamma;\Delta \vdash \diamond}{\Gamma;\Delta \vdash \mathsf{assume}\ \mathbf{1} : \mathsf{unit}}$$

(EXP ASSERT)
$$\frac{\Gamma;\Delta \vdash F}{\Gamma;\Delta \vdash \mathsf{assert}\ F : \mathsf{unit}}$$

(EXP RES)
$$\frac{E \rightsquigarrow^{a} [\Delta' \mid D] \qquad \Gamma, a \updownarrow T;\Delta,\Delta' \vdash D : U \qquad a \notin \mathit{fn}(U)}{\Gamma;\Delta \vdash (\nu a)E : U}$$

(EXP SEND)
$$\frac{\Gamma;\Delta \vdash M : T \qquad (a \updownarrow T) \in \Gamma}{\Gamma;\Delta \vdash a!M : \mathsf{unit}}$$

(EXP RECV)
$$\frac{\Gamma;\Delta \vdash \diamond \qquad (a \updownarrow T) \in \Gamma}{\Gamma;\Delta \vdash a? : T}$$

(EXP FORK)
$$\frac{E_1 \rightsquigarrow^{\emptyset} [\Delta_1 \mid D_1] \qquad E_2 \rightsquigarrow^{\emptyset} [\Delta_2 \mid D_2] \qquad \Gamma;\Delta_1' \vdash D_1 : T_1 \qquad \Gamma;\Delta_2' \vdash D_2 : T_2 \qquad \Gamma;\Delta,\Delta_1,\Delta_2 \hookrightarrow \Gamma;\Delta_1',\Delta_2'}{\Gamma;\Delta \vdash E_1 \upharpoonright E_2 : T_2}$$

function argument type; in the return type we substitute the argument to the variable bound in the function type, thus implementing a form of value dependent typing. In rule (EXP SPLIT) we exploit the logic to keep track of the performed pair splitting operation and make type-checking more precise; a similar technique is used also in (EXP MATCH) and (EXP EQ). The treatment of channels is mostly standard: For each new channel $a$, a message type is determined ($a \updownarrow T$) and added to the typing environment $\Gamma$ (cf. EXP RES) that is used to type-check the remaining expression. The rules for sending (EXP SEND) and receiving (EXP RECV) messages on such channel assure that the sent/received messages have the correct type. Rule (EXP ASSERT) is standard and requires an asserted formula $F$ to be derivable from the formulas collected by the typing environment: in fact, these formulas under-approximate the formulas which will be assumed at runtime. As we will see in the explanation of the rule (EXP FORK) below, due to the affine nature of the logic, the treatment of assumptions is a delicate task. Assumptions can be typed using either rule (EXP TRUE) or (EXP ASSUME). The former describes the trivial case of a truth assumption $\mathbf{1}$ that is always given type $\mathsf{unit}$, the latter is used for more complex formulas $F$, which are added to the formula environment $\Delta$. Intuitively, the intended usage of these rules to type-check an assumption $\mathsf{assume}\ F$ with type $T$ is as follows: (1) Prove $\Gamma; \Delta, F \vdash \mathsf{assume}\ \mathbf{1} : \mathsf{unit}$ by rule (EXP TRUE); (2) Refine the type $\mathsf{unit}$ into $T$ by subtyping; (3) Use (EXP ASSUME) to conclude $\Gamma; \Delta \vdash \mathsf{assume}\ F : T$.

The most complex rule is (EXP FORK): intuitively, when type-checking the parallel expressions $E_1 \Rsh E_2$, assumptions in $E_1$ can be safely used to type-check assertions in $E_2$ and vice-versa. On the other hand, we need to prevent an affine assumption in $E_1$ from being used twice to justify assertions in both $E_2$ *and* $E_1$. This is achieved by the extraction relation, i.e., through the premises of the form $E_i \leadsto [\Delta_i \mid D_i]$: the extraction operation destructively collects all the assumptions from the expression $E_i$ and returns the expression $D_i$ obtained by purging $E_i$ of its assumptions. The typing environment is then extended with the collected assumptions and partitioned to type-check the purged expressions $D_1$ and $D_2$. For instance, we can show that the expression $\mathsf{assume}\ F \Rsh \mathsf{assert}\ F$ is well-typed, while the expression $(\mathsf{assume}\ F \Rsh \mathsf{assert}\ F) \Rsh \mathsf{assert}\ F$ is not: indeed, notice that the latter is not safe according to Definition 5.2.

The extraction relation $E \leadsto^{\widetilde{a}} [\Delta \mid D]$ is formally defined in Table XII. Note that we annotate the arrow with a list of names $\widetilde{a}$ to prevent formulas containing free names from being extracted outside the scope of the respective binders. For instance, in the expression $((\nu a)\mathsf{assume}\ F(a)) \Rsh \mathsf{assert}\ F(a)$ we do not want to use the assumption to type-check the parallel assertion, since the scope of the name $a$ is limited to the assumption itself. The extraction relation is used to type-check any expression possibly containing "active" assumptions, i.e., lets (cf. (EXP LET)), restrictions (cf. (EXP RES)), and assumptions themselves (cf. (EXP ASSUME), which hardcodes the extraction).

**Table XII** The extraction relation

(EXTR FORK)
$$\frac{E_1 \leadsto^{\widetilde{a}} [\Delta_1 \mid D_1] \qquad E_2 \leadsto^{\widetilde{a}} [\Delta_2 \mid D_2]}{E_1 \Rsh E_2 \leadsto^{\widetilde{a}} [\Delta_1, \Delta_2 \mid D_1 \Rsh D_2]}$$

(EXTR LET)
$$\frac{E_1 \leadsto^{\widetilde{a}} [\Delta \mid D_1]}{\mathsf{let}\ x = E_1\ \mathsf{in}\ E_2 \leadsto^{\widetilde{a}} [\Delta \mid \mathsf{let}\ x = D_1\ \mathsf{in}\ E_2]}$$

(EXTR RES)
$$\frac{E \leadsto^{a,\widetilde{b}} [\Delta \mid D]}{(\nu a)E \leadsto^{\widetilde{b}} [\Delta \mid (\nu a)D]}$$

(EXTR ASSUME)
$$\frac{F \neq \mathbf{1} \qquad fn(F) \cap \{\widetilde{a}\} = \emptyset}{\mathsf{assume}\ F \leadsto^{\widetilde{a}} [F \mid \mathsf{assume}\ \mathbf{1}]}$$

(EXTR EXP)
$$\frac{\text{no other rule applies}}{E \leadsto^{\widetilde{a}} [\emptyset \mid E]}$$

### 6.7. Formal results

The main soundness results for our type system are given below.

THEOREM 6.1 (SAFETY). *If* $\varepsilon; \emptyset \vdash E : T$*, then* $E$ *is safe.*

PROOF. See Appendix B. □

THEOREM 6.2 (ROBUST SAFETY). *If* $\varepsilon; \emptyset \vdash E : \mathsf{Un}$*, then* $E$ *is robustly safe.*

PROOF. See Appendix B. □

Theorem 6.2 above and Theorem 4.4 (Soundness of Serialization) constitute the two building blocks of our static verification technique, which we may finally summarize as follows. Given any expression $E$, we identify the payload formulas assumed in $E$, and construct their serializers $S_1, \ldots, S_n$. Let then $E^\star = \mathsf{assume}\ S_1 \otimes \cdots \otimes S_n \Rsh E$ be the original expression extended with the serializers. By Theorem 6.2, if $\varepsilon; \emptyset \vdash E^\star : \mathsf{Un}$, then $E^\star$ is robustly safe. By Theorem 4.4, so is the original expression $E$, provided that a further invariant holds for $E^\star$, namely that all multisets of formulas assumed during the evaluation of $E^\star$ are controlled.

While this latter invariant is not enforced by our type system, the desired guarantees may be achieved by requiring that the assumption of control formulas be confined within system code packaged into library functions, providing certified access and management of the capabilities associated with those formulas. The certification of the system code provided by the library function, in turn, may be achieved with limited effort, based on the sufficient condition provided by Proposition 4.5. Actually, we observe that the syntactic criterion proposed by the proposition becomes a *semantic* property of the program to type-check, since programs contain variables to be replaced at runtime: we will discuss for our examples how we verify that the typing environment satisfies the conditions required for robust safety.

### 6.8. Discussion: encoding affine types

We now discuss how we can take advantage of exponential serialization to encode affine types in our type system. For the sake of simplicity, we focus on the encoding of affine pairs, but the same ideas applies uniformly to other data types (i.e., tagged unions and iso-recursive types).

Consider the typing environment $\Gamma; \Delta \triangleq x : \mathsf{Un}, y : \mathsf{Un}; A(x), B(y)$. Standard refinement type systems [Bengtson et al. 2011] allow for the following type judgement:

$$\Gamma; \Delta \vdash (x, y) : \{x : \mathsf{Un} \mid A(x)\} * \{y : \mathsf{Un} \mid B(y)\}$$

If the formulas $A(x)$ and $B(y)$ are interpreted as affine resources, however, the previous type assignment is sound only as long as the pair $(x, y)$ can be split only once, since every application of rule (EXP SPLIT) for pair destruction introduces the formulas $A(x), B(y)$ into the typing environment of the continuation. Since our type system does not feature affine types and has no way to enforce a single deconstruction of a pair, it conservatively forbids the previous type judgement, in that the premises of rule (VAL PAIR) require an exponential typing environment.

Nevertheless, the following type judgement is allowed by our type system:

$$x : \mathsf{Un}, y : \mathsf{Un}; A(x), B(y), S_1, S_2 \vdash (x, y) : \{x : \mathsf{Un} \mid A'(x)\} * \{y : \mathsf{Un} \mid B'(y)\}$$

where $A'(x) \triangleq\ !(C_1(x) \multimap A(x))$ and $B'(y) \triangleq\ !(C_2(y) \multimap B(y))$ are the serialized variants of $A(x)$ and $B(y)$ respectively, while $S_1 \triangleq\ !\forall x.(A(x) \multimap A'(x))$ and $S_2 \triangleq\ !\forall y.(B(y) \multimap B'(y))$ are the corresponding serializers. Here, the main idea for type-checking is to appeal to environment rewriting to consume the affine formulas $A(x)$ and $B(y)$, and introduce their exponential counterparts $A'(x)$ and $B'(y)$ into the typing environment before assigning a type to the components of the pair. In fact, notice that we have:

$$x : \mathsf{Un}, y : \mathsf{Un}; A(x), B(y), S_1, S_2 \hookrightarrow x : \mathsf{Un}, y : \mathsf{Un}; A'(x), B'(y),$$

hence we can prove the following type judgement:

$$\frac{x : \mathsf{Un}, y : \mathsf{Un}; A'(x) \vdash x : \{x : \mathsf{Un} \mid A'(x)\} \qquad x : \mathsf{Un}, y : \mathsf{Un}; B'(y) \vdash y : \{y : \mathsf{Un} \mid B'(y)\}}{x : \mathsf{Un}, y : \mathsf{Un}; A(x), B(y), S_1, S_2 \vdash (x, y) : \{x : \mathsf{Un} \mid A'(x)\} * \{y : \mathsf{Un} \mid B'(y)\}}$$

The interesting point now is that the pair $(x, y)$ can be split arbitrarily often, but the affine formulas $A(x)$ and $B(y)$ can be retrieved at most once, as long as the control formulas $C_1(x)$ and $C_2(y)$ are assumed at most once in the application code. In this way, we recover the expressiveness provided by affine types. We actually even go beyond that, allowing for a liberal usage of the value itself, as opposed to enforcing the affine usage of any data structure which contains an affine component, as dictated by many earlier substructural frameworks (see [Fähndrich and DeLine 2002] for a thorough discussion on this point).

## 7. A LIBRARY FOR COMMUNICATION AND CRYPTOGRAPHY

In this section we describe the primitives for communication that we use throughout the examples in this work and discuss how we encode cryptography using sealing. We note that our encoding of both communication and cryptography benefits from the notion of exponential serialization: we will never use channels, references, or cryptographic operations directly for messages with affine refinements, but we instead rely on exponentially serialized versions of such refinements. Formally, our libraries build on so-called *exponential types* that do not carry an affine refinement and are defined in Table XIII. Since these types do not need to be protected from replication we can immediately leverage existing non-affine libraries [Bengtson et al. 2011].

For the sake of simplicity, the definitions of the necessary functions and types are parametric in a type variable $\gamma$ used to denote exponential types. We recall, however, that our system does not support full polymorphism, but we can recover its effects by replicating library code to specialize it to the different types we need. Most of the content of this section is taken from [Bengtson et al. 2011] and included for the reader's convenience to make the paper self-contained.

**Table XIII** Exponential types

$$T \text{ exponential if } \begin{cases} T \in \{\mathsf{unit}, \alpha\} & \\ U \text{ exponential} & \text{for } T = \{x : U \mid !F\} \\ T_1 \text{ exponential and } T_2 \text{ exponential} & \text{for } T = x : T_1 \to T_2 \\ T_1 \text{ exponential and } T_2 \text{ exponential} & \text{for } T = x : T_1 * T_2 \\ T_1 \text{ exponential and } T_2 \text{ exponential} & \text{for } T = T_1 + T_2 \\ U \text{ exponential} & \text{for } T = \mu\alpha.\,U \end{cases}$$

### 7.1. An encoding of channels and messaging

In RCF channels are not values, hence they cannot be shared dynamically among principals. That same effect may however be recovered with the following encoding of channels for messages of exponential type $\gamma$ (and the associated primitives for message passing).

We report both the communication interface and its implementation below.

**type** $\mathsf{Ch}(\gamma)$ = $(\gamma \to \mathsf{unit}) * (\mathsf{unit} \to \gamma)$
**val** *mkchan :* $\mathsf{unit} \to \mathsf{Ch}(\gamma)$
**val** *send :* $\mathsf{Ch}(\gamma) \to \gamma \to \mathsf{unit}$
**val** *recv :* $\mathsf{Ch}(\gamma) \to \gamma$

**let** *mkchan* = **fun** _ $\to$ (**new** *a*)(**fun** *x* $\to$ *a!x*, **fun** _ $\to$ *a?*)
**let** *send* = **fun** *c x* $\to$ **let** *(s, r)* = *c* **in** *s x*
**let** *recv* = **fun** *c* $\to$ **let** *(s, r)* = *c* **in** *r ()*

We note that references can be encoded analogously.

**type** $\mathsf{Ref}(\gamma)$ = $\mathsf{Ch}(\gamma)$
**val** *mkref :* $\gamma \to \mathsf{Ref}(\gamma)$
**val** *setref :* $\mathsf{Ref}(\gamma) \to \gamma \to \mathsf{unit}$
**val** *deref :* $\mathsf{Ref}(\gamma) \to \gamma$

**let** *mkref* = **fun** *x* → **let** *r* = *mkchan ()* **in** *send r x; r*
**let** *setref* = **fun** *r x* → **let** _ = *recv r* **in** *send r x*
**let** *deref* = **fun** *r* → **let** *x* = *recv r* **in** *send r x; x*

In the following we typically write "*r := v*" for "*setref r v*" and "*!r*" for "*deref r*".

Note that the code for dereferencing a reference will not type-check for types that are not exponential, since the value retrieved from the reference is used twice: it is stored back into the reference and returned. Without the serialization approach, we would thus need to change the implementation, for instance, by using destructive references that erase their content after a read.

### 7.2. A sealing-based encoding of cryptography

Formal cryptography can be encoded inside RCF in terms of *sealing* [Morris 1973; Sumii and Pierce 2007]. A *seal* for a type $T$ is a pair of functions: a sealing function $T \to \mathsf{Un}$ and an unsealing function $\mathsf{Un} \to T$. Intuitively, for symmetric cryptography, these functions model encryption and decryption operations, respectively. A payload of type $T$ can be sealed to type $\mathsf{Un}$ and sent over the untrusted network; conversely, a message retrieved from the network with type $\mathsf{Un}$ can be unsealed to its correct type $T$. This mechanism is implemented in terms of a list of pairs, which is stored in a global reference that can only be accessed using the sealing and unsealing functions. Upon sealing, the payload $p$ is paired with a fresh, public value $h$ (the *handle*) representing its sealed version, and the pair $(p, h)$ is stored in the list; conversely, the unsealing function looks for the handle $h$ in the list and returns the associated payload $p$.

Since for symmetric cryptography the possession of the key allows to perform both encryption and decryption operations, for such cryptographic schemes we identify the key with the seal, i.e., we give access to both the sealing and the unsealing functions to any owner of the key and we let $\mathsf{SymKey}(T) \triangleq (T \to \mathsf{Un}) * (\mathsf{Un} \to T)$. Different cryptographic primitives, like public key encryptions and signature schemes, can be encoded following the same recipe: for instance, since the owner of a signing key is typically able to verify her own signature, the sealing-based abstraction of a signing key may consist of both the sealing and the unsealing functions, and be given type $\mathsf{SigKey}(T) \triangleq (T \to \mathsf{Un}) * (\mathsf{Un} \to T)$. The corresponding verification key, instead, should comprise only the unsealing function and be given type $\mathsf{VerKey}(T) \triangleq \mathsf{Un} \to T$. The functions "*sign*" and "*verify*" introduced in Section 2 can then be straightforwardly implemented: *sign* $M$ $N$ just extracts the first component of $M$ and calls it with parameter $N$, while *verify* $M$ $N$ simply invokes $M$ with parameter $N$.

As stated above, another important benefit of exponential serialization is that we can immediately leverage the sealing-based cryptographic library proposed by Bengtson et al. [Bengtson et al. 2011], since we will define cryptographic operations to be performed only on messages of exponential type. Without the serialization approach, we would need to define a different implementation of the sealing/unsealing functions: namely, we would have to enforce that an affine payload is never extracted more than once from the list stored in the global reference, hence the dereferencing/unsealing function would have to remove the payload from the secret list. This would complicate the sealing-based abstraction of cryptography and require additional reasoning to justify its soundness [Backes et al. 2010b]. Instead, with our approach, the unsealing function does not need to be changed: we can invoke it an arbitrary number of times to retrieve the payload, but the associated refinements will be retrieved at most once through exponential serialization.

We give full details of the cryptographic API used throughout this paper (just the types, not the code) below.

$\mathbf{type}$ $\mathsf{Seal}(\gamma)$ = $(\gamma \to \mathsf{Un}) * (\mathsf{Un} \to \gamma)$
$\mathbf{type}$ $\mathsf{SealRef}(\gamma)$ = $\mathsf{Ref}(\mathsf{List}(\gamma * \mathsf{Un}))$

$\mathbf{val}$ $\mathit{mkseal}$ : $\mathsf{string} \to \mathsf{Seal}(\gamma)$
$\mathbf{val}$ $\mathit{seal}$ : $\mathsf{SealRef}(\gamma) \to \gamma \to \mathsf{Un}$
$\mathbf{val}$ $\mathit{unseal}$ : $\mathsf{SealRef}(\gamma) \to \mathsf{Un} \to \gamma$

$\mathbf{type}$ $\mathsf{SymKey}(\gamma)$ = $\mathit{Sym}$ $\mathbf{of}$ $\mathsf{Seal}(\gamma)$
$\mathbf{val}$ $\mathit{mksymkey}$ : $\mathsf{unit} \to \mathsf{SymKey}(\gamma)$
$\mathbf{val}$ $\mathit{sencrypt}$ : $\mathsf{SymKey}(\gamma) \to \gamma \to \mathsf{Un}$
$\mathbf{val}$ $\mathit{sdecrypt}$ : $\mathsf{SymKey}(\gamma) \to \mathsf{Un} \to \gamma$

$\mathbf{type}$ $\mathsf{SigKey}(\gamma)$ = $\mathit{SK}$ $\mathbf{of}$ $\mathsf{Seal}(\gamma)$
$\mathbf{type}$ $\mathsf{VerKey}(\gamma)$ = $\mathit{VK}$ $\mathbf{of}$ $(\mathsf{Un} \to \gamma)$
$\mathbf{val}$ $\mathit{mksigkey}$ : $\mathsf{unit} \to \mathsf{SigKey}(\gamma)$
$\mathbf{val}$ $\mathit{mkverkey}$ : $\mathsf{SigKey}(\gamma) \to \mathsf{VerKey}(\gamma)$
$\mathbf{val}$ $\mathit{sign}$ : $\mathsf{SigKey}(\gamma) \to \gamma \to \mathsf{Un}$
$\mathbf{val}$ $\mathit{verify}$ : $\mathsf{VerKey}(\gamma) \to \mathsf{Un} \to \gamma$

$\mathbf{type}$ $\mathsf{DecKey}(\gamma)$ = $\mathit{DK}$ $\mathbf{of}$ $\mathsf{Seal}(\gamma)$
$\mathbf{type}$ $\mathsf{EncKey}(\gamma)$ = $\mathit{EK}$ $\mathbf{of}$ $(\gamma \to \mathsf{Un})$
$\mathbf{val}$ $\mathit{mkdeckey}$ : $\mathsf{unit} \to \mathsf{DecKey}(\gamma)$
$\mathbf{val}$ $\mathit{mkenckey}$ : $\mathsf{DecKey}(\gamma) \to \mathsf{EncKey}(\gamma)$
$\mathbf{val}$ $\mathit{encrypt}$ : $\mathsf{EncKey}(\gamma) \to \gamma \to \mathsf{Un}$
$\mathbf{val}$ $\mathit{decrypt}$ : $\mathsf{DecKey}(\gamma) \to \mathsf{Un} \to \gamma$

## 8. EXAMPLE: EPMO

We are finally ready to see our type system at work. We consider a variant of *EPMO*, a nonce-based e-payment protocol proposed by Guttman et al. [Guttman et al. 2004].

### 8.1. Protocol description

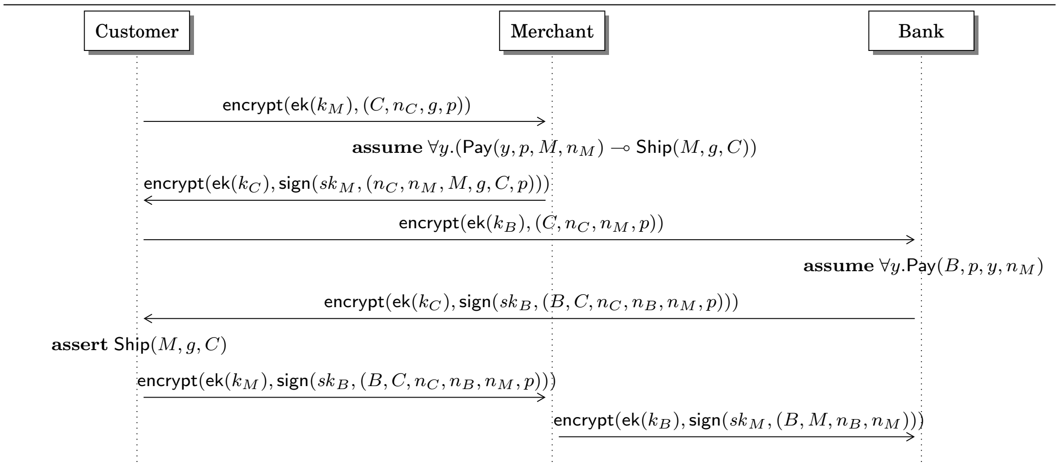
The protocol narration is informally represented in Table XIV (the meaning of the security annotations is explained below).

Initially, a customer $C$ contacts a merchant $M$ to buy some goods $g$ for a given price $p$; the request is encrypted under the public key of the merchant, $\mathsf{ek}(k_M)$ (which we use as shorthand for "$\mathit{mkenckey}\ k_M$" throughout the example), and includes a fresh nonce, $n_C$. If $M$ agrees to proceed in the transaction by providing a response signed with the signing key $sk_M$, $C$ informs her bank $B$ to authorize the payment. The bank replies by providing $C$ a receipt of authorization, called the *money order*, which is then forwarded to $M$. Now $M$ can verify that $C$ is entitled to pay for the goods and complete the transaction by sending a signed request to $B$ to cash the money order. At the end of the run, the bank transfers the funds and the merchant ships the goods to the customer.

### 8.2. Protocol analysis and challenges

A peculiarity of the protocol is that the identifier $n_C$ is employed by $C$ to authenticate *two* different messages, namely the replies by $M$ and $B$. This pattern cannot be validated by most existing type systems, since the mechanisms hardcoded therein to deal with nonce-handshakes enforce the freshness of each nonce to be checked only once. Our framework, instead, allows for a very natural treatment of such authentication pattern, whose implementation can be written mostly oblivious of the security verifi-

**Table XIV** A variant of the *EPMO* protocol




cation process, based on lightweight logical annotations. For the sake of simplicity, we focus only on the aspects of the verification connected to the guarantees provided to $C$, which are the most interesting ones.

We define two predicates used in the analysis: $\mathsf{Pay}(B, p, M, n_M)$ states that $B$ authorizes the payment $p$ to $M$ in reference to the order identified by $n_M$, while $\mathsf{Ship}(M, g, C)$ formalizes that $M$ will ship the goods $g$ to $C$. After the first step of the protocol, we let the merchant $M$ state the formula $\forall y.(\mathsf{Pay}(y, p, M, n_M) \multimap \mathsf{Ship}(M, g, C))$, to signify that she does not care about which bank is going to authorize the payment, but, as long as there is one authorizing bank, she will ship the good $g$ to the client $C$ at the end of the transaction. Conversely, after the appropriate checks on the client's account, we let the bank $B$ assume the formula $\forall y.\mathsf{Pay}(B, p, y, n_M)$, to model that she authorizes the payment for the transaction $n_M$ to any merchant chosen by the client. These two credentials allow the customer $C$ to assert the formula $\mathsf{Ship}(M, g, C)$, a formal assurance on the validity of the transaction.

### 8.3. Type-checking the customer

The protocol code for the customer, enriched with the most relevant type annotations and the necessary serializers, is shown below. For the sake of readability, we use again F#-like syntax and some standard syntactic sugar like tuples, refined tuple types, algebraic types, and pattern matchings: all these can be encoded in RCF and our type system using standard techniques [Bengtson et al. 2011].

*(∗ Serializer for M, needed to type-check M ∗)*
**assume** $!\forall xp, xM, xnM, xg, xC, xnC.$
$\quad (\forall y.(\mathsf{Pay}(y, xp, xM, xnM) \multimap \mathsf{Ship}(xM, xg, xC)) \multimap$
$\quad !(\mathsf{N1}(xnC) \multimap (\forall y.(\mathsf{Pay}(y, xp, xM, xnM) \multimap \mathsf{Ship}(xM, xg, xC))))$

*(∗ Serializer for B, needed to type-check B ∗)*
**assume** $!\forall yB, yp, ynC, ynM.$
$\quad (\forall y.(\mathsf{Pay}(yB, yp, y, ynM)) \multimap !(\mathsf{N2}(ynC) \multimap (\forall y.(\mathsf{Pay}(yB, yp, y, ynM))))$

*(∗ Typing the message from M to C ∗)*
**type** MsgMC = ***MsgMC*** **of** $(xnC$ : bytes $*$ $xnM$ : bytes $*$ $xM$ : string $*$ $xg$ : string

```
    ∗ xC : string ∗ xp : int)
    {!(N1(xnC) ⊸ ∀y.(Pay(y, xp, xM, xnM) ⊸ Ship(xM, xg, xC))}

(∗ Typing the message from B to C ∗)
type MsgBC = MsgBC of (yB : string ∗ yC : string ∗ ynC : bytes ∗ ynB : bytes
    ∗ ynM : bytes ∗ yp : int){!(N2(ynC) ⊸ ∀y.(Pay(yB, yp, y, ynM))}

(∗ Generate transaction identifiers ∗)
let mktid : unit → {x : bytes | N1(x) ⊗ N2(x)} = fun () →
    let xf = mkfresh () in assume (N1(xf) ⊗ N2(xf)); xf

(∗ Customer code ∗)
let cust C addC M addM B addB g p kC ekM ekB
        (vkM: VerKey(MsgMC + MsgMB) (vkB: VerKey(MsgBC)) =
    let nC = mktid () in
    (∗ N1(nC) and N2(nC) hold true ∗)
    let msgCM1 = encrypt ekM (C, nC, g, p) in send addM msgCM1;
    let signMC = decrypt kC (receive addC) in
    let plainMC = verify vkM signMC in
    match plainMC with MsgMC (=nC, xnM, =M, =g, =C, =p) →
        (∗ !(N1(nC) ⊸ ∀y.(Pay(y, p, M, xnM) ⊸ Ship(M, g, C)) holds true ∗)
        let msgCB = encrypt ekB (C, nC, xnM, p) in send addB msgCB;
        let signBC = decrypt kC (receive addC) in
        let plainBC = verify vkB signBC in
          match plainBC with MsgBC (=B, =C, =nC, xnB, =xnM, =p) →
            (∗ !(N2(nC) ⊸ ∀y.(Pay(B, p, y, xnM)) holds true ∗)
            assert Ship(M, g, C);
            let msgCM2 = encrypt ekM signBC in send addM msgCM2
```

Initially, we let the customer call the library function *mktid*, which generates a fresh transaction identifier, corresponding to $n_C$ in the protocol specification, and provides via its return type two distinct capabilities $\mathsf{N1}(nC)$ and $\mathsf{N2}(nC)$, later employed to authenticate the two different messages received by $C$.

Since the signing key of $M$ is used to certify messages of two different types (at steps 2 and 6 of the protocol), the corresponding verification key available to the customer through the variable *vkM* refers to a sum type. We present only the MsgMC component of this type, since it is the one needed to type-check the code of $C$: the refined formula in the corresponding type definition is retrieved upon verification of *signMC* and describes the promise by $M$ to ship the goods as soon as the requested payment has been authorized by any bank chosen by the client.

We finally use *vkB* to convey the other formula which is needed to type-check $C$, namely a statement that $B$ authorizes the payment to any merchant to whom $C$ wishes to transfer the money order: this statement is available after verifying the message *signBC*. The hypotheses collected by $C$ are enough to prove her assertion, i.e., to be sure that the request by $M$ has been fulfilled and the goods will be shipped, hence the implementation is well-typed.

Notice that, to conclude that the code actually respects the authorization policy despite the introduction of the serializers, we also have to show that the program ensures the invariant that each control formula is assumed at most once, corresponding to the sufficient condition for control dictated by Proposition 4.5. This is an easy task to carry out, since we can just observe that control formulas are only assumed in the body of the *mktid* function, which in turn only performs these assumptions over the results of the *mkfresh* function for the generation of fresh bitstrings.

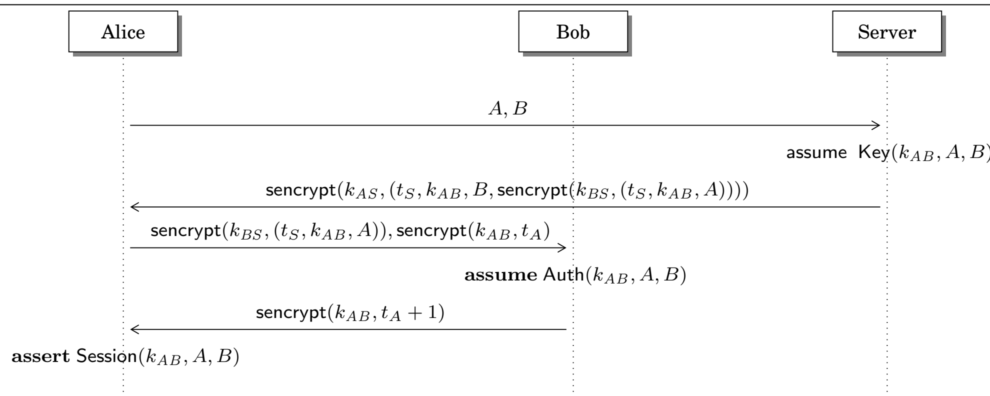
## 9. EXAMPLE: KERBEROS

In the *EPMO* protocol presented before, the nonce $n_C$ is checked twice by the customer $C$ and plays the role of a transaction identifier. Interestingly, there are protocols where these identifiers are not just checked multiple times, but also by different parties. This is exactly the case for the mutual authentication step of the Kerberos protocol [Steiner et al. 1988].

### 9.1. Protocol description

An informal narration of the protocol is shown in Table XV (the meaning of the security annotations is explained below).

**Table XV** The Kerberos protocol (mutual authentication)




The goal of the protocol is to establish a fresh session key $k_{AB}$ between principals $A$ and $B$ through a trusted server $S$, which shares a symmetric key with both $A$ and $B$. Kerberos employs timestamps like $t_S$ and $t_A$ to prove session recentness and protect against replay attacks. Initially, $A$ contacts the server $S$, providing the identities of the two agents $A$ and $B$ who want to establish a session. The server generates a fresh timestamp $t_S$ and a new session key $k_{AB}$, then it packages all this information into a message for $A$ and a message for $B$, which are combined by a nested encryption at step 2 of the protocol. Later, $A$ removes the outer layer of the encryption, checks $t_S$ and retrieves $k_{AB}$. If the timestamp is fresh, she forwards the inner encrypted message to $B$; additionally, $A$ includes a fresh timestamp $t_A$ encrypted under $k_{AB}$. Now $B$ can decrypt the message encrypted by $S$, check its freshness, and retrieve the session key $k_{AB}$. Using this key, $B$ can disclose the timestamp $t_A$ and reply to $A$ with $t_A + 1$, thus authenticating herself.

### 9.2. Protocol analysis and challenges

An intriguing point for our static verification technique is that the timestamp $t_S$ generated by the server is checked by both $A$ and $B$ to ensure that the session key $k_{AB}$ is fresh. As anticipated, this pattern is more sophisticated than the one we discussed for *EPMO*, but the expressiveness of our underlying affine logic framework allows for a simple encoding, discussed in the next section. For the sake of simplicity, in the following we will just focus on the verification of the initiator $A$.

We start by defining two predicates used in the analysis: $\mathsf{Key}(k_{AB}, A, B)$ states that $k_{AB}$ is a fresh symmetric key intended to establish a session between $A$ and $B$, while $\mathsf{Auth}(k_{AB}, A, B)$ formalizes that $B$ wishes to communicate with $A$ using key $k_{AB}$. Intuitively, these are the guarantees available to $A$ after steps 2 and 4 of the protocol, respectively: by combining these two assurances, $A$ can conclude that $k_{AB}$ is a fresh session key which can be safely used to communicate with $B$. We model this last information through the predicate $\mathsf{Session}(k_{AB}, A, B)$ and we formalize the previous deduction by assuming the authorization policy:

$$!\forall x, y, z.(\mathsf{Key}(x, y, z) \otimes \mathsf{Auth}(x, y, z) \multimap \mathsf{Session}(x, y, z)).$$

We next discuss how we can show the compliance of the protocol against the previous policy by refinement type-checking.

### 9.3. Implementing and typing timestamps

We turn our attention to the implementation. We build on a very simple library for timestamp management, that we allow the principals to access. We note that timestamps are modeled as monotonic counters. To guarantee the freshness of a timestamp in the case that the opponent executes the protocol function multiple times, we pair the counter with a global, instance-dependent, fresh random bitstring *rand* that is created at the beginning of the protocol specification using the function *mkfresh*. This usage of a random bitstring models the assumption that different sessions of Kerberos running in parallel will use different timestamps. Of course, we could consider more realistic and complicated implementations, but the following one suffices to convey the intuition about our methodology:

```
(* Typing a timestamp *)
type TStamp = TS of (bytes * int)

(* Increment a timestamp by 1 *)
let inc_ts t =
  match t with TS (rt, tt) →
  TS (rt, tt + 1)

(* Pick a fresh timestamp, based on the value stored in r *)
let get_ts r = fun () →
  r := inc_ts !r; !r

(* Check a timestamp t for freshness, based on the value stored in r *)
let check_ts r id t' =
  match !r with TS (rt, tt) →
  match t' with TS (=rt, tt') →
  if (tt' > tt) then
    r := t'; assume F(id, t')
  else
    failwith "not_a_fresh_timestamp"

(* The handle to access the two functions above *)
let init_ts rand glob id =
  let tss = !glob in
  let res = search tss id in
  match res with
  | Some(r) −> (get_ts r, check_ts r id)
```

```
| None    -> let newref = mkref TS (rand, 0) in
              glob := (id,newref)::tss; (get_ts newref, check_ts newref id)
```

Each principal stores the last received timestamp in a reference, created by an invocation to the function *init_ts*, described below. The function *inc_ts*: $\mathsf{TStamp} \to \mathsf{TStamp}$ is used to increment a timestamp by $1$, the function *get_ts*: $\mathsf{Ref}(\mathsf{TStamp}) \to \mathsf{unit} \to \mathsf{TStamp}$ is used to create fresh timestamps, and the dependent function *check_ts*: $\mathsf{Ref}(\mathsf{TStamp}) \to x : \mathsf{string} \to y : \mathsf{TStamp} \to \{\_ : \mathsf{unit} \mid \mathsf{F}(x, y)\}$ is used to check whether a received timestamp $y$ is fresh and can be used to deem timely a communication with the principal $x$. The code of the function performs a conditional branch: if the timestamp is fresh, it *assumes* the logical formula encoding such a fact; otherwise, it fails. The function **failwith** throws an exception, so it can be safely given the polymorphic type $\mathsf{string} \to \alpha$; as a consequence, *check_ts* can be given the previous dependent function type, whose refined return type provides the freshness assumption.

The function *init_ts* is more complicated. It takes three parameters: the global instantiation-specific nonce *rand*, the identity of a principal *id* and a global reference *glob*, containing a list of pairs *(id',r')*, where *id'* is the identity of a principal and *r'* is a reference containing the last timestamp presented by *id'* (*TS (rand, 0)* if none). The function starts by retrieving this list from *glob*, bounding it to *tss*, and then uses an auxiliary function *search* to detect if there exists an entry of the form *(id,r)* in *tss*. If this is the case, *init_ts* returns a pair of functions *(get_ts r, check_ts r id)*, which will allow the caller to get a fresh timestamp and to check the freshness of the timestamps received by $id$. If *id* has never presented a timestamp when *init_ts* is invoked, the function creates a fresh reference containing *TS (rand, 0)* and updates the list stored in the reference *glob* to preserve the expected invariant, then it returns again a pair of functions for timestamp management. This implementation ensures that different instances of a protocol participant with the same identity will share the same counter for timestamps, which is important to protect the protocol against replay attacks. The *init_ts* function has type:

$$\begin{array}{l}\mathsf{bytes} \to \mathsf{Ref}(\mathsf{List}(\mathsf{string} * \mathsf{TStamp})) \to x : \mathsf{string} \to \\ \quad ((\mathsf{unit} \to \mathsf{TStamp}) * (y : \mathsf{TStamp} \to \{\_ : \mathsf{unit} \mid \mathsf{F}(x, y)\})).\end{array}$$

### 9.4. Typing the session key using self-dependent key types

Before discussing the implementation of the principal $A$, we must first consider a subtle issue related to verification. We pointed out that, at step 4 of the protocol, $A$ must be able to infer that $k_{AB}$ has been previously authenticated by $B$. The problem for verification is that the formula $\mathsf{Auth}(k_{AB}, A, B)$ modeling this fact must be conveyed by the type of the key $k_{AB}$ itself, but neither the key $k_{AB}$ nor the two identifiers $A$ and $B$ occur in the *payload* of the last protocol message, hence we cannot predicate on them using dependent typing. While the problem of letting the payload of the key refer to the identifiers $A$ and $B$ can be solved quite easily, since the referred to identities are globally and publicly known, the problem of letting the payload of a key predicate over the key itself is more involved due to lexical scoping. We show how to devise an encoding to solve the problem of self-dependent key types, which is close in spirit to the session key treatment advocated in previous work [Bugliesi et al. 2012].

Here we rely on a sealing-based encoding, where the *self-dependent* key $k_{AB}$ consists of a *key identifier* $i_{AB}$ and a pair $k'_{AB}$ composed of the sealing and unsealing functions, thus having the form $k_{AB} = (i_{AB}, k'_{AB})$. The predicate $\mathsf{Auth}$ of the protocol refers to the identifier $i_{AB}$ of the key $k_{AB}$, i.e., we actually assume $\mathsf{Auth}(i_{AB}, A, B)$ rather than $\mathsf{Auth}(k_{AB}, A, B)$ as we were discussing in the previous informal overview. The link between each self-dependent key $k$ and its respective key identifier $i$ is logically modeled by the predicate $\mathsf{KeyIdent}(k, i)$, which holds true for all valid key-identifier pairs. The adapted authorization policy then looks as follows:

$$!\forall w, x, y, z.(\mathsf{Key}(x, y, z) \otimes \mathsf{KeyIdent}(x, w) \otimes \mathsf{Auth}(w, y, z) \multimap \mathsf{Session}(x, y, z)).$$

In the following we present the definition of our sealing-based library for the self-dependent session key $k_{AB}$. For presentation convenience, we make use of the following notation:

```
type MsgAB<x,y,z> = ({t : TStamp | !(F(y, t) ⊸ Auth(x, y, z))} + TStamp
```

to denote the (open) type MsgAB of the session key payload. Here, $x \in fv$(MsgAB<*x,y,z*>) refers to the key identifier, while $y, z \in fv$(MsgAB<*x,y,z*>) refer to the globally available public identifiers $A$ and $B$ respectively. Note that this type is a sum type, since the key $k_{AB}$ will be used by $B$ to encrypt a timestamp of type $(t : \mathsf{TStamp})\{!(\mathsf{F}(B, t) \multimap \mathsf{Auth}(x, y, z))\}$ and by $A$ to encrypt a non-refined timestamp of type TStamp (since we do not focus on the verification of $B$ here). The sealing-based library for the dependent key $k_{AB}$ shared between $A$ and $B$ is given below:

```
(* Closed type of the session key established by Kerberos.
   Here, w stands for the key identifier discussed above *)
type DSymKey = DSym of (w : string * ((MsgAB<w,A,B> → Un) *
                                      (Un → MsgAB<w,A,B>)))

(* Generate a fresh identifier *)
val new_fresh_id: unit → string

(* Create a new self-dependent key *)
let mkdepkey: unit → DSymKey = fun () →
  let id = new_fresh_id () in
    let s = mkseal "dsymkey" in
      DSym (id,s)

(* Get the key identifier corresponding to a self-dependent key *)
let get_key_ident k: (k : DSymKey → {x : string | !KeyIdent(k, x)}) =
    match k with DSym (x, _) → assume !KeyIdent(k, x); x

(* Self-dependent symmetric encryption function *)
let depencrypt x k m: (x : string → DSymKey → MsgAB<x,A,B> → Un) =
    match k with DSym (=x, (seal, _)) → seal m

(* Self-dependent symmetric decryption function *)
let depdecrypt x k c: (x : string → DSymKey → Un → MsgAB<x,A,B>) =
    match k with DSym (=x, (_, unseal)) → unseal c
```

In the function *mkdepkey* we call the existing seal creation function *mkseal*, which is used to generate a new seal that is paired with a fresh key identifier. Specifically, recall that we have:

```
type Seal(α) = (α → Un) * (Un → α)
val mkseal: string → Seal(α)
```

In the case of the key generation function *mkdepkey*, the placeholder $\alpha$ is replaced by the monomorphic type MsgAB<*id,A,B*>. Hence we must ensure that *id* is in scope when specializing the *mkseal* function.

Finally, we can briefly comment the other functions of our small library. The function *get_key_ident* extracts the identifier $i$ from a dependent key $k$ and tracks the logical dependence $\mathsf{KeyIdent}(k, i)$ through its refined return type. Contrary to standard sealing-based encryption and decryption, the functions *depencrypt* and *depdecrypt* take the key identifier as an additional argument and perform a pattern-matching operation to bridge the dependent typing allowed by pair splitting and the dependent typing enabled by the definition of these functions. In the syntax of types, the need for this pattern matching operation is made apparent by the occurrence of the same variable $x$ in both the function type of *depencrypt*/*depdecrypt* and the data type MsgAB<*x,A,B*>.

### 9.5. Type-checking the initiator

We finally have all the ingredients to discuss how the initiator $A$ is type-checked. The code of the principal looks as follows:

```
(* Authorization policy *)
assume !∀w,x,y,z.(Key(x,y,z) ⊗ KeyIdent(x,w) ⊗ Auth(w,y,z) ⊸ Session(x,y,z))

(* Typing the message from A to B, where MsgAB<x,y,z> will be
   closed by instantiating it in the definition of the session key type *)
type MsgAB<x,y,z> = MsgAB of {t : TStamp | !(F(y,t) ⊸ Auth(x,y,z))} + TStamp

(* Typing the session key established by Kerberos *)
type DSymKey = DSym of (w : string * ((MsgAB<w,A,B> → Un)*
                                      (Un → MsgAB<w,A,B>)))

(* Typing the message from S to A, where MsgSA<x> will be
   closed by instantiating it in the initiator function *)
type MsgSA<x> = MsgSA of (xts : TStamp * xkAB : DSymKey * xB : string * y : Un)
  {!(F(xB,xts) ⊸ Key(xkAB,x,xB))}

(* Initiator code, where rand is a fresh global bitstring and
glob denotes a global reference, which are both provided in the
protocol specification and are not under the control of the opponent *)
let initiator rand glob A addA B addB S addS (kAS: SymKey(MsgSA<A>)) =
    let (get_tsB, check_tsB) = init_ts rand glob B in
      send addS (A,B);
      let msgSA = receive addA in
      let plainSA = sdecrypt kAS msgSA in
      match plainSA with
        MsgSA (xts, xkAB, =B, y) →
          (* !(F(B,xts) ⊸ Key(xkAB,A,B)) holds true *)
          let _ = check_tsB xts in
          (* F(B,xts) holds true *)
          let tA = get_tsB () in
          let iAB = get_key_ident xkAB in
          (* !KeyIdent(xkAB,iAB) holds true *)
          let msgAB = depencrypt iAB xkAB tA in
          send addB (y,msgAB);
          let msgBA = receive addA in
          (* tA' = tA + 1 *)
          let tA' = inc_ts tA in
          let (=tA') = depdecrypt iAB xkAB msgBA in
```

*(* $!(\mathsf{F}(B, tA') \multimap \mathsf{Auth}(iAB, A, B))$ holds true *)*
**let** _ = *check_tsB tA'* **in**
*(* $\mathsf{F}(B, tA')$ holds true *)*
**assert** $\mathsf{Session}(xkAB, A, B)$

Decryption and pattern-matching introduce the guarded formulas needed to type-check the initiator, while invocations to the timestamp library extend the typing environment with the control formulas needed to retrieve the payload formulas of interest. Specifically, the initiator starts by creating the handle to the timestamp library through the call *init_ts B*, which returns the two functions *get_tsB* and *check_tsB*. The interesting point here is the type of *check_tsB*, i.e., $y : \mathsf{TStamp} \rightarrow \{\_ : \mathsf{unit} \mid \mathsf{F}(B, y)\}$, hence a successful call to this function allows for deeming a communication with $B$ as timely. To understand why the function is given that type, recall that *init_ts* has the following type:

$$\begin{array}{l}\mathsf{bytes} \rightarrow \mathsf{Ref}(\mathsf{List}(\mathsf{string} * \mathsf{TStamp})) \rightarrow x : \mathsf{string} \rightarrow \\ \quad ((\mathsf{unit} \rightarrow \mathsf{TStamp}) * (y : \mathsf{TStamp} \rightarrow \{\_ : \mathsf{unit} \mid \mathsf{F}(x, y)\})).\end{array}$$

and observe that *check_tsB* is obtained by projecting the second component of the pair returned by the call *init_ts rand glob B*. Now, the logical environment is populated as follows:

($i$) when $A$ decrypts the message from $S$ and performs pattern matching, we introduce the formula $!(\mathsf{F}(B, xts) \multimap \mathsf{Key}(xkAB, A, B))$, based on the type of the symmetric key $kAS$ : SymKey(MsgSA<A>);
($ii$) when $A$ calls the *check_tsB* function on the timestamp *xts* received by $S$, we introduce the formula $\mathsf{F}(B, xts)$, based on the typing discussed above;
($iii$) when $A$ calls the *get_key_ident* function on the self-dependent key *xkAB* shared with $B$, we introduce the formula $!\mathsf{KeyIdent}(xkAB, iAB)$, where *iAB* is the key identifier associated to *xkAB*;
($iv$) when $A$ decrypts the message from $B$ using the self-dependent key *xkAB* identified by *iAB*, we introduce the formula $!(\mathsf{F}(B, tA') \multimap \mathsf{Auth}(iAB, A, B))$, based on the type of the *depdecrypt* function associated to *xkAB*, where $tA'$ corresponds to $tA$ incremented by $1$;
($v$) finally, when $A$ calls the *check_tsB* function on the timestamp $tA'$ received by $B$, we introduce the formula $\mathsf{F}(B, tA')$, similarly to what we do at point ($ii$).

Using ($i$) and ($ii$), we can prove $\mathsf{Key}(xkAB, A, B)$, while using ($iv$) and ($v$) we can prove $\mathsf{Auth}(iAB, A, B)$. These two formulas, along with $!\mathsf{KeyIdent}(xkAB, iAB)$ at point ($iii$), allow to derive the assertion $\mathsf{Session}(xkAB, A, B)$ based on the underlying authorization policy, hence the initiator is well-typed.

To conclude that the protocol actually respects the authorization policy despite the introduction of the serializers, it is enough to ensure that $\mathsf{F}(B, t)$ is assumed at most once for any possible choice of $t$. To prove it, we must guarantee that at the beginning of the protocol specification function:

(1) the global fresh value *rand* is freshly generated using the function *mkfresh* that never generates the same value twice;
(2) the global reference *glob* storing the received timestamps is correctly instantiated to the empty list and is not provided by the opponent as an argument to the protocol specification function. We thus note that different participants running with identity $A$ share the same counter for timestamps management by construction of our library (cf. Section 9.3) and that each invocation to *check_tsB* always returns an assumption predicating over increasing values of $t$.

**Table XVI** Selected algorithmic rules for typing values and expressions

(VAL VAR ALG)

$$\frac{\Gamma \vdash_{\mathsf{alg}} \diamond \qquad (x : T) \in \Gamma}{\Gamma \vdash_{\mathsf{alg}} x : T; \mathbf{1}}$$

(VAL FUN ALG)

$$\frac{\Gamma, x : \psi(T) \vdash_{\mathsf{alg}} \overline{E} : U; F' \qquad \mathit{fnfv}(T) \subseteq \mathit{dom}(\Gamma) \cup \{x\}}{\Gamma \vdash_{\mathsf{alg}} \lambda x : T.\, \overline{E} : (x : T \to U); !\forall x.(\mathit{forms}(x : T) \multimap F')}$$

(VAL REF ALG)

$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{M} : T; F' \qquad \mathit{fnfv}(F) \subseteq \mathit{dom}(\Gamma) \cup \{x\}}{\Gamma \vdash_{\mathsf{alg}} \overline{M}_{\{x:\_ \mid F\}} : \{x : T \mid F\}; F' \otimes F\{M/x\}}$$

(VAL PAIR ALG)

$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{M} : T; F_1 \qquad \Gamma \vdash_{\mathsf{alg}} \overline{N} : U\{M/x\}; F_2}{\Gamma \vdash_{\mathsf{alg}} (\overline{M}, \overline{N}) : x : T * U; !F_1 \otimes !F_2}$$

(EXP LET ALG)

$$\frac{\overline{E} \rightsquigarrow^{\emptyset} [\Delta' \mid \overline{E'}] \qquad \Gamma \vdash_{\mathsf{alg}} \overline{E'} : T; F_1 \qquad \Gamma, x : \psi(T) \vdash_{\mathsf{alg}} \overline{D} : U; F_2 \qquad x \notin \mathit{fv}(U) \qquad \mathit{fnfv}(\Delta') \subseteq \mathit{dom}(\Gamma)}{\Gamma \vdash_{\mathsf{alg}} \mathsf{let}\ x = \overline{E}\ \mathsf{in}\ \overline{D} : U; \Delta' \multimap (F_1 \otimes \forall x.(\mathit{forms}(x : T) \multimap F_2))}$$

**Notation:** Here $E := \langle \overline{E} \rangle$ denotes the expression obtained from $\overline{E}$ by erasing all its typing annotations.

## 10. ALGORITHMIC TYPE-CHECKING

The type system presented in Section 6 includes several non-deterministic rules, which make it hard to implement an efficient decision procedure for typing. In this section we outline an algorithmic variant of the type system, which we prove sound and complete. We first focus on presenting the main intuitions behind the algorithmic type system design and then show the complete formalization.

### 10.1. Overview

While standard sources of non-determinism (like subtyping or refining value types) can be eliminated using type annotations, the rewriting of logical environments, which is the distinctive source of non-determinism of our system, is harder to deal with. The core idea underlying the algorithmic version of the type system is to dispense with the logical environment $\Delta$ and to construct bottom-up a single logical formula that characterizes all the proof obligations that would normally be introduced along the type derivation. In such a way, all the burden due to resource management can be shifted to an external affine logic theorem prover, which has to deal with this issue anyway.

More in detail, every typing judgement of the form $\Gamma; \Delta \vdash \mathcal{J}$ is matched by an algorithmic counterpart of the form $\Gamma \vdash_{\mathsf{alg}} \mathcal{J}; F$. Intuitively, typing an expression algorithmically constitutes of two steps:

(1) The expression (decorated with type annotations whenever needed) is type-checked using the algorithmic type system. This process is syntax-directed and fully deterministic, and in case of success yields *one* proof obligation $F$.
(2) The proof obligation is verified, e.g., using an external theorem prover.

If both steps succeed, then the expression is well-typed.

### 10.2. Key ideas

We illustrate the main ideas behind our algorithmic type system on some representative rules, shown in Table XVI. The algorithmic rules for kinding (cf. Section 10.4), subtyping (cf. Section 10.5), and typing the remaining values and expressions (cf. Section 10.6) follow along the same lines. For the sake of readability we often abuse notation and we let the multiset $F_1, \ldots, F_n$ stand for the formula $F_1 \otimes \ldots \otimes F_n$.

We first notice that, according to standard practice, we rely on typing annotations to deal with non-structural rules. Annotated terms and expressions are denoted by $\overline{M}$ and $\overline{E}$, respectively. Their syntax is given in Table XXI. The explicit erasure of all typing annotations of an expression is denoted by $\langle\overline{E}\rangle$. For instance, we explicitly annotate values that are expected to be given a refinement type (cf. (VAL REF ALG)) with the expected refinement $F$ and use annotations to assign an explicit argument type $T$ to functional values (cf. (VAL FUN ALG)). In this way, every possible syntactic form for expressions is matched by a single type rule and the selection of appropriate types and refinements does not rely on non-determinism.

We now exemplify the general concepts underlying our technique by contrasting the standard typing rule (VAL FUN) with its algorithmic counterpart (VAL FUN ALG). The main source of non-determinism in (VAL FUN) is the rewriting of $\Delta$ to $!\Delta'$. As previously mentioned, our goal is to dispense with logical environments and their rewriting, by collecting a single proof obligation that accounts for the proof obligations generated in the original type system. In the algorithmic version, the proof obligation obtained by giving $\lambda x : T.\,\overline{E}$ type $V = x : T \rightarrow U$ in the environment $\Gamma$ is:

$$!\forall x.(\mathit{forms}(x : T) \multimap F'),$$

where $F'$ is the proof obligation collected by giving $\overline{E}$ type $U$ in $\Gamma, x : \psi(T)$.

In the following, we briefly justify why this approach is sound, i.e., we argue why $\Gamma \vdash_{\mathsf{alg}} \lambda x : T.\,\overline{E} : V; !\forall x.(\mathit{forms}(x : T) \multimap F')$ implies that $\Gamma; \Delta \vdash \lambda x.\,\langle\overline{E}\rangle : V$ for any $\Delta$ such that $\Gamma; \Delta \vdash !\forall x.(\mathit{forms}(x : T) \multimap F')$. Notice that the latter judgement is equivalent to assuming that $\Delta$ entails $!\forall x.(\mathit{forms}(x : T) \multimap F')$ and both the multiset and the formula are well-formed with respect to $\Gamma$. Using the rules of the logic, we can show that a proof of $\Gamma; \Delta \vdash !\forall x.(\mathit{forms}(x : T) \multimap F')$ implies that there exists $\Delta'$ such that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$ and:

$$\Gamma; !\Delta' \vdash \forall x.\mathit{forms}(x : T) \multimap F'.$$

Intuitively, this means that we can eliminate the exponential modality by rewriting the logical environment in exponential form. Furthermore, the well-formedness of the (algorithmic) environment $\Gamma, x : \psi(T)$ and the (non-algorithmic) environment $\Gamma; !\Delta'$ ensures that $x \notin \mathit{dom}(\Gamma)$ and thus $x \notin \mathit{fv}(!\Delta')$: in this case, the logic allows us to further eliminate the universal quantification, adding a type binding for $x$ in order to keep the logical environment well-formed (the actual type is not relevant from the logic point of view). Thus, we have:

$$\Gamma, x : \psi(T); !\Delta' \vdash \mathit{forms}(x : T) \multimap F'.$$

Using rule ($\multimap$-LEFT), we can finally prove:

$$\Gamma, x : \psi(T); !\Delta', \mathit{forms}(x : T) \vdash F'.$$

By inductive reasoning, $\Gamma, x : \psi(T); !\Delta', \mathit{forms}(x : T) \vdash \langle\overline{E}\rangle : U$, hence (VAL FUN) allows us to derive $\Gamma; \Delta \vdash \lambda x.\,\langle\overline{E}\rangle : V$. The proof of completeness is similar.

The other algorithmic (typing) rules are constructed along the same lines, using the following additional observations:

— If a typing rule contains no kinding, subtyping, or typing premise (e.g., (VAL VAR)), the proof obligation of the corresponding algorithmic rule is set to $\mathbf{1}$ (cf. (VAL VAR ALG)) and thus trivially fulfilled.
— If a typing rule contains multiple premises (e.g., (VAL PAIR)), then we combine the proof obligations obtained from the premises conjunctively (cf. (VAL PAIR ALG)).
— If a typing rule relies on extraction (e.g., (EXP LET)) and adds the extracted environment $\Delta'$ to the environment before rewriting, the algorithmic variant of the rule (EXP

LET ALG) creates a proof obligation of the form $\Delta' \multimap F$, where $F$ is the proof obligation obtained by combining the proof obligations of the premises using the techniques described above.

With these insights in mind, we now show the complete formalization of the algorithmic type system.

### 10.3. Base judgements

The base judgements of the algorithmic type system are reported in Table XVII.

**Table XVII** Algorithmic well-formedness rules

(ENV EMPTY ALG)
$$\varepsilon \vdash_{\mathsf{alg}} \diamond$$

(TYPE ENV ENTRY ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \diamond \qquad dom(\mu) \cap dom(\Gamma) = \emptyset \qquad \mu = x : T \Rightarrow T = \psi(T) \wedge fnfv(T) \subseteq dom(\Gamma)}{\Gamma, \mu \vdash_{\mathsf{alg}} \diamond}$$

(TYPE ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \diamond \qquad fnfv(T) \subseteq dom(\Gamma)}{\Gamma \vdash_{\mathsf{alg}} T}$$

The only remarkable point here is that we do not have any algorithmic counterpart of rule (DERIVE). In fact, we never need to prove a formula in the algorithmic formulation of the type system, but we just collect the proof obligation for the external affine logic theorem prover.

### 10.4. Kinding

Table XVIII presents the algorithmic kinding rules. The non-inductive standard kinding rules (KIND VAR) and (KIND UNIT), which just check well-formedness of the environment (or environment membership) and which do not contain a proof obligation of the form $\Gamma; \Delta \vdash F$ amongst their hypotheses, are translated into algorithmic rules that generate the proof obligation $\mathbf{1}$. All other (recursive) rules (e.g., (KIND FUN)) strongly resemble their algorithmic counterparts (e.g., (KIND FUN ALG)). The proof obligation that is generated in the algorithmic variant consists of a conjunction of the proof obligations that are recursively generated by the premises of that rule, following the same principles of the algorithmic typing rules for values and expressions that we discussed in Section 10.2. Note that a premise that checks the well-formedness of an environment or type does not generate a proof obligation (cf. (KIND REFINE PUBLIC ALG)).

**Table XVIII** Algorithmic kinding rules

(KIND VAR ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \diamond \qquad (\alpha :: k) \in \Gamma}{\Gamma \vdash_{\mathsf{alg}} \alpha :: k; \mathbf{1}}$$

(KIND UNIT ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \diamond}{\Gamma \vdash_{\mathsf{alg}} \mathsf{unit} :: k; \mathbf{1}}$$

(KIND FUN ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} T :: \overline{k}; F_1 \qquad \Gamma, x : \psi(T) \vdash_{\mathsf{alg}} U :: k; F_2}{\Gamma \vdash_{\mathsf{alg}} x : T \to U :: k; !F_1 \otimes !F_2}$$

(KIND PAIR ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} T :: k; F_1 \qquad \Gamma, x : \psi(T) \vdash_{\mathsf{alg}} U :: k; F_2}{\Gamma \vdash_{\mathsf{alg}} x : T * U :: k; !F_1 \otimes !F_2}$$

(KIND SUM ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} T :: k; F_1 \qquad \Gamma \vdash_{\mathsf{alg}} U :: k; F_2}{\Gamma \vdash_{\mathsf{alg}} T + U :: k; !F_1 \otimes !F_2}$$

(KIND REC ALG)
$$\frac{\Gamma, \alpha :: k \vdash_{\mathsf{alg}} T :: k; F}{\Gamma \vdash_{\mathsf{alg}} \mu\alpha. T :: k; !F}$$

(KIND REFINE PUBLIC ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \{x : T \mid F\} \qquad \Gamma \vdash_{\mathsf{alg}} T :: \mathsf{pub}; F'}{\Gamma \vdash_{\mathsf{alg}} \{x : T \mid F\} :: \mathsf{pub}; F'}$$

(KIND REFINE TAINTED ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \psi(T) :: \mathsf{tnt}; F' \qquad \Gamma, x : \psi(T) \vdash_{\mathsf{alg}} \diamond \qquad T \text{ refined}}{\Gamma \vdash_{\mathsf{alg}} \{x : T \mid F\} :: \mathsf{tnt}; (\forall x. forms(x : T)) \otimes F'}$$

### 10.5. Subtyping

The algorithmic subtyping rules are presented in Table XX. They resolve the non-determinism related to the environment splitting by following the key insights of algorithmic typing presented in Section 10.2.

Furthermore, the algorithmic subtyping rules resolve the non-determinism that arises due to the fact that standard subtyping is not syntax-driven as described in the following. We use $T \neq_\top U$ to denote that $T$ and $U$ are not refined and do not share the same top-level constructor.

The algorithmic type system makes use of the following observation: for all non-refined types $T, U$ there are at most three standard subtyping rules applicable, namely (SUB REFL), (SUB PUB TNT), and in the case that $T$ and $U$ share the same top-level constructor one corresponding structural subtyping rule, e.g., (SUB FUN) or (SUB PAIR). In the case that $T$ or $U$ are refined, the three standard subtyping rules (SUB REFL), (SUB PUB TNT), or (SUB REFINE) might be applicable.

To reduce this level of non-determinism the algorithmic subtyping rules allow the reflexivity rule (SUB REFL ALG) to be applied only to the non-inductive type $\mathsf{unit}$ and type variables $\alpha$. Furthermore, we restrict the application of the kinding based rule (SUB PUB TNT ALG) to types $T, U$ that are structurally different and not refined, i.e., $T \neq_\top U$. Therefore, we can determine the appropriate subtyping rule by simple syntactic checks. Note that two types $T, U$, which share the same top-level constructor can still be subtyped using reflexivity or kinding by recursively applying the corresponding structural subtyping rule until one of the subgoals matches the premise of either the (SUB REFL ALG) or (SUB PUB TNT ALG) rule. Similarly, if either $T$ or $U$ or both are refined they can be typed using reflexivity or kinding by first applying the refinement rule (SUB REFINE ALG) and then applying either the (SUB REFL ALG) or (SUB PUB TNT ALG) rule to the subgoal.

This approach is sound and complete for all but the subtyping of two iso-recursive types. This is related to our choice of adapting the iso-recursive subtyping proposed by Backes et al. [Backes et al. 2011], which requires the recursive variable to occur only positively in the iso-recursive type, instead of the Amber rule (cf. (SUB POS REC) in Section 6.4). For instance, given the above constraints, subtyping $\Gamma \vdash_{\mathsf{alg}} \mu\alpha.\,(x : \alpha \to T) <: \mu\alpha.\,(x : \alpha \to T); F$ or $\Gamma \vdash_{\mathsf{alg}} \mu\alpha.\,(x : \alpha \to \mathsf{unit}) <: \mu\alpha.\,(x : \alpha \to \mathsf{unit} + \mathsf{unit}); F$ would not be possible, thus lacking reflexivity and kinding based algorithmic subtyping for iso-recursive types. Therefore, our algorithmic type system contains three rules for subtyping two iso-recursive types: (SUB REFL REC ALG), (SUB PUB TNT REC ALG), and (SUB POS REC ALG), respectively. While checking whether or not to apply rule (SUB REFL REC ALG) can be done by performing a simple equality check on the types, the decision between (SUB PUB TNT REC ALG) and (SUB POS REC ALG) requires some guidance, leading to the introduction of manual annotations of the form $\mathsf{SPT}$ to denote that the rule (SUB PUB TNT REC ALG) should be applied. This annotation appears in the subtyping rule for expressions (cf. (EXP SUBSUM ALG)), which we explain in Section 10.6.

The syntax of annotated types $\overline{T}$ is introduced in Table XIX. Intuitively, we allow type annotations $\mathsf{SPT}$ only on iso-recursive types and require them to not be nested. We let $\langle\overline{T}\rangle$ denote the explicit erasure of all annotations $\mathsf{SPT}$ from an annotated type $\overline{T}$. To facilitate readability we often write $T$ to denote the non-annotated counterpart $\langle\overline{T}\rangle$ of the annotated type $\overline{T}$. We can easily extend the definition of the function $\psi$ (used for the removal of top-level refinements) to annotated types.

**Table XIX** Syntax of annotated types and annotation erasure

| | | |
|---|---|---|
| $\overline{T}, \overline{U}, \overline{V}$ ::= | | *annotated types* |
| | $\mathsf{unit}$ | unit |
| | $\alpha$ | type variable |
| | $x : \overline{T} \to \overline{U}$ | dependent function type (scope of $x$ is $\overline{U}$) |
| | $x : \overline{T} * \overline{U}$ | dependent pair type (scope of $x$ is $\overline{U}$) |
| | $\overline{T} + \overline{U}$ | sum type |
| | $\mu\alpha.\overline{T}$ | iso-recursive type without top-level annotation |
| | $(\mu\alpha.T)_{\mathsf{SPT}}$ | iso-recursive type with top-level annotation |
| | $\{x : \overline{T} \mid F\}$ | refinement type (scope of $x$ is $F$) |

$$\psi(\overline{U}) = \begin{cases} \psi(\overline{T}) & \text{if } \overline{U} = \{x : \overline{T} \mid F\} \\ \overline{U} & \text{otherwise} \end{cases}$$

$$\langle\overline{U}\rangle = \begin{cases} \mathsf{unit} & \text{if } \overline{U} = \mathsf{unit} \\ \alpha & \text{if } \overline{U} = \alpha \\ x : \langle\overline{U_1}\rangle \to \langle\overline{U_2}\rangle & \text{if } \overline{U} = x : \overline{U_1} \to \overline{U_2} \\ x : \langle\overline{U_1}\rangle * \langle\overline{U_2}\rangle & \text{if } \overline{U} = x : \overline{U_1} * \overline{U_2} \\ \langle\overline{U_1}\rangle + \langle\overline{U_2}\rangle & \text{if } \overline{U} = \overline{U_1} + \overline{U_2} \\ \mu\alpha.\langle\overline{T}\rangle & \text{if } \overline{U} = \mu\alpha.\overline{T} \\ \mu\alpha.T & \text{if } \overline{U} = (\mu\alpha.T)_{\mathsf{SPT}} \\ \{x : \langle\overline{T}\rangle \mid F\} & \text{if } \overline{U} = \{x : \overline{T} \mid F\} \end{cases}$$

### 10.6. Typing values and expressions

The algorithmic typing rules for values and expressions are given in Table XXII and Table XXIII, respectively. The rules follow according to the intuition described in Section 10.2. We furthermore rely on type annotations to guide the selection of applicable typing rules and appropriate types. The syntax of annotated values and expressions is given in Table XXI. Here "_" is used to denote a type that is derived by the typing rules and thus does not need to be specified by the annotator. We denote the recursive erasure of all typing annotations by $\langle\overline{E}\rangle$ and often use $E$ to denote the expression $\langle\overline{E}\rangle$ obtained from the annotated expression $\overline{E}$ by erasing all its typing annotations. The extraction relation $\overline{E} \rightsquigarrow^{\emptyset} [\Delta \mid \overline{D}]$ for annotated expressions (cf. Table XXIV) extracts formulas as in the non-annotated case while keeping annotations on the expressions intact but for the case of assumptions, where it changes the type annotation in the original assumption to a subtyping annotation in the extracted assumption for all types different from $\mathsf{unit}$. The notions of free names and free variables correspond to the non-annotated case.

Since in the typing rule (VAL FUN) for functions the type of the input is chosen non-deterministically, we use the annotation $\lambda x : T. E$ to guide the algorithmic type system (VAL FUN ALG) in the selection of a suitable input type $T$. The annotation $M_{\{x:\_ \mid F\}}$ explicitly triggers the rule (VAL REF ALG) and expects $M$ to type-check with refinement $F$, while the annotations $(\mathsf{inl}\ M)_{\_+U}$ and $(\mathsf{inl}\ M)_{T+\_}$ are used to provide the respective missing type in the sum type $T + U$ that will be assigned to $\mathsf{inl}\ M$ and $\mathsf{inr}\ M$ (cf. (VAL INL ALG) and (VAL INR ALG)). Furthermore, the rule (EXP SUBSUM) is highly non-deterministic, since its application can be tried at any time using any combination of possible sub- and supertypes. In the algorithmic version of the type system we prevent the unnecessary application of subtyping and help the choice of an appropriate supertype $T'$ by annotating an expression $\overline{E}$ as $\overline{E}_{\_<:\overline{T'}}$ whenever subtyping is necessary

**Table XX** Algorithmic subtyping rules

(SUB REFL ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} T \qquad T \in \{\mathsf{unit}, \alpha\}}{\Gamma \vdash_{\mathsf{alg}} T <: T; \mathbf{1}}$$

(SUB PUB TNT ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} T :: \mathsf{pub}; F_1 \qquad \Gamma \vdash_{\mathsf{alg}} U :: \mathsf{tnt}; F_2 \qquad T \neq_{\top} U}{\Gamma \vdash_{\mathsf{alg}} T <: U; F_1 \otimes F_2}$$

(SUB FUN ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{T'} <: \overline{T}; F_1 \qquad \Gamma, x : \psi(T') \vdash_{\mathsf{alg}} \overline{U} <: \overline{U'}; F_2}{\Gamma \vdash_{\mathsf{alg}} x : \overline{T} \to \overline{U} <: x : \overline{T'} \to \overline{U'}; !F_1 \otimes !F_2}$$

(SUB PAIR ALG)
$$\frac{\begin{array}{c}\Gamma \vdash_{\mathsf{alg}} \overline{T} <: \overline{T'}; F_1 \\ \Gamma, x : \psi(T) \vdash_{\mathsf{alg}} \overline{U} <: \overline{U'}; F_2\end{array}}{\Gamma \vdash_{\mathsf{alg}} x : \overline{T} * \overline{U} <: x : \overline{T'} * \overline{U'}; !F_1 \otimes !F_2}$$

(SUB SUM ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{T} <: \overline{T'}; F_1 \qquad \Gamma \vdash_{\mathsf{alg}} \overline{U} <: \overline{U'}; F_2}{\Gamma \vdash_{\mathsf{alg}} \overline{T} + \overline{U} <: \overline{T'} + \overline{U'}; !F_1 \otimes !F_2}$$

(SUB POS REC ALG)
$$\frac{\Gamma, \alpha \vdash_{\mathsf{alg}} T <: T'; F \qquad \overline{T} \neq \overline{T'} \qquad \alpha \text{ occurs only positively in } \overline{T} \text{ and } \overline{T'}}{\Gamma \vdash_{\mathsf{alg}} \mu\alpha.\, \overline{T} <: \mu\alpha.\, \overline{T'}; !F}$$

(SUB REFL REC ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \mu\alpha.\, T}{\Gamma \vdash_{\mathsf{alg}} \mu\alpha.\, T <: \mu\alpha.\, T; \mathbf{1}}$$

(SUB PUB TNT REC ALG)
$$\frac{\begin{array}{c}\Gamma \vdash_{\mathsf{alg}} \mu\alpha.\, T :: \mathsf{pub}; F_1 \qquad \Gamma \vdash_{\mathsf{alg}} \mu\alpha.\, U :: \mathsf{tnt}; F_2 \\ s = \mathsf{SPT} \oplus s' = \mathsf{SPT}\end{array}}{\Gamma \vdash_{\mathsf{alg}} (\mu\alpha.\, T)_s <: (\mu\alpha.\, T')_{s'}; F_1 \otimes F_2}$$

(SUB REFINE ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \psi(\overline{T}) <: \psi(\overline{U}); F \qquad \overline{T} \text{ and/or } \overline{U} \text{ refined} \qquad \Gamma \vdash_{\mathsf{alg}} T \qquad \Gamma \vdash_{\mathsf{alg}} U}{\Gamma \vdash_{\mathsf{alg}} \overline{T} <: \overline{U}; F \otimes \forall y.(\mathit{forms}(y : T) \multimap \mathit{forms}(y : U))}$$

**Notation:** We write $T \neq_{\top} U$ to denote that $T$ and $U$ are not refined and do not share the same top-level constructor. $\oplus$ denotes the exclusive or. We use $T$ to denote the non-annotated counterpart $\langle \overline{T} \rangle$ of the annotated type $\overline{T}$.

(cf. (EXP SUBSUM ALG)). Note that the type $T'$ will additionally be annotated with SPT in case that the subtyping should make use of rule (SUB PUB TNT REC ALG), resulting in the annotated type $\overline{T'}$. Since the typing rule (EXP ASSUME) non-deterministically chooses a type $T$, its algorithmic counterpart (EXP ASSUME ALG) requires an explicit annotation of the form $(\mathsf{assume}\ F)_T$ to provide the expected type $T$.

### 10.7. Formal results

We can state and prove the following formal results, which highlight the correctness and the accuracy of the algorithmic type system.

THEOREM 10.1 (SOUNDNESS OF ALGORITHMIC TYPING). *If* $\Gamma \vdash_{\mathsf{alg}} \overline{E} : T; F$ *and* $\Gamma; \Delta \vdash F$*, then* $\Gamma; \Delta \vdash \langle \overline{E} \rangle : T$.

PROOF. See Appendix C. □

THEOREM 10.2 (COMPLETENESS OF ALGORITHMIC TYPING). *If* $\Gamma; \Delta \vdash E : T$*, then there exist* $\overline{E}, F$ *such that* $\langle \overline{E} \rangle = E$ *and* $\Gamma \vdash_{\mathsf{alg}} \overline{E} : T; F$ *and* $\Gamma; \Delta \vdash F$.

PROOF. See Appendix C. □

### 10.8. Example

The proof obligation assigned to the *cust* function in Section 8 by the algorithmic formulation of our type system is shown below:

$$\forall C.\forall M.\forall B.\forall g.\forall p.$$

**Table XXI** Syntax of annotated RCF expressions

| | | |
|---|---|---|
| $\overline{M}, \overline{N} ::=$ | | *values* |
| | $x$ | variable |
| | $()$ | unit |
| | $(\overline{M}, \overline{N})$ | pair |
| | $\lambda x : T. \overline{E}$ | annotated function with input of type $T$ |
| | $(\mathsf{inl}\ \overline{M})_{\_+T}$ | annotated left constructor |
| | $(\mathsf{inr}\ \overline{M})_{T+\_}$ | annotated right constructor |
| | $\mathsf{fold}\ \overline{M}$ | fold constructor |
| | $\overline{M}_{\{x:\_ \mid F\}}$ | value to be refined with $F$ |
| | $\overline{M}_{\_<:\overline{T}}$ | value to be subtyped to $\overline{T}$ |
| $\overline{D}, \overline{E} ::=$ | | *expressions* |
| | $\overline{M}$ | value |
| | $\overline{M}\ \overline{N}$ | application |
| | $\overline{M} = \overline{N}$ | syntactic equality |
| | $\mathsf{let}\ x = \overline{E}\ \mathsf{in}\ \overline{E}'$ | let (scope of $x$ is $\overline{E'}$) |
| | $\mathsf{let}\ (x, y) = \overline{M}\ \mathsf{in}\ \overline{E}$ | pair split (scope of $x, y$ is $\overline{E}$) |
| | $\mathsf{match}\ \overline{M}\ \mathsf{with}\ h\ x\ \mathsf{then}\ \overline{E}\ \mathsf{else}\ \overline{E'}$ | match (scope of $x$ is $\overline{E}$) |
| | $(\nu a)\overline{E}$ | restriction (scope of $a$ is $\overline{E}$) |
| | $\overline{E} \upharpoonright \overline{E'}$ | fork |
| | $a!\overline{M}$ | message send |
| | $a?$ | message receive |
| | $\mathsf{assume}\ 1$ | non-annotated truth assumption |
| | $(\mathsf{assume}\ F)_T$ | annotated assumption with expected type $T$ |
| | $\mathsf{assert}\ F$ | assertion |
| | $\overline{E}_{\_<:\overline{T}}$ | expression to be subtyped to $\overline{T}$ |

**Table XXII** Algorithmic typing rules for values

(VAL VAR ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \diamond \qquad (x : T) \in \Gamma}{\Gamma \vdash_{\mathsf{alg}} x : T; \mathbf{1}}$$

(VAL UNIT ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \diamond}{\Gamma \vdash_{\mathsf{alg}} () : \mathsf{unit}; \mathbf{1}}$$

(VAL FUN ALG)
$$\frac{\Gamma, x : \psi(T) \vdash_{\mathsf{alg}} \overline{E} : U; F' \qquad \mathit{fnfv}(T) \subseteq \mathit{dom}(\Gamma) \cup \{x\}}{\Gamma \vdash_{\mathsf{alg}} \lambda x : T. \overline{E} : x : T \to U; !\forall x.(\mathit{forms}(x : T) \multimap F')}$$

(VAL PAIR ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{M} : T; F_1 \qquad \Gamma \vdash_{\mathsf{alg}} \overline{N} : U\{M/x\}; F_2}{\Gamma \vdash_{\mathsf{alg}} (\overline{M}, \overline{N}) : x : T * U; !F_1 \otimes !F_2}$$

(VAL REF ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{M} : T; F' \qquad \mathit{fnfv}(F) \subseteq \mathit{dom}(\Gamma) \cup \{x\}}{\Gamma \vdash_{\mathsf{alg}} \overline{M}_{\{x:\_ \mid F\}} : \{x : T \mid F\}; F' \otimes F\{M/x\}}$$

(VAL INL ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{M} : T; F' \qquad \Gamma \vdash_{\mathsf{alg}} U}{\Gamma \vdash_{\mathsf{alg}} (\mathsf{inl}\ \overline{M})_{\_+U} : T + U; !F'}$$

(VAL INR ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{M} : U; F' \qquad \Gamma \vdash_{\mathsf{alg}} T}{\Gamma \vdash_{\mathsf{alg}} (\mathsf{inr}\ \overline{M})_{T+\_} : T + U; !F'}$$

(VAL FOLD ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{M} : T\{\mu a. T/\alpha\}; F'}{\Gamma \vdash_{\mathsf{alg}} \mathsf{fold}\ \overline{M} : \mu\alpha. T; !F'}$$

**Notation:** Here $M = \langle \overline{M} \rangle$ denotes the value obtained from $\overline{M}$ by erasing all its typing annotations.

$$\begin{array}{l}\forall nC.((\mathsf{N1}(nC) \otimes \mathsf{N2}(nC)) \multimap \\ \quad \forall xnM.(!(\mathsf{N1}(nC) \multimap (\forall y.\mathsf{Pay}(y, p, M, xnM) \multimap \mathsf{Ship}(M, g, C))) \multimap \\ \qquad\qquad !(\mathsf{N2}(nC) \multimap (\forall z.\mathsf{Pay}(B, p, z, xnM))) \multimap \\ \qquad\qquad\quad \mathsf{Ship}(M, g, C)))\end{array}$$

For the sake of readability we removed all unnecessary occurrences of $\mathbf{1}$ and all unused quantified variables. In this example, as well as in the other protocol we considered, the problem of solving equalities is reduced to the unification of variables. This allows us to use the `llprover` [Tomura 1995] theorem prover, which at the time of writing does not support equality theories. The above formula is discharged in less than $20$ ms.

**Table XXIII** Algorithmic typing rules for expressions

(EXP SUBSUM ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{E} : T; F_1 \qquad \Gamma \vdash_{\mathsf{alg}} T <: \overline{T'}; F_2}{\Gamma \vdash_{\mathsf{alg}} \overline{E}_{\_<:\overline{T'}} : T'; F_1 \otimes F_2}$$

(EXP APPL ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{M} : x : T \to U; F_1 \qquad \Gamma \vdash_{\mathsf{alg}} \overline{N} : T; F_2}{\Gamma \vdash_{\mathsf{alg}} \overline{M}\ \overline{N} : U\{N/x\}; F_1 \otimes F_2}$$

(EXP LET ALG)
$$\frac{\overline{E} \rightsquigarrow^{\emptyset} [\Delta' \mid \overline{E'}] \qquad \Gamma \vdash_{\mathsf{alg}} \overline{E'} : T; F_1 \qquad \Gamma, x : \psi(T) \vdash_{\mathsf{alg}} \overline{D} : U; F_2 \qquad x \notin \mathit{fv}(U) \qquad \mathit{fnfv}(\Delta') \subseteq \mathit{dom}(\Gamma)}{\Gamma \vdash_{\mathsf{alg}} \mathsf{let}\ x = \overline{E}\ \mathsf{in}\ \overline{D} : U; \Delta' \multimap (F_1 \otimes \forall x.(\mathit{forms}(x : T) \multimap F_2))}$$

(EXP SPLIT ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{M} : x : T * U; F_1 \qquad \Gamma, x : \psi(T), y : \psi(U) \vdash_{\mathsf{alg}} \overline{E} : V; F_2 \qquad \{x, y\} \cap \mathit{fv}(V) = \emptyset}{\Gamma; \Delta \vdash_{\mathsf{alg}} \mathsf{let}\ (x, y) = \overline{M}\ \mathsf{in}\ \overline{E} : V; F_1 \otimes \forall x.\forall y.(\mathit{forms}(x : T) \otimes \mathit{forms}(y : U) \otimes !((x, y) = M) \multimap F_2)}$$

(EXP MATCH ALG)
$$\frac{\begin{array}{c}\Gamma \vdash_{\mathsf{alg}} \overline{M} : T; F_1 \qquad \Gamma, x : \psi(H) \vdash_{\mathsf{alg}} \overline{E} : U; F_2 \qquad \Gamma; \Delta_2 \vdash_{\mathsf{alg}} \overline{D} : U; F_3 \\ (h, H, T) \in \{(\mathsf{inl}, T_1, T_1 + T_2), (\mathsf{inr}, T_2, T_1 + T_2), (\mathsf{fold}, T'\{\mu\alpha.T'/\alpha\}, \mu\alpha.T')\} \qquad \mathit{fnfv}(H) \subseteq \mathit{dom}(\Gamma) \cup \{x\}\end{array}}{\Gamma; \Delta \vdash_{\mathsf{alg}} \mathsf{match}\ \overline{M}\ \mathsf{with}\ h\ x\ \mathsf{then}\ \overline{E}\ \mathsf{else}\ \overline{D} : U; F_1 \otimes \forall x.(\mathit{forms}(x : H) \otimes !(h\ x = M) \multimap F_2) \otimes F_3}$$

(EXP EQ ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{M} : T; F_1 \qquad \Gamma \vdash_{\mathsf{alg}} \overline{N} : U; F_2 \qquad x \notin (\mathit{fv}(\overline{M}) \cup \mathit{fv}(\overline{N}))}{\Gamma \vdash_{\mathsf{alg}} \overline{M} = \overline{N} : \{x : \mathsf{bool} \mid !(x = \mathsf{true} \multimap M = N)\}; F_1 \otimes F_2}$$

(EXP ASSUME ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} (\mathsf{assume}\ \mathbf{1})_{\_<:T} : T; F' \qquad F \neq \mathbf{1} \qquad \mathit{fnfv}(F) \subseteq \mathit{dom}(\Gamma)}{\Gamma \vdash_{\mathsf{alg}} (\mathsf{assume}\ F)_T : T; F \multimap F'}$$

(EXP TRUE ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \diamond}{\Gamma \vdash \mathsf{assume}\ \mathbf{1} : \mathsf{unit}; \mathbf{1}}$$

(EXP ASSERT ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \diamond \qquad \mathit{fnfv}(F) \subseteq \mathit{dom}(\Gamma)}{\Gamma \vdash_{\mathsf{alg}} \mathsf{assert}\ F : \mathsf{unit}; F}$$

(EXP RES ALG)
$$\frac{\overline{E} \rightsquigarrow^{a} [\Delta' \mid \overline{E'}] \qquad \Gamma, a \updownarrow T \vdash_{\mathsf{alg}} \overline{E'} : U; F \qquad a \notin \mathit{fn}(U) \qquad \mathit{fnfv}(\Delta') \subseteq \mathit{dom}(\Gamma)}{\Gamma \vdash_{\mathsf{alg}} (\nu a \updownarrow T)\overline{E} : U; \Delta' \multimap F}$$

(EXP SEND ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \overline{M} : T; F \qquad (a \updownarrow T) \in \Gamma}{\Gamma \vdash_{\mathsf{alg}} a!\overline{M} : \mathsf{unit}; F}$$

(EXP RECV ALG)
$$\frac{\Gamma \vdash_{\mathsf{alg}} \diamond \qquad (a \updownarrow T) \in \Gamma}{\Gamma \vdash_{\mathsf{alg}} a? : T; \mathbf{1}}$$

(EXP FORK ALG)
$$\frac{\begin{array}{c}\overline{E_1} \rightsquigarrow^{\emptyset} [\Delta_1 \mid \overline{D_1}] \\ \overline{E_2} \rightsquigarrow^{\emptyset} [\Delta_2 \mid \overline{D_2}] \qquad \Gamma \vdash_{\mathsf{alg}} \overline{D_1} : T_1; F_A \qquad \Gamma \vdash_{\mathsf{alg}} \overline{D_2} : T_2; F_B \qquad \mathit{fnfv}(\Delta_1, \Delta_2) \subseteq \mathit{dom}(\Gamma)\end{array}}{\Gamma \vdash_{\mathsf{alg}} \overline{E_1} \upharpoonright \overline{E_2} : T_2; (\Delta_1, \Delta_2) \multimap (F_A \otimes F_B)}$$

**Notation:** Here $E = \langle \overline{E} \rangle$ denotes the expression obtained from $\overline{E}$ by erasing all its typing annotations.

## 11. RELATED WORK

Several papers develop type systems for (variants of) RCF [Bhargavan et al. 2010; Bengtson et al. 2011; Fournet et al. 2011; Backes et al. 2011; Swamy et al. 2011] but, with the exception of F* [Swamy et al. 2011], they do not support resource-aware policies: in fact, even for simple linearity properties like injective agreement they rely on hand-written proofs [Bhargavan et al. 2009].

F* [Swamy et al. 2011] is a dependently typed functional language for secure distributed programming, featuring refinement types to reason about authorization poli-

**Table XXIV** The extraction relation for annotated expressions

(EXTR FORK)
$$\frac{\overline{E_1} \rightsquigarrow^{\widetilde{a}} [\Delta_1 \mid \overline{D_1}] \qquad \overline{E_2} \rightsquigarrow^{\widetilde{a}} [\Delta_2 \mid \overline{D_2}]}{(\overline{E_1} \upharpoonright \overline{E_2})_s \rightsquigarrow^{\widetilde{a}} [\Delta_1, \Delta_2 \mid (\overline{D_1} \upharpoonright \overline{D_2})_s]}$$

(EXTR LET)
$$\frac{\overline{E_1} \rightsquigarrow^{\widetilde{a}} [\Delta \mid \overline{D_1}]}{(\mathsf{let}\ x = \overline{E_1}\ \mathsf{in}\ \overline{E_2})_s \rightsquigarrow^{\widetilde{a}} [\Delta \mid (\mathsf{let}\ x = \overline{D_1}\ \mathsf{in}\ \overline{E_2})_s]}$$

(EXTR RES)
$$\frac{\overline{E} \rightsquigarrow^{a,\widetilde{b}} [\Delta \mid \overline{D}]}{((\nu a)\overline{E})_s \rightsquigarrow^{\widetilde{b}} [\Delta \mid ((\nu a)\overline{D})_s]}$$

(EXTR ASSUME UNIT)
$$\frac{F \neq \mathbf{1} \qquad fn(F) \cap \{\widetilde{a}\} = \emptyset}{((\mathsf{assume}\ F)_{\mathsf{unit}})_s \rightsquigarrow^{\widetilde{a}} [F \mid (\mathsf{assume}\ \mathbf{1})_s]}$$

(EXTR ASSUME)
$$\frac{F \neq \mathbf{1} \qquad fn(F) \cap \{\widetilde{a}\} = \emptyset \qquad T \neq \mathsf{unit}}{((\mathsf{assume}\ F)_T)_s \rightsquigarrow^{\widetilde{a}} [F \mid ((\mathsf{assume}\ \mathbf{1})_{\_<:T})_s]}$$

(EXTR EXP)
$$\frac{\text{no other rule applies}}{\overline{E} \rightsquigarrow^{\widetilde{a}} [\emptyset \mid \overline{E}]}$$

**Remark:** Note that here $s$ is either SPT or $\epsilon$ (i.e., no annotation).

cies and affine types to reason about stateful computations on affine *values*. Similarly to companion proposals for RCF, however, the type system of F* assumes the existence of the contraction rule in the underlying logic, hence it does not support authorization policies built over affine *formulas*. While some simple authentication patterns (e.g., basic nonce handshakes) may certainly be expressed by encoding affine predicates in terms of affine values, other more complex authentication mechanisms are much harder to handle in these terms. The *EPMO* protocol we analyze in Section 8 provides one such case, as $(i)$ the nonce it employs may not be construed as an affine value because it is used twice, and $(ii)$ the logical formulas justified by cryptographic message exchanges are more structured than simple predicates. Though it might be possible to come up with sophisticated encodings of these authentication mechanisms in the programming language (by resorting to, e.g., pairs of affine tokens to encode a double usage of the same nonce and special functions to eliminate logical implications), such encodings are hard to formulate in a general manner and, we argue, are much better expressed in terms of policy annotations than in some ad-hoc programming pattern.

Bhargavan et al. [Bhargavan et al. 2008] propose a technique for the verification of F# protocol implementations by automatically extracting ProVerif models [Blanchet 2001], using an extension of the functions-as-processes encoding proposed by Milner [Milner 1992]. Remarkably, the analysis can deal with injective agreement. On the other hand, the analysis carried out with ProVerif is not modular and has been shown less robust and scalable than type-checking [Bhargavan et al. 2010]. Furthermore, the fragment of F# considered is rather restrictive: for instance, it does not include higher-order functions and admits only very limited uses of recursion and state.

A formal account on the integration of refinement types and substructural logics was first proposed by Mandelbaum et al. [Mandelbaum et al. 2003] with a system for local reasoning about program state built around a fragment of intuitionistic linear logic. Later, Bierhoff and Aldrich developed a framework for modular type-state checking of object-oriented programs [Bierhoff and Aldrich 2007; Sunshine et al. 2011; Naden et al. 2012]. However, none of these systems deals with the presence of hostile (or untyped) program components, or attackers, a feature that is instead distinctive of our system: adapting the previous frameworks to take into account interactions with an untyped context would require fundamental changes to their typing rules. The original RCF type-checker [Bengtson et al. 2011], for instance, employs a security-oriented kinding relation to reason about messages sent to and received from the attacker, which we also adopt in our type system. Recent variants of the RCF type-checker dispense with the kinding relation and even with concurrency [Swamy et al. 2011], but they rely on manually proven logical invariants capturing security properties of the cryptographic library and, in some cases, of the protocol itself.

Tov and Pucella [Tov and Pucella 2010] have recently shown how to use behavioral contracts to link code written in an affine language to code written in a conventionally typed language. The idea is to coerce affine values to non-affine ones that can be shared with the context, but can still be reasoned about safely using dynamic access counts. There are intriguing similarities between this approach and the usage of nonces and session keys to enforce linearity properties in an adversarial setting, which are worth to be investigated in the future. The two type systems are, however, fundamentally different, since our present work deals with an affine refinement logic and an adversarial setting, which makes a precise comparison hard to formulate.

Various techniques have been proposed to statically analyze authenticity properties of cryptographic protocols [Armando et al. 2005; Backes et al. 2007a; Cremers 2008; Backes et al. 2008; Meier et al. 2013; Blanchet 2011; Backes et al. 2008], among which several types and effects systems [Gordon and Jeffrey 2003; 2004; Bugliesi et al. 2004b; Maffei 2004; Bugliesi et al. 2004a; 2005; Maffei 2005; Bugliesi et al. 2007; Backes et al. 2007b; Backes et al. 2009; Backes et al. 2010a; Focardi and Maffei 2011]. These type systems incorporate ad-hoc mechanisms to deal with nonce handshakes and, thus, to enforce injective agreement properties. Our exponential serialization technique can be seen as a logic-based generalization of such mechanisms, independent of the language and the type system. As a consequence, our type system is similarly able to verify authenticity in terms of injective agreement, while allowing for expressing also a number of more sophisticated properties involving access counts and usage bounds. As a downside, the current formulation of our type system does not allow to validate some specific nonce-handshake idioms, like the SOSH scheme [Gordon and Jeffrey 2004]. Still, this can be recovered by extending our type system with union and intersection types, as shown in [Backes et al. 2011; 2014].

In previous work [Bugliesi et al. 2011; 2012], we made initial steps towards the design of a sound system for resource-sensitive authorization, drawing on techniques from type systems for authentication and an affine extension of existing refinement type systems for the applied pi-calculus [Abadi and Fournet 2001]. That work aims at analyzing cryptographic protocols as opposed to their implementations. Furthermore, such a type system is designed around a specific cryptographic library: the consequence is that extending the analysis to new primitives requires significant changes in the soundness proof of the type system. In contrast, the usage of a $\lambda$-calculus in this work allows us to encode cryptography in the language using a standard sealing mechanism (cf. Section 7.2), which makes the analysis technique easily extensible to new cryptographic primitives. Finally, the non-standard nature of our previous type system makes it difficult to devise an efficient algorithmic variant, which in turn can be cleanly designed for the present proposal.

## 12. CONCLUSIONS

We presented the first type system for statically enforcing the (robust) safety of cryptographic protocol implementations with respect to authorization policies expressed in affine logic. Our type system benefits from the novel concept of exponential serialization to achieve a general and flexible treatment of affine formulas in distributed systems: we showed the effectiveness of this technique on two existing cryptographic protocols. We finally proposed an efficient, sound, and complete algorithmic variant of the type system, which is the key for a practical implementation of our analysis technique.

We are currently working on the mechanization of our theory by implementing a type-checker based on the algorithmic typing rules. We plan to facilitate type-checking and reduce the need for manual type annotations by taking advantage of recent research on type inference in intuitionistic linear logic [Baillot and Hofmann 2010].

**Acknowledgments.** This work was supported by the German research foundation (DFG) through the Emmy Noether program, the German Federal Ministry of Education and Research (BMBF) through the Center for IT-Security, Privacy and Accountability (CISPA), and by the Italian ministry for university and research (MIUR) through the ADAPT and CINA projects.

## REFERENCES

Martín Abadi and Cédric Fournet. 2001. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL)*. ACM Press, 104–115.

Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. 2005. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proc. Computer Aided Verification'05 (CAV) (Lecture Notes in Computer Science)*. 281–285.

Michael Backes, Agostino Cortesi, Riccardo Focardi, and Matteo Maffei. 2007b. A Calculus of Challenges and Responses. In *Proc. 5th ACM Workshop on Formal Methods in Security Engineering (FMSE)*. ACM Press, 101–116.

Michael Backes, Agostino Cortesi, and Matteo Maffei. 2007a. Causality-based Abstraction of Multiplicity in Cryptographic Protocols. In *Proc. 20th IEEE Symposium on Computer Security Foundations (CSF)*. IEEE Computer Society Press, 355–369.

Michael Backes, Martin P. Grochulla, Catalin Hritcu, and Matteo Maffei. 2009. Achieving Security Despite Compromise Using Zero-knowledge. In *Proc. 22nd IEEE Symposium on Computer Security Foundations (CSF)*. IEEE Computer Society Press, 308–323.

Michael Backes, Cătălin Hriţcu, and Matteo Maffei. 2008. Type-checking Zero-knowledge. In *15th ACM Conference on Computer and Communications Security (CCS)*. ACM Press, 357–370.

Michael Backes, Catalin Hriţcu, and Matteo Maffei. 2011. Union and Intersection Types for Secure Protocol Implementations. In *Proc. Theory of Security and Applications (TOSCA) (Lecture Notes in Computer Science)*. Springer-Verlag, 1–28.

Michael Backes, Catalin Hriţcu, and Matteo Maffei. 2014. Union and Intersection Types for Secure Protocol Implementations. *Journal of Computer Security* 22, 2 (2014), 301–353.

Michael Backes, Stefan Lorenz, Matteo Maffei, and Kim Pecina. 2008. The CASPA Tool: Causality-Based Abstraction for Security Protocol Analysis. In *Proc. Computer Aided Verification'08 (CAV) (Lecture Notes in Computer Science)*. Springer-Verlag, 419–422.

Michael Backes, Matteo Maffei, and Esfandiar Mohammadi. 2010a. Computationally Sound Abstraction and Verification of Secure Multi-Party Computations. In *Proc. IARCS Annual Conference on on Foundations of Software Technology and Theoretical Computer Science (FSTTCS) (LIPIcs)*, Vol. 8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 352–363.

Michael Backes, Matteo Maffei, and Dominique Unruh. 2010b. Computationally Sound Verification of Source Code. In *Proc. 17th ACM Conference on Computer and Communications Security (CCS)*. ACM Press, 387–398.

Patrick Baillot and Martin Hofmann. 2010. Type Inference in Intuitionistic Linear Logic. In *Proc. 12th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP)*. ACM Press, 219–230.

Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. 2011. Refinement types for secure implementations. *ACM Transactions on Programming Languages and Systems* 33, 2, Article 8 (2011), 45 pages.

Karthikeyan Bhargavan, Ricardo Corin, Pierre-Malo Deniélou, Cédric Fournet, and James J. Leifer. 2009. Cryptographic Protocol Synthesis and Verification for Multiparty Sessions. In *Proc. 22nd IEEE Symposium on Computer Security Foundations (CSF)*. IEEE Computer Society Press, 124–140.

Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. 2010. Modular Verification of Security Protocol Code by Typing. In *Proc. 37th Symposium on Principles of Programming Languages (POPL)*. ACM Press, 445–456.

Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Stephen Tse. 2008. Verified Interoperable Implementations of Security Protocols. *ACM Transactions on Programming Languages and Systems* 31, 1, Article 5 (2008), 61 pages.

Kevin Bierhoff and Jonathan Aldrich. 2007. Modular typestate checking of aliased objects. In *Proc. 22nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications (OOPSLA)*. ACM Press, 301–320.

Bruno Blanchet. 2001. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules.. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*. IEEE Computer Society Press, 82–96.

Bruno Blanchet. 2011. Using Horn Clauses for Analyzing Security Protocols. In *Formal Models and Techniques for Analyzing Security Protocols*. Cryptology and Information Security, Vol. 5. IOS Press, Chapter 7, 86–111.

Michele Bugliesi, Stefano Calzavara, Fabienne Eigner, and Matteo Maffei. 2011. Resource-Aware Authorization Policies for Statically Typed Cryptographic Protocols. In *Proc. 24th IEEE Symposium on Computer Security Foundations (CSF)*. IEEE Computer Society Press, 83–98.

Michele Bugliesi, Stefano Calzavara, Fabienne Eigner, and Matteo Maffei. 2012. Affine Refinement Types for Authentication and Authorization. In *Proc. 7th International Symposium on Trustworthy Global Computing (TGC)*. Springer-Verlag, 19–33.

Michele Bugliesi, Stefano Calzavara, Fabienne Eigner, and Matteo Maffei. 2013. Logical Foundations of Secure Resource Management in Protocol Implementations. In *Proc. 2nd Conference on Principles of Security and Trust (POST)*. Springer-Verlag, 105–125.

Michele Bugliesi, Riccardo Focardi, and Matteo Maffei. 2004a. Authenticity by Tagging and Typing. In *Proc. 2nd ACM Workshop on Formal Methods in Security Engineering (FMSE)*. ACM Press, 1–12.

Michele Bugliesi, Riccardo Focardi, and Matteo Maffei. 2004b. Compositional Analysis of Authentication Protocols. In *Proc. 13th European Symposium on Programming (ESOP) (Lecture Notes in Computer Science)*, Vol. 2986. Springer-Verlag, 140–154.

Michele Bugliesi, Riccardo Focardi, and Matteo Maffei. 2005. Analysis of Typed Analyses of Authentication Protocols. In *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW)*. IEEE Computer Society Press, 112–125.

Michele Bugliesi, Riccardo Focardi, and Matteo Maffei. 2007. Dynamic Types for Authentication. *Journal of Computer Security* 15, 6 (2007), 563–617.

Peter C. Chapin, Christian Skalka, and Xiaoyang Sean Wang. 2008. Authorization in Trust Management: Features and Foundations. *Comput. Surveys* 40, 3, Article 9 (2008), 48 pages.

Cas J. F. Cremers. 2008. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In *Proc. Computer Aided Verification'08 (CAV) (Lecture Notes in Computer Science)*. Springer-Verlag, 414–418.

Nicholaas G. de Bruijn. 1972. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)* 75, 5 (1972), 381 – 392.

Manuel Fähndrich and Robert DeLine. 2002. Adoption and focus: practical linear types for imperative programming. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. ACM Press, 13–24.

Riccardo Focardi and Matteo Maffei. 2011. Types for Security Protocols. In *Formal Models and Techniques for Analyzing Security Protocols*. Cryptology and Information Security, Vol. 5. IOS Press, Chapter 7, 143–181.

Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. 2005. A Type Discipline for Authorization Policies. In *Proc. 14th European Symposium on Programming (ESOP) (Lecture Notes in Computer Science)*. Springer-Verlag, 141–156.

Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. 2007. A Type Discipline for Authorization in Distributed Systems. In *Proc. 20th IEEE Symposium on Computer Security Foundations (CSF)*. IEEE Computer Society Press, 31–45.

Cédric Fournet, Markulf Kohlweiss, and Pierre-Yves Strub. 2011. Modular Code-Based Cryptographic Verification. In *Proc. 18th ACM Conference on Computer and Communications Security (CCS)*. ACM Press, 341–350.

Jean-Yves Girard. 1995. Linear Logic: Its Syntax and Semantics. In *Advances in Linear Logic (London Mathematical Society Lecture Note Series)*, Vol. 22. Cambridge University Press, 1–42.

Andrew D. Gordon and Alan Jeffrey. 2003. Authenticity by Typing for Security Protocols. *Journal of Computer Security* 11, 4 (2003), 451–519.

Andrew D. Gordon and Alan Jeffrey. 2004. Types and Effects for Asymmetric Cryptographic Protocols. *Journal of Computer Security* 12, 3 (2004), 435–484.

Joshua D. Guttman, F. Javier Thayer, Jay A. Carlson, Jonathan C. Herzog, John D. Ramsdell, and Brian T. Sniffen. 2004. Trust Management in Strand Spaces: A Rely-Guarantee Method. In *Proc. 13th European Symposium on Programming (ESOP) (Lecture Notes in Computer Science)*. Springer-Verlag, 325–339.

Matteo Maffei. 2004. Tags for Multi-Protocol Authentication. In *Proc. 2nd International Workshop on Security Issues in Coordination Models, Languages, and Systems (SECCO '04) (Electronic Notes on Theoretical Computer Science)*. Elsevier Science Publishers Ltd., 55–63.

Matteo Maffei. 2005. Tags for Multi-Protocol Authentication. *Electron. Notes Theor. Comput. Sci.* 128, 5 (2005), 55–63.

Yitzhak Mandelbaum, David Walker, and Robert Harper. 2003. An effective theory of type refinements. In *Proc. 8th ACM SIGPLAN International Conference on Functional Programming (ICFP)*. ACM Press, 213–225.

Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Proc. Computer Aided Verification'13 (CAV) (Lecture Notes in Computer Science)*. Springer-Verlag, 696–701.

Robin Milner. 1992. Functions as Processes. *Mathematical Structures in Computer Science* 2, 2 (1992), 119–141.

James H. Morris. 1973. Protection in Programming Languages. *Commun. ACM* 16, 1 (1973), 15–21.

Karl Naden, Robert Bocchino, Jonathan Aldrich, and Kevin Bierhoff. 2012. A Type System for Borrowing Permissions. In *Proc. 39th Symposium on Principles of Programming Languages (POPL)*. ACM Press, 557–570.

Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. 1988. Kerberos: An Authentication Service for Open Network Systems. In *Proc. USENIX Summer Conference*. USENIX Association, 191–202.

Eijiro Sumii and Benjamin C. Pierce. 2007. A Bisimulation for Dynamic Sealing. *Theoretical Computer Science* 375, 1-3 (2007), 169–192.

Joshua Sunshine, Karl Naden, Sven Stork, Jonathan Aldrich, and Éric Tanter. 2011. First-Class State Change in Plaid. In *Proc. 26th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications (OOPSLA)*. ACM Press, 713–732.

Nikhil Swamy, Juan Chen, Cédric Fournet, Pierre-Yves Strub, Karthikeyan Bhargavan, and Jean Yang. 2011. Secure Distributed Programming with Value-Dependent Types. In *Proc. 16th ACM SIGPLAN International Conference on Functional Programming (ICFP)*. ACM Press, 266–278.

Alwen Tiu and Alberto Momigliano. 2012. Cut elimination for a logic with induction and co-induction. *J. Applied Logic* 10, 4 (2012), 330–367.

Naoyuki Tomura. 1995. llprover - A Linear Logic Prover. (1995). http://bach.istc.kobe-u.ac.jp/llprover/.

Jesse Tov and Riccardo Pucella. 2010. Stateful Contracts for Affine Types. In *Proc. 19th European Symposium on Programming (ESOP)*. Springer-Verlag, 550–569.

Anne S. Troelstra. 1992. Lectures on Linear Logic. CSLI Stanford, LNS, vol. 29. (1992).

# APPENDIX

## A. SOUNDNESS OF EXPONENTIAL SERIALIZATION

We detail a soundness proof for the exponential serialization technique. The main idea behind the proof is to extend the notion of *rank* from formulas to multiset of assumptions: the rank of the multiset is computed by taking into account the serializers occurring therein. On the basis of the rank, we can always identify a serializer whose presence in the multiset can be shown to not affect the logical entailment of payload formulas, thus it can be safely removed from the assumptions. Iterating this process for each serializer, we establish that none of them affects derivability. In the proof we heavily rely on the stratification hypothesis.

In Appendix A.1 we introduce definitions and notations, which are needed for the technical development. In Appendix A.2 we present several auxiliary lemmas, which are used in the proof of the main results. In Appendix A.3 we carry out the proof of Theorem 4.4, establishing the soundness of exponential serialization. Finally, in Appendix A.4 we prove Proposition 4.5, which states a syntactic criterion for checking if a multiset of formulas is controlled.

### A.1. Preliminaries

We first introduce some notational conventions. We let:

$$\hat{S} \in \{!\forall\widetilde{x}.(P \multimap !(C \multimap P)), \forall\widetilde{x}.(P \multimap !(C \multimap P))\}$$

for some (possibly empty) $\widetilde{x}$ and some payload formula $P$ and some control formula $C$. We also write $\Delta, F^n$ as a short for the multiset $\Delta, F, \ldots, F$ (with $n$ occurrences of $F$).

We say that a multiset of formulas is *well-formed* when it is stratified and it satisfies further simple syntactic conditions, consistent with the productions given in Section 4.

*Definition* A.1 (*Well-formation*). A multiset of formulas $\Delta = \Delta_1, \Delta_2, \Delta_3$ is *well-formed* if and only if it is stratified and $\Delta_1 = P_1, \ldots, P_l$, $\Delta_2 = G_1, \ldots, G_m$, $\Delta_3 = \hat{S}_1, \ldots, \hat{S}_n$.

We define a partial function $guard$ from formulas to control formulas, defined in the following cases:

— $guard(C \multimap P) = C$;
— $guard(P \multimap G) = guard(G)$;
— $guard(\forall x.F) = guard(F)$ whenever $guard(F)$ is defined;
— $guard(!F) = guard(F)$ whenever $guard(F)$ is defined.

We extend the notion of *rank* to a multiset of formulas $\Delta$ as follows:

$$\mathit{rk}(\Delta) = \mathit{min}\ \{\mathit{rk}(C) \mid \exists F \in \Delta : guard(F) = C\}$$

If the previous set is empty, we stipulate $\mathit{rk}(\Delta) = +\infty$.

A control formula $C$ is *active* in $\Delta$ if and only if $\mathit{rk}(C) \leq \mathit{rk}(\Delta)$; we simply say that $C$ is active whenever $\Delta$ is clear from the context. The previous notion is useful to relax the definition of controlled multiset to a weaker variant.

*Definition* A.2 (*Weak Control*). A well-formed multiset $\Delta$ is *weakly controlled* if and only if, for every active control formula $C$, we have that $\Delta \vdash C^k$ implies $k \leq 1$.

We note as expected that any controlled multiset is also weakly controlled.

PROPOSITION A.3. *If $\Delta$ is controlled, then it is weakly controlled.*

In the next results we focus without loss of generality on cut-free proofs.

### A.2. Auxiliary results

The first two lemmas are needed to show that the induction hypothesis can indeed be applied in the proof of a number of subsequent results.

LEMMA A.4. *Let $\Delta$ be well-formed. The following implications hold:*

(1) *if $\Delta = \Delta', F$, then $\Delta'$ is well-formed and rk$(\Delta) \leq rk(\Delta')$;*
(2) *if $\Delta = \Delta', !F$, then $\Delta, !F$ is well-formed and rk$(\Delta) =$ rk$(\Delta, !F)$;*
(3) *if $\Delta = \Delta', F_1 \otimes F_2$, then $\Delta', F_1, F_2$ is well-formed and rk$(\Delta) =$ rk$(\Delta', F_1, F_2)$;*
(4) *if $\Delta = \Delta_1, \Delta_2, F_1 \multimap F_2$, then $\Delta_2, F_2$ is well-formed and rk$(\Delta) \leq$ rk$(\Delta_2, F_2)$;*
(5) *if $\Delta = \Delta', \forall x.F$, then for every $t$ we have that $\Delta', F\{t/x\}$ is well-formed and rk$(\Delta) =$ rk$(\Delta', F\{t/x\})$;*
(6) *if $\Delta = \Delta', !F$, then $\Delta', F$ is well-formed and rk$(\Delta) =$ rk$(\Delta', F)$.*

PROOF. By some simple syntactic checks. □

LEMMA A.5. *Let $\Delta$ be weakly controlled. The following implications hold:*

(1) *if $\Delta = \Delta', F$, then $\Delta'$ is weakly controlled;*
(2) *if $\Delta = \Delta', !F$, then $\Delta, !F$ is weakly controlled;*
(3) *if $\Delta = \Delta', F_1 \otimes F_2$, then $\Delta', F_1, F_2$ is weakly controlled;*
(4) *if $\Delta = \Delta_1, \Delta_2, F_1 \multimap F_2$ and $\Delta_1 \vdash F_1$, then $\Delta_2, F_2$ is weakly controlled;*
(5) *if $\Delta = \Delta', \forall x.F$, then $\Delta', F\{t/x\}$ is weakly controlled for every $t$;*
(6) *if $\Delta = \Delta', !F$, then $\Delta', F$ is weakly controlled.*

PROOF. For all the points of the statement, let $\Delta_c$ denote the multiset in the conclusion. Lemma A.4 guarantees that $\Delta_c$ is well-formed. Now we observe that in each case, for every formula $F$, we have that $\Delta_c \vdash F$ implies $\Delta \vdash F$ by the application of a specific rule of the logic. Thus, let us assume by contradiction that $\Delta_c \vdash C^n$ with $n > 1$ for some active control formula $C$. By the previous observation, we have $\Delta \vdash C^n$, but this is contradictory, since $\Delta$ is weakly controlled by hypothesis. □

The next lemma formalizes the intuition behind stratification: formulas with a given rank are never needed in the proof of a control formula with a lower rank. This observation plays a prominent role in many of the later results.

LEMMA A.6 (STRATIFICATION). *Let $\Delta = \Delta', F_1, \ldots, F_m$ be well-formed and let $C$ be active in $\Delta$. If $\Delta \vdash C^n$ with $n \geq 1$ and $\forall i \in [1, m]$ : rk$(F_i) >$ rk$(C)$, then $\Delta' \vdash C^n$.*

PROOF. By induction on the derivation of $\Delta \vdash C^n$:

*Case* (IDENT): the case is trivial, since the hypothesis on the rank cannot hold;
*Case* (WEAK): let us assume that the principal formula is $F_1$, so we have $\Delta \vdash C^n$ by the premise $\Delta', F_2, \ldots, F_m \vdash C^n$. The latter multiset is well-formed and $C$ is active there by Lemma A.4, so we can apply the inductive hypothesis to get $\Delta' \vdash C^n$. Otherwise, let the principal formula belong to $\Delta'$, the conclusion follows by inductive hypothesis and (WEAK);
*Case* (CONTR): let us assume that the principal formula is $F_1$, so we have $\Delta \vdash C^n$ by the premise $\Delta, F_1 \vdash C^n$. The latter multiset is well-formed and $C$ is active there by Lemma A.4, so we can apply the inductive hypothesis to get $\Delta' \vdash C^n$. Otherwise, let the principal formula belong to $\Delta'$, the conclusion follows by inductive hypothesis and (CONTR);
*Case* ($\otimes$-LEFT): let us assume that the principal formula is $F_1 = F' \otimes F''$, so we have $\Delta \vdash C^n$ by the premises $\Delta', F', F'', F_2, \ldots, F_m \vdash C^n$. The latter multiset is well-formed and $C$ is active there by Lemma A.4 . Since rk$(F' \otimes F'') = min$ $\{$rk$(F')$, rk$(F'')\}$ and rk$(F' \otimes F'') >$ rk$(C)$, we know that both rk$(F') >$ rk$(C)$ and rk$(F'') >$ rk$(C)$. By

inductive hypothesis we then get $\Delta' \vdash C^n$ as desired. Otherwise, let the principal formula belong to $\Delta'$, the conclusion follows by inductive hypothesis and ($\otimes$-LEFT);

*Case* ($\otimes$-RIGHT): let $\Delta \vdash C^n$ with $n \geq 2$ by the premises $\Delta_1 \vdash C^j$ and $\Delta_2 \vdash C^k$ with $\Delta = \Delta_1, \Delta_2$ and $j+k = n$. Let us assume without loss of generality that $\Delta_1 = \Delta'_1, F_1, \ldots, F_h$ and $\Delta_2 = \Delta'_2, F_{h+1}, \ldots, F_m$. Both $\Delta_1$ and $\Delta_2$ are well-formed and $C$ is active there by Lemma A.4. By inductive hypothesis we get $\Delta'_1 \vdash C^j$ and $\Delta'_2 \vdash C^k$, so we conclude $\Delta' \vdash C^n$ by an application of ($\otimes$-RIGHT);

*Case* ($\multimap$-LEFT): let us assume that the principal formula is $F_1 = F' \multimap F''$. Since $\Delta$ is well-formed, we can distinguish three cases:

— $F' = B_1, F'' = B_2$. We have $\Delta_1 \vdash B_1$ and $\Delta_2, B_2 \vdash C^n$ with $\Delta', F_2, \ldots, F_m = \Delta_1, \Delta_2$. Let us assume without loss of generality that $\Delta_1 = \Delta'_1, F_2, \ldots, F_h$ and $\Delta_2 = \Delta'_2, F_{h+1}, \ldots, F_m$. We know that $\Delta_2, B_2$ is well-formed and $C$ is active there by Lemma A.4. Moreover, we note that $\mathit{rk}(B_2) = +\infty$, while $\mathit{rk}(C)$ is finite, so we can apply the inductive hypothesis to get $\Delta'_2 \vdash C^n$. The conclusion follows by applying (WEAK) an appropriate number of times;

— $F' = \hat{C}, F'' = \hat{P}$. We have $\Delta_1 \vdash \hat{C}$ and $\Delta_2, \hat{P} \vdash C^n$ with $\Delta', F_2, \ldots, F_m = \Delta_1, \Delta_2$. Let us assume without loss of generality that $\Delta_1 = \Delta'_1, F_2, \ldots, F_h$ and $\Delta_2 = \Delta'_2, F_{h+1}, \ldots, F_m$. We know that $\Delta_2, \hat{P}$ is well-formed and $C$ is active there by Lemma A.4. Moreover, we note that $\mathit{rk}(C) \leq \mathit{rk}(\hat{C}) < \mathit{rk}(\hat{P})$ by the hypothesis of stratification, so we can apply the inductive hypothesis to get $\Delta'_2 \vdash C^n$. The conclusion follows by applying (WEAK) an appropriate number of times;

— $F' = \hat{P}, F'' = !(\hat{C} \multimap \hat{P})$. We have $\Delta_1 \vdash \hat{P}$ and $\Delta_2, !(\hat{C} \multimap \hat{P}) \vdash C^n$ with $\Delta', F_2, \ldots, F_m = \Delta_1, \Delta_2$. Let us assume without loss of generality that $\Delta_1 = \Delta'_1, F_2, \ldots, F_h$ and $\Delta_2 = \Delta'_2, F_{h+1}, \ldots, F_m$. We know that $\Delta_2, !(\hat{C} \multimap \hat{P})$ is well-formed and $C$ is active there by Lemma A.4. Moreover, we note that $\mathit{rk}(!(\hat{C} \multimap \hat{P})) = +\infty$, while $\mathit{rk}(C)$ is finite, so we can apply the inductive hypothesis to get $\Delta'_2 \vdash C^n$. The conclusion follows by applying (WEAK) an appropriate number of times.

Otherwise, let the principal formula be $F = F' \multimap F''$ with $\Delta' = \Delta'', F$. Without loss of generality, we can assume $\mathit{rk}(F) \leq \mathit{rk}(C)$, but this is contradictory, since any implication has an infinite rank, which is strictly greater than the finite rank of $C$.

*Case* ($\forall$-LEFT): let us assume that the principal formula is $F_1 = \forall x.F'$, so we have $\Delta \vdash C^n$ by the premises $\Delta', F'\{t/x\}, F_2, \ldots, F_m \vdash C^n$ for some $t$. The latter multiset is well-formed and $C$ is active there by Lemma A.4. Note that $\mathit{rk}(\forall x.F') = \mathit{rk}(F'\{t/x\}) = +\infty$, since the multiset is well-formed, hence by inductive hypothesis we get $\Delta' \vdash C^n$ as desired. Otherwise, let the principal formula belong to $\Delta'$, the conclusion follows by inductive hypothesis and ($\forall$-LEFT);

*Case* (!-LEFT): let us assume that the principal formula is $F_1 = !F'$, so we have $\Delta \vdash C^n$ by the premises $\Delta', F', F_2, \ldots, F_m \vdash C^n$. The latter multiset is well-formed and $C$ is active there by Lemma A.4. Note that $\mathit{rk}(!F') = \mathit{rk}(F') = +\infty$, since the multiset is well-formed, hence by inductive hypothesis we get $\Delta' \vdash C^n$ as desired. Otherwise, let the principal formula belong to $\Delta'$, the conclusion follows by inductive hypothesis and (!-LEFT).

□

The next result is a strengthening lemma: if a multiset does not entail a given control formula $C$, then any implication of the form $C \multimap P$ occurring therein can be removed without affecting derivability.

LEMMA A.7 (STRENGTHENING). *Let $\Delta = \Delta', C \multimap P$ be well-formed and let $C$ be active in $\Delta$. If $\Delta \vdash P'$ and $\Delta' \nvdash C$, then $\Delta' \vdash P'$.*

PROOF. By induction on the derivation of $\Delta \vdash P'$. We show just the most interesting cases:

*Case* (IDENT): the rule cannot be applied, since $C \multimap P$ is not a payload formula;
*Case* ($\otimes$-LEFT): let $\Delta' = \Delta'', F_1 \otimes F_2$ and let $\Delta \vdash P'$ by $\Delta'', F_1, F_2, C \multimap P \vdash P'$. By Lemma A.4 we know that the latter multiset is well-formed and $C$ is active. Moreover, we note that $\Delta'', F_1, F_2 \nvdash C$, otherwise we could get $\Delta' \vdash C$ by an application of ($\otimes$-LEFT). Thus, we can apply the inductive hypothesis to get $\Delta'', F_1, F_2 \vdash P'$ and conclude by ($\otimes$-LEFT);
*Case* ($\otimes$-RIGHT): let $\Delta \vdash P_1 \otimes P_2$ by the hypotheses $\Delta_1 \vdash P_1$ and $\Delta_2 \vdash P_2$ with $\Delta = \Delta_1, \Delta_2$. By Lemma A.4 both $\Delta_1$ and $\Delta_2$ are well-formed and $C$ is active in both. Without loss of generality, let us assume $\Delta_1 = \Delta_1', C \multimap P$. We note that $\Delta_1' \nvdash C$, otherwise we could get $\Delta' \vdash C$ by (WEAK). Thus, we can apply the inductive hypothesis to get $\Delta_1' \vdash P_1$ and conclude by ($\otimes$-RIGHT);
*Case* ($\multimap$-LEFT): we distinguish two cases, according to the principal formula:

— let $\Delta' = \Delta'', F_1 \multimap F_2$ and let $\Delta \vdash P'$ by the hypotheses $\Delta_1 \vdash F_1$ and $\Delta_2, F_2 \vdash P'$ with $\Delta'', C \multimap P = \Delta_1, \Delta_2$. By Lemma A.4 both $\Delta_1$ and $\Delta_2, F_2$ are well-formed and $C$ is active in both. We distinguish two cases, according to whether $C \multimap P$ belongs to $\Delta_1$ or to $\Delta_2$.
  Let $\Delta_1 = \Delta_1', C \multimap P$, we have that $F_1$ is a payload formula. Moreover, since $\Delta' \nvdash C$ and $\Delta_1' \subseteq \Delta'$, we know that $\Delta_1' \nvdash C$, otherwise we could deduce $\Delta' \vdash C$ by (WEAK). Thus, we can apply the inductive hypothesis to get $\Delta_1' \vdash F_1$ and conclude by ($\multimap$-LEFT).
  Otherwise, let $\Delta_2 = \Delta_2', C \multimap P$. We know that $\Delta_2', F_2 \nvdash C$, otherwise we could deduce $\Delta' \vdash C$ by ($\multimap$-LEFT). Thus, we can apply the inductive hypothesis to get $\Delta_2', F_2 \vdash P'$ and conclude by ($\multimap$-LEFT);
— let $\Delta' \vdash P'$ by the hypotheses $\Delta_1 \vdash C$ and $\Delta_2, P \vdash P'$ with $\Delta' = \Delta_1, \Delta_2$. We have a contradiction, since $\Delta_1 \vdash C$ implies $\Delta' \vdash C$ by (WEAK).

*Case* ($\multimap$-RIGHT): the only possibility is that $\Delta \vdash B_1 \multimap B_2$ by the hypothesis $\Delta, B_1 \vdash B_2$. It is immediate to note that $\Delta, B_1$ is well-formed and that $C$ is active, since the introduction of $B_1$ cannot change the rank of the multiset. Let us assume by contradiction that $\Delta', B_1 \vdash C$. Since $\mathit{rk}(C)$ is finite, while $\mathit{rk}(B_1) = +\infty$, we have that $\Delta' \vdash C$ by Lemma A.6 (Stratification), but this is contradictory. Thus, we have $\Delta', B_1 \nvdash C$, so we can apply the inductive hypothesis to get $\Delta', B_1 \vdash B_2$ and conclude by ($\multimap$-RIGHT);
*Case* (!-LEFT): let $\Delta' = \Delta'', !F$ and let $\Delta \vdash P'$ by the hypothesis $\Delta'', F, C \multimap P \vdash P'$. By Lemma A.4 we know that the latter multiset is well-formed and $C$ is active. Moreover, we note that $\Delta'', F \nvdash C$, otherwise we could get $\Delta' \vdash C$ by (!-LEFT). Thus, we can apply the inductive hypothesis to get $\Delta'', F \vdash P'$ and conclude by (!-LEFT);
*Case* (!-RIGHT): the rule cannot be applied, since $\Delta', C \multimap P$ is not exponential.

□

The next technical lemma allows to apply the induction hypothesis in the proof of the subsequent results.

LEMMA A.8. *Let $\Delta$ be weakly controlled, then $\Delta, B$ is weakly controlled. Moreover, we have $\mathit{rk}(\Delta) = \mathit{rk}(\Delta, B)$.*

PROOF. It is immediate to note that $\Delta, B$ is well-formed and that the introduction of $B$ does not change the rank of the multiset. As to weak control, let us assume by contradiction that $\Delta, B \vdash C^n$ with $n > 1$ for some active control formula $C$. Since $rk(C)$ is finite, while $rk(B) = +\infty$, we have that $\Delta \vdash C^n$ by Lemma A.6 (Stratification). But this is contradictory, since $\Delta$ is weakly controlled. $\square$

The next lemma formalizes an important intuition: since any active control formula $C$ can be proved at most once in a weakly controlled multiset, all the implications of the form $C \multimap P$ occurring therein can be replaced by a single implication of the same form without affecting derivability. This is needed in the proof of Lemma A.11 (Dereliction).

LEMMA A.9 (BOUNDED REACTION). *Let $\Delta = \Delta', (C \multimap P)^n$ be weakly controlled. If $\Delta \vdash P'$ and $C$ is active in $\Delta$, then $\Delta', C \multimap P \vdash P'$.*

PROOF. By induction on the derivation of $\Delta \vdash P'$. Without loss of generality in the following inductive cases we assume $n \geq 2$, since the conclusion follows by (WEAK) for $n = 0$ and it is trivial for $n = 1$. We show just the most interesting cases:

*Case* (IDENT): the rule cannot be applied, since we are assuming to have at least two copies of $C \multimap P$ in the multiset. Moreover, $C \multimap P$ is not a payload formula;

*Case* ($\otimes$-LEFT): let $\Delta' = \Delta'', F_1 \otimes F_2$ and let $\Delta \vdash P'$ from $\Delta'', F_1, F_2, (C \multimap P)^n \vdash P'$. The latter multiset is weakly controlled by Lemma A.5 and $C$ is active by Lemma A.4, so we can apply the inductive hypothesis to get $\Delta'', F_1, F_2, C \multimap P \vdash P'$ and conclude $\Delta', C \multimap P \vdash P'$ by ($\otimes$-LEFT);

*Case* ($\otimes$-RIGHT): let $\Delta \vdash P_1 \otimes P_2$ by the premises $\Delta_1 \vdash P_1$ and $\Delta_2 \vdash P_2$ with $\Delta = \Delta_1, \Delta_2$. Let $\Delta_1 = \Delta_1', (C \multimap P)^h$ and $\Delta_2 = \Delta_2', (C \multimap P)^k$ with $h + k = n$. Both $\Delta_1$ and $\Delta_2$ are weakly controlled by Lemma A.5 and $C$ is active in both by Lemma A.4. We apply the inductive hypothesis to both $\Delta_1 \vdash P_1$ and $\Delta_2 \vdash P_2$ to get $\Delta_1', C \multimap P \vdash P_1$ and $\Delta_2', C \multimap P \vdash P_2$. Now we assume by contradiction that both $\Delta_1' \vdash C$ and $\Delta_2' \vdash C$, then we have $\Delta \vdash C \otimes C$ by ($\otimes$-RIGHT) and an appropriate number of applications of (WEAK), but, given that $\Delta$ is weakly controlled, this is contradictory. Without loss of generality we can then assume that $\Delta_1' \nvdash C$, so by Lemma A.7 (Strengthening) we have $\Delta_1' \vdash P_1$ and we conclude $\Delta_1', \Delta_2', C \multimap P \vdash P_1 \otimes P_2$ by ($\otimes$-RIGHT);

*Case* ($\multimap$-LEFT): we distinguish four cases, according to the principal formula:

— let $\Delta \vdash P'$ by the premises $\Delta_1 \vdash C$ and $\Delta_2, P \vdash P'$ with $\Delta', (C \multimap P)^{n-1} = \Delta_1, \Delta_2$. Let $\Delta_1 = \Delta_1', (C \multimap P)^h$ and $\Delta_2 = \Delta_2', (C \multimap P)^k$ with $h + k = n - 1$. By Lemma A.5 we have that both $\Delta_1$ and $\Delta_2, P$ are weakly controlled and $C$ is active in both by Lemma A.4. Moreover, $C$ is a control formula, i.e., it is a payload formula. We are thus allowed to apply the inductive hypothesis to both $\Delta_1 \vdash C$ and $\Delta_2, P \vdash P'$ to get $\Delta_1', C \multimap P \vdash C$ and $\Delta_2', P, C \multimap P \vdash P'$. Since $rk(C \multimap P) = +\infty > rk(C)$, by Lemma A.6 (Stratification) we have $\Delta_1' \vdash C$. Let us assume by contradiction that $\Delta_2', P \vdash C$: since $rk(C) < rk(P)$ by the stratification hypothesis, by Lemma A.6 (Stratification) we know that $\Delta_2' \vdash C$, but this is contradictory, since we would first get $\Delta_1', \Delta_2' \vdash C \otimes C$ by ($\otimes$-RIGHT) and then $\Delta \vdash C \otimes C$ by (WEAK). Thus, $\Delta_2', P \nvdash C$ and we have $\Delta_2', P \vdash P'$ by Lemma A.7 (Strengthening). We can then conclude $\Delta_1', \Delta_2', C \multimap P \vdash P'$ by an application of ($\multimap$-LEFT);

— let $\Delta' = \Delta'', B_1 \multimap B_2$ by the premises $\Delta_1 \vdash B_1$ and $\Delta_2, B_2 \vdash P'$ with $\Delta'', (C \multimap P)^n = \Delta_1, \Delta_2$. Let $\Delta_1 = \Delta_1', (C \multimap P)^h$ and $\Delta_2 = \Delta_2', (C \multimap P)^k$ with $h + k = n$. By Lemma A.5 we have that both $\Delta_1$ and $\Delta_2, B_2$ are weakly controlled and $C$ is active in both by Lemma A.4. Moreover, $B_1$ is a base formula, i.e., it is a payload formula. We are thus allowed to apply the inductive hypothesis to both $\Delta_1 \vdash B_1$ and $\Delta_2, B_2 \vdash P'$ to get $\Delta_1', C \multimap P \vdash B_1$ and $\Delta_2', B_2, C \multimap P \vdash P'$. Let us assume by contradiction that both $\Delta_1' \vdash C$ and $\Delta_2', B_2 \vdash C$: since $rk(B_2) = +\infty > rk(C)$, by Lemma A.6 (Stratification) we have $\Delta_2' \vdash C$, whence $\Delta \vdash C \otimes C$ by ($\otimes$-RIGHT) and (WEAK), which is contradictory. Thus, we can apply Lemma A.7 (Strengthening) either on $\Delta_1', C \multimap P \vdash B_1$ or on $\Delta_2', B_2, C \multimap P \vdash P'$ to remove $C \multimap P$ from the judgement. The conclusion follows by ($\multimap$-LEFT);

— let $\Delta' = \Delta'', \hat{C} \multimap \hat{P}$ by the premises $\Delta_1 \vdash \hat{C}$ and $\Delta_2, \hat{P} \vdash P'$ with $\Delta'', (C \multimap P)^n = \Delta_1, \Delta_2$. Let $\Delta_1 = \Delta_1', (C \multimap P)^h$ and $\Delta_2 = \Delta_2', (C \multimap P)^k$ with $h + k = n$. By Lemma A.5 we have that both $\Delta_1$ and $\Delta_2, \hat{P}$ are weakly controlled and $C$ is active in both by Lemma A.4. Moreover, $\hat{C}$ is a control formula, i.e., it is a payload formula. We are thus allowed to apply the inductive hypothesis to both $\Delta_1 \vdash \hat{C}$ and $\Delta_2, \hat{P} \vdash P'$ to get $\Delta_1', C \multimap P \vdash \hat{C}$ and $\Delta_2', \hat{P}, C \multimap P \vdash P'$. Let us assume by contradiction that both $\Delta_1' \vdash C$ and $\Delta_2', \hat{P} \vdash C$: since $\mathit{rk}(C) \leq \mathit{rk}(\hat{C}) < \mathit{rk}(\hat{P})$ by the stratification hypothesis, by Lemma A.6 (Stratification) we have $\Delta_2' \vdash C$, whence $\Delta \vdash C \otimes C$ by ($\otimes$-RIGHT) and (WEAK), which is contradictory. Thus, we can apply Lemma A.7 (Strengthening) either on $\Delta_1', C \multimap P \vdash \hat{C}$ or on $\Delta_2', \hat{P}, C \multimap P \vdash P'$ to remove $C \multimap P$ from the judgement. The conclusion follows by ($\multimap$-LEFT);

— let $\Delta' = \Delta'', \hat{P} \multimap !(\hat{C} \multimap \hat{P})$ by the premises $\Delta_1 \vdash \hat{P}$ and $\Delta_2, !(\hat{C} \multimap \hat{P}) \vdash P'$ with $\Delta'', (C \multimap P)^n = \Delta_1, \Delta_2$. Let $\Delta_1 = \Delta_1', (C \multimap P)^h$ and $\Delta_2 = \Delta_2', (C \multimap P)^k$ with $h + k = n$. By Lemma A.5 we have that both $\Delta_1$ and $\Delta_2, !(\hat{C} \multimap \hat{P})$ are weakly controlled and $C$ is active in both by Lemma A.4. Moreover, $\hat{P}$ is a payload formula. We are thus allowed to apply the inductive hypothesis to both $\Delta_1 \vdash \hat{P}$ and $\Delta_2, !(\hat{C} \multimap \hat{P}) \vdash P'$ to get $\Delta_1', C \multimap P \vdash \hat{P}$ and $\Delta_2', !(\hat{C} \multimap \hat{P}), C \multimap P \vdash P'$. Let us assume by contradiction that both $\Delta_1' \vdash C$ and $\Delta_2', !(\hat{C} \multimap \hat{P}) \vdash C$: since $\mathit{rk}(!(\hat{C} \multimap \hat{P})) = +\infty > \mathit{rk}(C)$, by Lemma A.6 (Stratification) we have $\Delta_2' \vdash C$, whence $\Delta \vdash C \otimes C$ by ($\otimes$-RIGHT) and (WEAK), which is contradictory. Thus, we can apply Lemma A.7 (Strengthening) either on $\Delta_1', C \multimap P \vdash \hat{P}$ or on $\Delta_2', !(\hat{C} \multimap \hat{P}), C \multimap P \vdash P'$ to remove $C \multimap P$ from the judgement. The conclusion follows by ($\multimap$-LEFT);

*Case* ($\multimap$-RIGHT): let $\Delta \vdash B_1 \multimap B_2$ by the hypothesis $\Delta, B_1 \vdash B_2$. By Lemma A.8 we know that $\Delta, B_1$ is weakly controlled and $C$ is active, so by inductive hypothesis we get $\Delta', B_1, C \multimap P \vdash B_2$ and we conclude by ($\multimap$-RIGHT);

*Case* (!-LEFT): let $\Delta' = \Delta'', !F$ and let $\Delta \vdash P'$ by the hypothesis $\Delta'', F, (C \multimap P)^n \vdash P'$. The latter multiset is weakly controlled by Lemma A.5 and $C$ is active by Lemma A.4, so we can apply the inductive hypothesis to get $\Delta'', F, C \multimap P \vdash P'$ and we conclude by (!-LEFT);

*Case* (!-RIGHT): the rule cannot be applied, since we are assuming to have at least two copies of $C \multimap P$ in the multiset, i.e., the multiset is not exponential.

□

The next technical corollary is used in the proof of Lemma A.11 (Dereliction) below.

COROLLARY A.10. *Let $\Delta$ be well-formed. If $\Delta \vdash C$ and $C$ is active in $\Delta$, then $\Delta$ contains at least an affine formula.*

PROOF. Let $\Delta'$ be the multiset obtained from $\Delta$ by removing all the formulas $F$ such that $\mathit{rk}(F) > \mathit{rk}(C)$. By Lemma A.6 (Stratification) we have $\Delta' \vdash C$. Clearly, $\Delta'$ must contain at least a formula $\hat{F}$, since $\emptyset \nvdash C$, and by construction we know that $\mathit{rk}(\hat{F}) \leq \mathit{rk}(C)$. Since $\mathit{rk}(C)$ is finite, also $\mathit{rk}(\hat{F})$ is finite, so this formula must be affine by Definition 4.1. □

The next lemma is reminiscent of the idea behind Lemma A.9 (Bounded Reaction): since any active control formula $C$ can be proved at most once in a weakly controlled multiset, an exponential implication of the form $!(C \multimap P)$ occurring therein can be replaced by an affine implication $C \multimap P$ without affecting derivability.

LEMMA A.11 (DERELICTION). *Let $\Delta = \Delta', !(C \multimap P)$ be weakly controlled. If $\Delta \vdash P'$ and $C$ is active in $\Delta$, then $\Delta', C \multimap P \vdash P'$.*

PROOF. We prove a stronger statement, namely:

$$\forall n \geq 0 : \Delta', (!(C \multimap P))^n \vdash P' \text{ implies } \Delta', C \multimap P \vdash P',$$

provided that the initial hypotheses are satisfied. We proceed by induction on the derivation of the judgement in the premise, we show just the most interesting cases:

*Case* (IDENT): if $n > 0$, this rule cannot be applied, since $!(C \multimap P)$ is not a payload formula. If $n = 0$, we have $P' \vdash P'$ by hypothesis and we conclude $P', C \multimap P \vdash P'$ by (WEAK);

*Case* (WEAK): we have two cases. If the principal formula is $!(C \multimap P)$, the conclusion is immediate by inductive hypothesis. Otherwise, if the principal formula belongs to $\Delta'$, the conclusion follows by applying the inductive hypothesis and (WEAK);

*Case* (CONTR): we have two cases. If the principal formula is $!(C \multimap P)$, the conclusion is immediate by inductive hypothesis. Otherwise, if the principal formula belongs to $\Delta'$, the conclusion follows by applying the inductive hypothesis and (CONTR);

*Case* ($\otimes$-LEFT): $\Delta' = \Delta'', F_1 \otimes F_2$ and let $\Delta \vdash P'$ from $\Delta'', F_1, F_2, (!(C \multimap P))^n \vdash P'$. The latter multiset is weakly controlled by Lemma A.5 and $C$ is active by Lemma A.4, so we can apply the inductive hypothesis to get $\Delta'', F_1, F_2, C \multimap P \vdash P'$ and conclude $\Delta', C \multimap P \vdash P'$ by ($\otimes$-LEFT);

*Case* ($\otimes$-RIGHT): let $\Delta \vdash P_1 \otimes P_2$ by the hypotheses $\Delta_1 \vdash P_1$ and $\Delta_2 \vdash P_2$ with $\Delta = \Delta_1, \Delta_2$. Let $\Delta_1 = \Delta_1', (!(C \multimap P))^h$ and $\Delta_2 = \Delta_2', (!(C \multimap P))^k$ with $h + k = n$. We have that both $\Delta_1$ and $\Delta_2$ are weakly controlled by Lemma A.5 and $C$ is active in both by Lemma A.4. We apply the inductive hypothesis to both $\Delta_1 \vdash P_1$ and $\Delta_2 \vdash P_2$ to get $\Delta_1', C \multimap P \vdash P_1$ and $\Delta_2', C \multimap P \vdash P_2$. Thus, we have $\Delta_1', \Delta_2', C \multimap P, C \multimap P \vdash P_1 \otimes P_2$ by ($\otimes$-RIGHT) and we conclude $\Delta_1', \Delta_2', C \multimap P \vdash P_1 \otimes P_2$ by Lemma A.9 (Bounded Reaction);

*Case* ($\multimap$-LEFT): let $\Delta' = \Delta'', F_1 \multimap F_2$ and let $\Delta \vdash P'$ by the hypotheses $\Delta_1 \vdash F_1$ and $\Delta_2, F_2 \vdash P'$ with $\Delta'', (!(C \multimap P))^n = \Delta_1, \Delta_2$. Let $\Delta_1 = \Delta_1', (!(C \multimap P))^h$ and $\Delta_2 = \Delta_2', (!(C \multimap P))^k$ with $h + k = n$. By Lemma A.5 we have that both $\Delta_1$ and $\Delta_2, F_2$ are weakly controlled and $C$ is active in both by Lemma A.4. Moreover, we note that $F_1$ is a payload formula. We are thus allowed to apply the inductive hypothesis to both $\Delta_1 \vdash F_1$ and $\Delta_2, F_2 \vdash P'$ to get respectively $\Delta_1', C \multimap P \vdash F_1$ and $\Delta_2', F_2, C \multimap P \vdash P'$. Thus, we get $\Delta', C \multimap P, C \multimap P \vdash P'$ by ($\multimap$-LEFT) and we conclude $\Delta', C \multimap P \vdash P'$ by Lemma A.9 (Bounded Reaction);

*Case* ($\multimap$-RIGHT): let $\Delta \vdash B_1 \multimap B_2$ by the hypothesis $\Delta, B_1 \vdash B_2$. By Lemma A.8 we know that $\Delta, B_1$ is weakly controlled and $C$ is active, so by inductive hypothesis we get $\Delta', B_1, C \multimap P \vdash B_2$ and we conclude by ($\multimap$-RIGHT);

*Case* (!-LEFT): if the principal formula is $!(C \multimap P)$, we have $\Delta', C \multimap P, !(C \multimap P)^{n-1} \vdash P'$. If $n - 1 = 0$, we are done, otherwise we apply the inductive hypothesis to get $\Delta', C \multimap P, C \multimap P \vdash P'$ and we conclude by Lemma A.9 (Bounded Reaction). Otherwise, if the principal formula belongs to $\Delta'$, the conclusion follows by first applying the inductive hypothesis and then using (!-LEFT);

*Case* (!-RIGHT): we have $\Delta', (!(C \multimap P))^n \vdash !B$ by the hypothesis $\Delta', (!(C \multimap P))^n \vdash B$ with $\Delta'$ exponential. By inductive hypothesis $\Delta', C \multimap P \vdash B$. Let us assume by contradiction that $\Delta' \vdash C$: then, by Corollary A.10, there exists an affine formula in $\Delta'$, but this is contradictory. Thus, we have $\Delta' \nvdash C$, which implies $\Delta' \vdash B$ by Lemma A.7 (Strengthening). We can then apply (!-RIGHT) to derive $\Delta' \vdash !B$ and we conclude $\Delta', C \multimap P \vdash !B$ by (WEAK).

□

The next lemma states that an implication of the form $C \multimap P$ is useless whenever the payload formula $P$ is already available in the context. This is needed to prove Corollary A.13 (Bounded Usage), which is the real result of interest and formalizes a similar idea.

LEMMA A.12. *Let $\Delta = \Delta', C \multimap P$ be well-formed. If $\Delta \vdash P'$, then $\Delta', P \vdash P'$.*

PROOF. By induction on the derivation of $\Delta \vdash P'$. The proof strongly resembles those of the previous results, but it is actually easier. We just show the most interesting cases:

*Case* (IDENT): the rule cannot be applied, since $C \multimap P$ is not a payload formula;
*Case* (WEAK): if the principal formula is $C \multimap P$, the conclusion follows by (WEAK). Otherwise, if the principal formula belongs to $\Delta'$, the conclusion follows by first applying the inductive hypothesis and then using (WEAK);
*Case* ($\otimes$-LEFT): let $\Delta' = \Delta'', F_1 \otimes F_2$ and let $\Delta \vdash P'$ from $\Delta'', F_1, F_2, C \multimap P \vdash P'$. The latter multiset is well-formed by Lemma A.4, so we can apply the inductive hypothesis to get $\Delta'', F_1, F_2, P \vdash P'$ and conclude $\Delta', P \vdash P'$ by ($\otimes$-LEFT);
*Case* ($\otimes$-RIGHT): let $\Delta \vdash P_1 \otimes P_2$ by the hypotheses $\Delta_1 \vdash P_1$ and $\Delta_2 \vdash P_2$ with $\Delta = \Delta_1, \Delta_2$. By Lemma A.4 both $\Delta_1$ and $\Delta_2$ are well-formed, so we are allowed to apply the inductive hypothesis to either $\Delta_1 \vdash P_1$ or $\Delta_2 \vdash P_2$, according to whether $C \multimap P$ occurs in $\Delta_1$ or in $\Delta_2$, and conclude by ($\otimes$-RIGHT);
*Case* ($\multimap$-LEFT): we distinguish two cases, according to the principal formula:
- let $\Delta' = \Delta'', F_1 \multimap F_2$ and let $\Delta \vdash P'$ by the hypotheses $\Delta_1 \vdash F_1$ and $\Delta_2, F_2 \vdash P'$ with $\Delta'', C \multimap P = \Delta_1, \Delta_2$. By Lemma A.4 we have that both $\Delta_1$ and $\Delta_2, F_2$ are well-formed. Moreover, we note that $F_1$ is a payload formula. We are thus allowed to apply the inductive hypothesis to either $\Delta_1 \vdash F_1$ or $\Delta_2, F_2 \vdash P'$, according to whether $C \multimap P$ occurs in $\Delta_1$ or in $\Delta_2$, and conclude by ($\multimap$-LEFT);
- let $\Delta \vdash P'$ by the hypotheses $\Delta_1 \vdash C$ and $\Delta_2, P \vdash P'$ with $\Delta' = \Delta_1, \Delta_2$. We conclude $\Delta', P \vdash P'$ by applying (WEAK) for an appropriate number of times.

*Case* ($\multimap$-RIGHT): let $\Delta \vdash B_1 \multimap B_2$ by the hypothesis $\Delta, B_1 \vdash B_2$. It is immediate to note that $\Delta, B_1$ is well-formed, so by inductive hypothesis we get $\Delta', B_1, P \vdash B_2$ and we conclude $\Delta', P \vdash B_1 \multimap B_2$ by ($\multimap$-RIGHT);
*Case* (!-LEFT): let $\Delta' = \Delta'', !F$ and let $\Delta \vdash P'$ by the hypothesis $\Delta'', F, C \multimap P \vdash P'$. The latter multiset is well-formed by Lemma A.4, so we can apply the inductive hypothesis to get $\Delta'', F, P \vdash P'$ and we conclude $\Delta', P \vdash P'$ by (!-LEFT);
*Case* (!-RIGHT): the rule cannot be applied, since $C \multimap P$ is not exponential.

□

COROLLARY A.13 (BOUNDED USAGE). *Let $\Delta = \Delta', !(C \multimap P)$ be weakly controlled. If $\Delta \vdash P'$ and $C$ is active in $\Delta$, then $\Delta', P \vdash P'$.*

PROOF. By Lemma A.11 (Dereliction) we have $\Delta', C \multimap P \vdash P'$. By Lemma A.5 we know that $\Delta', C \multimap P$ is weakly controlled, i.e., it is well-formed, thus the conclusion follows by Lemma A.12. □

### A.3. Proof of Theorem 4.4

We are finally ready to show that serializers do not affect the derivability of payload formulas. However, since a serializer $S = !\forall\widetilde{x}.(P \multimap !(C \multimap P))$ has a non-trivial structure, in the proof of the main theorem we must take into account the possibility of decomposing $S$ through the left rules of the logic, by $(i)$ removing the exponential modality, and $(ii)$ instantiating the quantifiers. Lemma A.14 below accounts for such scenario: its proof relies on Corollary A.13 (Bounded Usage), the central result of the previous section.

LEMMA A.14. *Let $\Delta = \Delta', P \multimap !(C \multimap P)$ be weakly controlled and let $C$ be active in $\Delta$. If $\Delta \vdash P'$, then $\Delta' \vdash P'$.*

PROOF. By induction on the derivation of $\Delta \vdash P'$:

*Case* (IDENT): this rule cannot be applied, since $P \multimap !(C \multimap P)$ is not a payload formula;

*Case* (WEAK): if the principal formula is $P \multimap !(C \multimap P)$, the conclusion is immediate. Otherwise, let $\Delta' = \Delta'', F$ and let $\Delta \vdash P'$ by the hypothesis $\Delta'', P \multimap !(C \multimap P) \vdash P'$. By Lemma A.5 we know that the latter multiset is weakly controlled and $C$ is active by Lemma A.4, so we can apply the inductive hypothesis to get $\Delta'' \vdash P'$ and conclude $\Delta' \vdash P'$ by (WEAK);

*Case* (CONTR): let $\Delta' = \Delta'', !F$ and let $\Delta \vdash P'$ by the hypothesis $\Delta, !F \vdash P'$. By Lemma A.5 we know that the latter multiset is weakly controlled and $C$ is active by Lemma A.4, so we can apply the inductive hypothesis to get $\Delta', !F \vdash P'$ and conclude $\Delta' \vdash P'$ by (CONTR);

*Case* ($\otimes$-LEFT): let $\Delta' = \Delta'', F_1 \otimes F_2$ and let $\Delta \vdash P'$ from the hypothesis $\Delta'', F_1, F_2, P \multimap !(C \multimap P) \vdash P'$. By Lemma A.5 we know that the latter multiset is weakly controlled and $C$ is active by Lemma A.4, so we can apply the inductive hypothesis to get $\Delta'', F_1, F_2 \vdash P'$ and conclude $\Delta' \vdash P'$ by ($\otimes$-LEFT);

*Case* ($\otimes$-RIGHT): let $\Delta \vdash P_1 \otimes P_2$ by the hypotheses $\Delta_1 \vdash P_1$ and $\Delta_2 \vdash P_2$ with $\Delta = \Delta_1, \Delta_2$. By Lemma A.5 both $\Delta_1$ and $\Delta_2$ are weakly controlled and $C$ is active in both by Lemma A.4, so we can apply the inductive hypothesis to either $\Delta_1 \vdash P_1$ or $\Delta_2 \vdash P_2$, according to whether $P \multimap !(C \multimap P)$ occurs in $\Delta_1$ or in $\Delta_2$, and conclude by ($\otimes$-RIGHT);

*Case* ($\multimap$-LEFT): we distinguish two cases, according to the principal formula:

— let $\Delta' = \Delta'', F_1 \multimap F_2$ and let $\Delta \vdash P'$ by the hypotheses $\Delta_1 \vdash F_1$ and $\Delta_2, F_2 \vdash P'$ with $\Delta'', P \multimap !(C \multimap P) = \Delta_1, \Delta_2$. By Lemma A.5 we have that both $\Delta_1$ and $\Delta_2, F_2$ are weakly controlled and $C$ is active in both by Lemma A.4. Moreover, we note that $F_1$ is a payload formula. We are thus allowed to apply the inductive hypothesis to either $\Delta_1 \vdash F_1$ or $\Delta_2, F_2 \vdash P'$, according to whether $P \multimap !(C \multimap P)$ occurs in $\Delta_1$ or in $\Delta_2$, and conclude by ($\multimap$-LEFT);

— let $\Delta \vdash P'$ by the hypotheses $\Delta_1 \vdash P$ and $\Delta_2, !(C \multimap P) \vdash P'$ with $\Delta' = \Delta_1, \Delta_2$. By Lemma A.5 we have that $\Delta_2, !(C \multimap P)$ is weakly controlled and $C$ is active there by Lemma A.4, so by Corollary A.13 (Bounded Usage) we have $\Delta_2, P \vdash P'$. We can then conclude $\Delta_1, \Delta_2 \vdash P'$ by a standard Cut elimination argument;

*Case* ($\multimap$-RIGHT): the only possibility is that $\Delta \vdash B_1 \multimap B_2$ by the hypothesis $\Delta, B_1 \vdash B_2$. By Lemma A.8 we know that $\Delta, B_1$ is weakly controlled and $C$ is active, so by inductive hypothesis we get $\Delta', B_1 \vdash B_2$ and we conclude $\Delta' \vdash B_1 \multimap B_2$ by ($\multimap$-RIGHT);

*Case* ($\forall$-LEFT): let $\Delta' = \Delta'', \forall x.F$ and let $\Delta \vdash P'$ from $\Delta'', F\{t/x\}, P \multimap !(C \multimap P) \vdash P'$ for some $t$. By Lemma A.5 we know that the latter multiset is weakly controlled and $C$ is active by Lemma A.4, so we can apply the inductive hypothesis to get $\Delta'', F\{t/x\} \vdash P'$ and conclude $\Delta' \vdash P'$ by ($\forall$-LEFT);

*Case* ($\forall$-RIGHT): the only possibility is that $\Delta \vdash \forall x.B$ by $\Delta \vdash B$ with $x \notin fv(\Delta)$. By inductive hypothesis $\Delta' \vdash B$, so we conclude $\Delta' \vdash \forall x.B$ by ($\forall$-RIGHT);

*Case* (!-LEFT): let $\Delta' = \Delta'', !F$ and let $\Delta \vdash P'$ by the hypothesis $\Delta'', F, P \multimap !(C \multimap P) \vdash P'$. By Lemma A.5 we know that the latter multiset is weakly controlled and $C$ is active by Lemma A.4, so we can apply the inductive hypothesis to get $\Delta'', F \vdash P'$ and conclude $\Delta' \vdash P'$ by (!-LEFT);

*Case* (!-RIGHT): this rule cannot be applied, since $P \multimap !(C \multimap P)$ is not exponential.

□

Lemma A.15 strongly resembles Lemma A.14 and serves a similar purpose: a formula of the form $\forall\widetilde{x}.(P \multimap !(C \multimap P))$ is obtained from a serializer by removing the exponential modality from it. Since formulas of this form can arise in the proof of the main result, we must first deal with them.

LEMMA A.15. *Let* $\Delta = \Delta', \forall\widetilde{x}.(P \multimap !(C \multimap P))$ *be weakly controlled and let* $C$ *be active in* $\Delta$. *If* $\Delta \vdash P'$, *then* $\Delta' \vdash P'$.

PROOF. By induction on the derivation of $\Delta \vdash P'$, much as in the proof of Lemma A.14. We show just the cases which are actually different:

*Case* ($\multimap$-LEFT): if $\widetilde{x}$ is empty, we immediately conclude by Lemma A.14. Otherwise, let $\Delta' = \Delta'', F_1 \multimap F_2$ and let $\Delta \vdash P'$ by the hypotheses $\Delta_1 \vdash F_1$ and $\Delta_2, F_2 \vdash P'$ with $\Delta'', \forall\widetilde{x}.(P \multimap !(C \multimap P)) = \Delta_1, \Delta_2$. By Lemma A.5 we have that both $\Delta_1$ and $\Delta_2, F_2$ are weakly controlled and $C$ is active in both by Lemma A.4. Moreover, we note that $F_1$ is a payload formula. We are thus allowed to apply the inductive hypothesis to either $\Delta_1 \vdash F_1$ or $\Delta_2, F_2 \vdash P'$, according to whether $\forall\widetilde{x}.(P \multimap !(C \multimap P))$ occurs in $\Delta_1$ or in $\Delta_2$, and conclude by ($\multimap$-LEFT);
*Case* ($\forall$-LEFT): we have two cases. If the principal formula is $\forall\widetilde{x}.(P \multimap !(C \multimap P))$, we have again two possibilities: either every quantified variable is instantiated by the application of the rule or not. In the first case, we conclude by Lemma A.14; in the second case, the conclusion is immediate by inductive hypothesis. Otherwise, if the instantiated formula belongs to $\Delta'$, the conclusion follows by applying the inductive hypothesis and ($\forall$-LEFT).

□

The next lemma is the key to proving our main theorem: it states that serializers whose guard is active can be safely removed from a weakly controlled multiset without affecting the derivability of payload formulas.

LEMMA A.16 (WEAK SOUNDNESS). *Let* $\Delta = \Delta', !\forall\widetilde{x}.(P \multimap !(C \multimap P))$ *be weakly controlled and let* $C$ *be active in* $\Delta$. *If* $\Delta \vdash P'$, *then* $\Delta' \vdash P'$.

PROOF. We prove a stronger statement, namely:

$$\forall n \geq 0 : \Delta', (!\forall\widetilde{x}.(P \multimap !(C \multimap P)))^n \vdash P' \text{ implies } \Delta' \vdash P',$$

provided that the initial hypotheses are satisfied. We proceed by induction on the derivation of the judgement in the premise, much as in the proofs of Lemmas A.14 and A.15, and we show just the cases which are actually different:

*Case* (IDENT): if $n \geq 1$, the rule cannot be applied, so the conclusion is trivial;
*Case* (WEAK): if the principal formula is $!\forall\widetilde{x}.(P \multimap !(C \multimap P))$, the conclusion is immediate by inductive hypothesis. Otherwise, if the principal formula belongs to $\Delta'$, the conclusion follows by applying the inductive hypothesis and (WEAK);
*Case* (CONTR): we have two cases. If the principal formula is $!\forall\widetilde{x}.(P \multimap !(C \multimap P))$, the conclusion is immediate by inductive hypothesis. Otherwise, if the principal formula belongs to $\Delta'$, the conclusion follows by applying the inductive hypothesis and (CONTR);
*Case* ($\otimes$-RIGHT): let $\Delta \vdash P_1 \otimes P_2$ by the hypotheses $\Delta_1 \vdash P_1$ and $\Delta_2 \vdash P_2$ with $\Delta = \Delta_1, \Delta_2$. By Lemma A.5 both $\Delta_1$ and $\Delta_2$ are weakly controlled and $C$ is active in both

by Lemma A.4, so we can apply the inductive hypothesis to both $\Delta_1 \vdash P_1$ and $\Delta_2 \vdash P_2$, and conclude by ($\otimes$-RIGHT);

*Case* (!-LEFT): we have two cases. If the principal formula is $!\forall\widetilde{x}.(P \multimap !(C \multimap P))$, we first apply the inductive hypothesis and then we conclude by Lemma A.15. Otherwise, if the principal formula belongs to $\Delta'$, the conclusion follows by first applying the inductive hypothesis and then using (!-LEFT);

*Case* (!-RIGHT): let $\Delta \vdash !B$ by the hypothesis $\Delta \vdash B$ with $\Delta$ exponential. By inductive hypothesis $\Delta' \vdash B$, so we conclude $\Delta' \vdash !B$ by (!-RIGHT).

□

The next proposition is needed to identify a candidate serializer to remove in the proof of the main theorem, according to the explained proof strategy.

PROPOSITION A.17. *Let $\Delta = P_1, \ldots, P_m, S_1, \ldots, S_n$. If $n > 0$, then there exists $S_i \in \Delta$ such that $guard(S_i)$ is active in $\Delta$.*

In the next lemma we make explicit our proof strategy. The lemma immediately entails our real result of interest, Theorem 4.4 below.

LEMMA A.18. *Let $\Delta' = P_1, \ldots, P_m$. If $\Delta = \Delta', S_1, \ldots, S_n$ is weakly controlled and $\Delta \vdash P'$, then $\Delta' \vdash P'$.*

PROOF. By an appropriate number of applications of Lemma A.16 (Weak Soundness), using Proposition A.17 to identify a candidate serializer to remove at each application. □

RESTATEMENT 1 (OF THEOREM 4.4). *Let $\Delta' = P_1, \ldots, P_m$. If $\Delta = \Delta', S_1, \ldots, S_n$ is controlled and $\Delta \vdash P'$, then $\Delta' \vdash P'$.*

PROOF. Immediate by Lemma A.18, since any controlled multiset is also weakly controlled by Proposition A.3. □

### A.4. Proof of Proposition 4.5

The next proposition is a simple observation on the derivability of control formulas. It is needed in the proof of Proposition 4.5 below.

PROPOSITION A.19. *Let $\Delta = B_1, \ldots, B_l, C_1, \ldots, C_m$. If $\Delta \vdash C^k$, then $C$ occurs at least $k$ times in $\Delta$.*

PROOF. If $k = 0$, the result is trivial. Otherwise, we proceed by a simple induction on the derivation of $\Delta \vdash C^k$. □

Finally, we are ready to prove the correctness of our syntactic criterion for checking if a multiset is controlled.

RESTATEMENT 2 (OF PROPOSITION 4.5). *If $\Delta = B_1, \ldots, B_l, C_1, \ldots, C_m, S_1, \ldots, S_n$ is stratified and the control formulas in $\Delta$ are pairwise distinct, then $\Delta$ is controlled.*

PROOF. We first show that $\Delta$ is weakly controlled. Let $\Delta_1 = B_1, \ldots, B_l$, $\Delta_2 = C_1, \ldots, C_m$ and $\Delta_3 = S_1, \ldots, S_n$. Let us assume by contradiction that $\Delta \vdash C^h$ with $h \geq 2$ for some active control formula $C$. By Lemma A.6 (Stratification) we have $\Delta_2 \vdash C^h$, since any formula in $\Delta_1$ and $\Delta_3$ has an infinite rank, while the rank of $C$ is finite. By Proposition A.19, $C$ must occur at least $h$ times in $\Delta_2$, but this is contradictory with respect to the initial hypotheses.

Now we know that $\Delta$ is weakly controlled and we can show that it is, in fact, controlled. Let us assume by contradiction that $\Delta \vdash C^h$ with $h \geq 2$ for some arbitrary control formula $C$, then by Lemma A.18 we have $\Delta_1, \Delta_2 \vdash C^h$. By Proposition A.19,

$C$ must occur at least $h$ times in $\Delta_1, \Delta_2$, but this is contradictory with respect to the initial hypotheses. $\square$

## B. SOUNDNESS OF THE TYPE SYSTEM

We present a complete soundness proof for our type system. The structure of the proof is standard: we first establish a Subject Reduction theorem, which shows that types are preserved upon reduction, and then we prove that well-typed programs are *statically* safe. By combining these two guarantees, we establish that the type system enforces our safety notion. Finally, we prove an Opponent Typability lemma, which states that any opponent is trivially well-typed: this allows us to carry out a simple proof of robust safety, based on our safety theorem.

The present appendix is organized as follows:

— Appendix B.1 develops basic properties of affine logic, which are needed in the soundness proof of the type system;
— Appendix B.2 establishes some basic results about the type system and the environment rewriting relation;
— Appendix B.3 presents the main properties of kinding and subtyping, most notably the transitivity of the subtyping relation;
— Appendix B.4 establishes a standard substitution lemma;
— Appendix B.5 provides the inversion lemmas for the constructed values of our framework;
— Appendix B.6 presents the fundamental properties of the extraction relation;
— Appendix B.7 details the proof of the Subject Reduction theorem, building upon the results of the previous sections;
— Appendix B.8 presents the proof of robust safety.

### B.1. Properties of the logic

We first show that affine logic is closed under substitution of variables with closed terms. This is important to prove the substitution lemma of our type system.

LEMMA B.1 (SUBSTITUTION FOR THE LOGIC). *For all $\Delta, F$ and all substitutions $\sigma$ of variables with closed terms, it holds that $\Delta \vdash F$ implies $\Delta\sigma \vdash F\sigma$.*

PROOF. By induction on the derivation of $\Delta \vdash F$.

*Case* (IDENT): we can immediately conclude by (IDENT).
*Case* ($\forall$-RIGHT): we know that $F = \forall x.F'$ and:

$$\frac{\Delta \vdash F' \qquad x \notin fv(\Delta)}{\Delta \vdash \forall x.F'}$$

We define the slightly modified substitution $\sigma'$ as follows:

$$y\sigma' := \begin{cases} y\sigma & \text{if } x \neq y \\ y & \text{if } x = y \end{cases}$$

It follows that $(\forall x.F')\sigma = \forall x.(F'\sigma')$. Since $x \notin fv(\Delta)$, we know that $\Delta\sigma = \Delta\sigma'$. We apply the induction hypothesis to $\Delta \vdash F'$ and $\sigma'$, so we get $\Delta\sigma' \vdash F'\sigma'$. Using the previous observations, we conclude $\Delta\sigma \vdash (\forall x.F')\sigma$ by an application of ($\forall$-RIGHT). Notice that the rule can be applied, since $x \notin fv(\Delta\sigma)$ by the assumption that $\sigma$ does not introduce variables.
*Case* ($\forall$-LEFT): we know that $\Delta \triangleq \Delta', \forall x.F'$ and:

$$\frac{\Delta', F'\{t/x\} \vdash F}{\Delta', \forall x.F' \vdash F}$$

We define the slightly modified substitution $\sigma'$ as follows:

$$y\sigma' := \begin{cases} y\sigma & \text{if } x \neq y \\ y & \text{if } x = y \end{cases}$$

By the induction hypothesis we know that $(\Delta', F'\{t/x\})\sigma \vdash F\sigma$. This is equivalent to $\Delta'\sigma, (F'\{t/x\})\sigma \vdash F\sigma$, which is equivalent to $\Delta'\sigma, (F'\sigma')\{t\sigma/x\} \vdash F\sigma$ by the definition of $\sigma$ and $\sigma'$ and the fact that both $\sigma$ and $\sigma'$ do not introduce variables. We can apply ($\forall$-LEFT) to derive:

$$\frac{\Delta'\sigma, (F'\sigma')\{t\sigma/x\} \vdash F\sigma}{\Delta'\sigma, \forall x.(F'\sigma') \vdash F\sigma}$$

We know that by definition of $\sigma, \sigma'$ it holds that $\Delta'\sigma, \forall x.(F'\sigma') \vdash F\sigma$ is equivalent to $\Delta'\sigma, (\forall x.F')\sigma \vdash F\sigma$ and thus to $(\Delta', \forall x.F')\sigma \vdash F\sigma$, which is the conclusion.

*Case* (=-SUBST): we know that $\Delta \triangleq \Delta', t = t'$ and:

$$\frac{\exists \sigma' = mgu(t, t') \Rightarrow \Delta'\sigma' \vdash F\sigma'}{\Delta', t = t' \vdash F}$$

We need to show that $(\Delta', t = t')\sigma \vdash F\sigma$, which by definition of substitution is equivalent to showing that $\Delta'\sigma, t\sigma = t'\sigma \vdash F\sigma$. We distinguish two cases: if there does not exist a most general unifier $\sigma'' = mgu(t\sigma, t'\sigma)$, the premise for concluding by an application of (=-SUBST) is immediately met and we are done.
Otherwise, we know that there exists $\sigma'' = mgu(t\sigma, t'\sigma)$. By definition of most general unifier, we know that $(t\sigma)\sigma''$ and $(t'\sigma)\sigma''$ are identical, which in particular means that $\sigma \circ \sigma''$ is a unifier for $t$ and $t'$; this also implies the existence of a most general unifier $\sigma' = mgu(t, t')$ and a (potentially empty) substitution $\sigma'''$ such that $\sigma \circ \sigma'' = \sigma' \circ \sigma'''$. We can apply the induction hypothesis to $\Delta'\sigma' \vdash F\sigma'$ and $\sigma'''$ and derive that $(\Delta'\sigma')\sigma''' \vdash (F\sigma')\sigma'''$. As we have seen above, this is equivalent to $(\Delta'\sigma)\sigma'' \vdash (F\sigma)\sigma''$. We can then apply (=-SUBST) to conclude that:

$$\frac{\sigma'' = mgu(t\sigma, t'\sigma) \qquad (\Delta'\sigma)\sigma'' \vdash (F\sigma)\sigma''}{\Delta'\sigma, t\sigma = t'\sigma \vdash F\sigma}$$

*Case* (=-REFL): we can immediately conclude by (=-REFL).
*Case* (FALSE): we can immediately conclude by (FALSE).

In all other cases we apply the induction hypothesis to the premises of the rule and conclude by applying the rule again. □

In the next result we recall that we write $\Delta \vdash \Delta'$ to stand for $\Delta \vdash F_1 \otimes \ldots \otimes F_n$ whenever $\Delta' = F_1, \ldots, F_n$. If $\Delta'$ is empty, we let $\Delta \vdash \Delta'$ stand for $\Delta \vdash \mathbf{1}$.

LEMMA B.2 (PROPERTIES OF CONJUNCTION). *The following properties hold:*

(*1*) *For all $n \geq 0$, we have $\Delta, F_1, \ldots, F_n \vdash F$ iff $\Delta, F_1 \otimes \ldots \otimes F_n \vdash F$.*
(*2*) *For all $\Delta, \Delta'$ it holds that $\Delta' \subseteq \Delta$ implies $\Delta \vdash \Delta'$.*

PROOF. We proceed as follows:

(1) We show both directions separately:
— $\Delta, F_1 \otimes \ldots \otimes F_n \vdash F \Rightarrow \Delta, F_1, \ldots, F_n \vdash F$: by induction on $n$.
  — The case for $n = 1$ is trivial.
  — We show the case for $n = 2$ in detail.
    We know that $\Delta, F_1 \otimes F_2 \vdash F$ and need to show that $\Delta, F_1, F_2 \vdash F$.

We know that:

$$\dfrac{\textsc{Ident}\ \dfrac{}{F_1 \vdash F_1} \qquad \dfrac{}{F_2 \vdash F_2}\ \textsc{Ident}}{F_1, F_2 \vdash F_1 \otimes F_2}\ \otimes\text{-}\textsc{Right}$$

Since $F_1, F_2 \vdash F_1 \otimes F_2$ and $\Delta, F_1 \otimes F_2 \vdash F$ we can apply a Cut elimination argument to derive that $\Delta, F_1, F_2 \vdash F$.

— In the remaining cases $n > 2$ we know that $F_1 \otimes \ldots \otimes F_n$ actually denotes a formula of the form $(F_1 \otimes \ldots \otimes F_i) \otimes (F_{i+1} \otimes \ldots \otimes F_n)$, where $F_1 \otimes \ldots \otimes F_i$ and $F_{i+1} \otimes \ldots \otimes F_n$ also contain disambiguating parentheses, for $i \in \{1, \ldots, n-1\}$. We apply the induction hypothesis (for $2 < n$) to the top-level conjunction, which lets us derive that $\Delta, (F_1 \otimes \ldots \otimes F_i), (F_{i+1} \otimes \ldots \otimes F_n) \vdash F$. We then apply the induction hypothesis (for $i < n$) to $F_1 \otimes \ldots \otimes F_i$ and derive that $\Delta, F_1, \ldots, F_i, (F_{i+1} \otimes \ldots \otimes F_n) \vdash F$. Applying the induction hypothesis (for $n-i < n$) to $F_{i+1} \otimes \ldots \otimes F_n$ lets us conclude.

— $\Delta, F_1, \ldots, F_n \vdash F \Rightarrow \Delta, F_1 \otimes \ldots \otimes F_n \vdash F$ : by induction on $n$.
The case $n = 1$ is trivial, the case for $n = 2$ follows by ($\otimes$-LEFT). The remaining cases $n > 2$ follow by applying the induction hypothesis three times, similar to the previous direction.

(2) Let $\Delta' = \emptyset$. We interpret $\Delta \vdash \emptyset$ as $\Delta \vdash \mathbf{1}$, which is defined as $\Delta \vdash () = ()$. This immediately follows by ($=$-REFL).
Let $\Delta' = F_1, \ldots, F_n$ for some $F_1, \ldots, F_n$. By (IDENT) we know that $F_1 \otimes \ldots \otimes F_n \vdash F_1 \otimes \ldots \otimes F_n$. Using property (1) it follows that $F_1, \ldots, F_n \vdash F_1 \otimes \ldots \otimes F_n$, which is equivalent to $\Delta' \vdash \Delta'$ using our standard notation. We can conclude using (WEAK) repeatedly.

□

The next result is a generalization to multisets of formulas of the standard Cut rule, characteristic of sequent calculi presentation of formal logic.

LEMMA B.3 (MULTICUT). *If $\Delta \vdash \Delta'$ and $\Delta', \Delta'' \vdash \Delta'''$, then $\Delta, \Delta'' \vdash \Delta'''$.*

PROOF. Let $\Delta' = \emptyset$. We know that $\Delta'' \vdash \Delta'''$ and can immediately conclude by repeated applications of (WEAK). Let then $\Delta' = F_1, \ldots, F_n$ for some $F_1, \ldots, F_n$. We know that $\Delta \vdash F_1, \ldots, F_n$, which denotes $\Delta \vdash F_1 \otimes \ldots \otimes F_n$. Using Lemma B.2 we also know that $F_1 \otimes \ldots \otimes F_n, \Delta'' \vdash \Delta'''$. We can conclude by a standard Cut elimination argument. □

The next technical lemma formalizes the intuition that exponential formulas can be proved an arbitrary number of times.

LEMMA B.4 (PROPERTIES OF CONTRACTION). *The following properties hold:*

(*1*) *For all $\Delta$ it holds that $!\Delta \vdash !\Delta, !\Delta$.*
(*2*) *For all $\Delta, \Delta'$ it holds that if $\Delta \vdash !\Delta'$, then $\Delta \vdash !\Delta', !\Delta'$.*

PROOF. We proceed as follows:

(1) We know that $!\Delta, !\Delta \vdash !\Delta, !\Delta$ by Lemma B.2. We can conclude by applying (CONTR) to each element in $!\Delta$.
(2) Using property (1) we know that $!\Delta' \vdash !\Delta', !\Delta'$. Since $\Delta \vdash !\Delta'$, we can conclude using Lemma B.3.

□

### B.2. Basic results

The next results are completely standard. In the following we typically let $\mathcal{J}$ range over the judgements $\{\diamond, T, F, T :: k, T <: U, E : T\}$.

LEMMA B.5 (DERIVED JUDGEMENTS). *It holds that:*

(1) *If* $\Gamma; \Delta \vdash \diamond$*, then* $fnfv(\Delta) \subseteq dom(\Gamma)$ *and* $\forall \Delta' \subseteq \Delta : \Gamma; \Delta' \vdash \diamond$.
(2) *If* $\Gamma; \Delta \vdash \diamond$ *and* $(x : T) \in \Gamma$*, then* $T = \psi(T)$.
(3) *If* $\Gamma; \Delta \vdash T$*, then* $\Gamma; \emptyset \vdash \psi(T)$.
(4) *If* $\Gamma; \Delta \vdash T$*, then* $\Gamma; \Delta \vdash \diamond$ *and* $fnfv(T) \subseteq dom(\Gamma)$.
(5) *If* $\Gamma; \Delta \vdash F$*, then* $\Gamma; \Delta \vdash \diamond$ *and* $fnfv(F) \subseteq dom(\Gamma)$.
(6) *If* $\Gamma; \Delta \hookrightarrow \Gamma; \Delta'$*, then* $\Gamma; \Delta \vdash \diamond$ *and* $\Gamma; \Delta' \vdash \diamond$.
(7) *If* $\Gamma; \Delta \vdash T :: k$*, then* $\Gamma; \Delta \vdash T$.
(8) *If* $\Gamma; \Delta \vdash T <: T'$*, then* $\Gamma; \Delta \vdash T$ *and* $\Gamma; \Delta \vdash T'$.
(9) *If* $\Gamma; \Delta \vdash E : T$*, then* $\Gamma; \Delta \vdash T$ *and* $fnfv(E) \subseteq dom(\Gamma)$.

PROOF. By induction on the depth of the derivation of the judgements. □

LEMMA B.6 (JOINING ENVS). *If* $\Gamma; \Delta \vdash \diamond$ *and* $\Gamma; \Delta' \vdash \diamond$*, then* $\Gamma; \Delta, \Delta' \vdash \diamond$.

PROOF. By induction on the size of $\Delta'$, using Lemma B.5 (point 1). □

*Notation* 1 (*Environment Entry* $\eta$). We define an environment entry $\eta$ to be either a type environment entry $\mu$ or a formula $F$.

*Notation* 2 (*Environment Join* $\bullet$). We introduce the following notation for environment join:

$$(\Gamma; \Delta) \bullet \mu \triangleq \begin{cases} \Gamma, x : \psi(T); \Delta, forms(x : T) & \text{if } \mu = x : T \\ \Gamma, \mu; \Delta & \text{otherwise} \end{cases}$$

$$(\Gamma; \Delta) \bullet F \triangleq \Gamma; \Delta, F$$

$$(\Gamma; \Delta) \bullet (\Gamma'; \Delta') \triangleq \Gamma, \Gamma'; \Delta, \Delta'$$

LEMMA B.7 (WEAKENING). *If* $(\Gamma; \Delta) \bullet (\Gamma'; \Delta') \vdash \mathcal{J}$ *and* $(\Gamma; \Delta) \bullet \eta \bullet (\Gamma'; \Delta') \vdash \diamond$*, then* $(\Gamma; \Delta) \bullet \eta \bullet (\Gamma'; \Delta') \vdash \mathcal{J}$.

PROOF. By induction on the derivation of $(\Gamma; \Delta) \bullet (\Gamma'; \Delta') \vdash \mathcal{J}$. □

The next lemma establishes some basic properties of the rewriting relation. Intuitively, we show that this relation is consistent with logical entailment, in that it satisfies some expected properties which hold true for the latter.

LEMMA B.8 (PROPERTIES OF REWRITING). *The following statements hold true:*

(1) *If* $\Gamma; \Delta \vdash \diamond$ *and* $\Delta' \subseteq \Delta$*, then* $\Gamma; \Delta \hookrightarrow \Gamma; \Delta'$.
(2) *If* $\Gamma; \Delta_1 \hookrightarrow \Gamma; \Delta'_1$ *and* $\Gamma; \Delta_2 \hookrightarrow \Gamma; \Delta'_2$*, then* $\Gamma; \Delta_1, \Delta_2 \hookrightarrow \Gamma; \Delta'_1, \Delta'_2$.
(3) *If* $\Gamma; \Delta \hookrightarrow \Gamma; \Delta'$ *and* $\Gamma; \Delta' \hookrightarrow \Gamma; \Delta''$*, then* $\Gamma; \Delta \hookrightarrow \Gamma; \Delta''$.
(4) *If* $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$*, then* $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta', !\Delta'$.

PROOF. We proceed as follows:

(1) Since $\Delta' \subseteq \Delta$, we know that $\Gamma; \Delta' \vdash \diamond$ by Lemma B.5. By Lemma B.2 we know that $\Delta \vdash \Delta'$, hence $\Gamma; \Delta \hookrightarrow \Gamma; \Delta'$ by (REWRITE).
(2) By inverting (REWRITE) we know that $\Gamma; \Delta_1 \vdash \diamond$, $\Gamma; \Delta'_1 \vdash \diamond$, $\Gamma; \Delta_2 \vdash \diamond$, $\Gamma; \Delta'_2 \vdash \diamond$, $\Delta_1 \vdash \Delta'_1$ and $\Delta_2 \vdash \Delta'_2$. By Lemma B.6 we have $\Gamma; \Delta_1, \Delta_2 \vdash \diamond$ and $\Gamma; \Delta'_1, \Delta'_2 \vdash \diamond$. By ($\otimes$-RIGHT) we get $\Delta_1, \Delta_2 \vdash \Delta'_1, \Delta'_2$ from $\Delta_1 \vdash \Delta'_1$ and $\Delta_2 \vdash \Delta'_2$, hence we conclude $\Gamma; \Delta_1, \Delta_2 \hookrightarrow \Gamma; \Delta'_1, \Delta'_2$ by (REWRITE).

(3) By inverting (REWRITE) we know that $\Gamma; \Delta \vdash \diamond$, $\Gamma; \Delta' \vdash \diamond$, $\Gamma; \Delta'' \vdash \diamond$, $\Delta \vdash \Delta'$ and $\Delta' \vdash \Delta''$. By Lemma B.3 we know that $\Delta \vdash \Delta'$ and $\Delta' \vdash \Delta''$ imply $\Delta \vdash \Delta''$, hence we conclude $\Gamma; \Delta \hookrightarrow \Gamma; \Delta''$ by (REWRITE).
(4) By inverting (REWRITE) we know that $\Gamma; \Delta \vdash \diamond$, $\Gamma; !\Delta' \vdash \diamond$ and $\Delta \vdash !\Delta'$. Since $\Gamma; !\Delta' \vdash \diamond$ implies $\mathit{fnfv}(!\Delta') \subseteq \mathit{dom}(\Gamma)$ by Lemma B.5, we get $\Gamma; !\Delta', !\Delta' \vdash \diamond$ by multiple applications of (FORM ENV ENTRY). By Lemma B.4 we know that $\Delta \vdash !\Delta'$ implies $\Delta \vdash !\Delta', !\Delta'$, hence we conclude $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta', !\Delta'$ by using (REWRITE).

□

The next result states that, if an environment $\Gamma; \Delta$ can be rewritten to an environment $\Gamma; \Delta'$, then it can derive all the judgements provable by the latter.

LEMMA B.9 (REWRITE WEAK). *If* $\Gamma; \Delta' \vdash \mathcal{J}$ *and* $\Gamma; \Delta \hookrightarrow \Gamma; \Delta'$*, then* $\Gamma; \Delta \vdash \mathcal{J}$.

PROOF. We distinguish on $\mathcal{J}$:

(1) $\mathcal{J} = \diamond$: This case follows immediately by the definition of (REWRITE).
(2) $\mathcal{J} = T$: By definition of rule (TYPE) we know that $\Gamma; \Delta' \vdash \diamond$ and $\mathit{fnfv}(T) \subseteq \mathit{dom}(\Gamma)$. We also know that $\Gamma; \Delta \hookrightarrow \Gamma; \Delta'$, which by (REWRITE) implies $\Gamma; \Delta \vdash \diamond$. Since $\Gamma; \Delta \vdash \diamond$ and $\mathit{fnfv}(T) \subseteq \mathit{dom}(\Gamma)$, we conclude $\Gamma; \Delta \vdash T$ by using (TYPE).
(3) $\mathcal{J} = F$: By definition of rule (DERIVE) we know that $\Gamma; \Delta' \vdash \diamond$, $\mathit{fnfv}(F) \subseteq \mathit{dom}(\Gamma)$ and $\Delta' \vdash F$. We also know that $\Gamma; \Delta \hookrightarrow \Gamma; \Delta'$, which by (REWRITE) implies $\Gamma; \Delta \vdash \diamond$ and $\Delta \vdash \Delta'$. We can apply Lemma B.3 to $\Delta \vdash \Delta'$ and $\Delta' \vdash F$, and get $\Delta \vdash F$. Since $\Gamma; \Delta \vdash \diamond$, $\mathit{fnfv}(F) \subseteq \mathit{dom}(\Gamma)$ and $\Delta \vdash F$, we conclude $\Gamma; \Delta \vdash F$ by (DERIVE).
(4) $\mathcal{J} = T :: k$: We proceed by induction on the derivation of $\Gamma; \Delta' \vdash T :: k$. The cases (KIND VAR) and (KIND UNIT) follow immediately by proof part (1). The case (KIND REFINE PUBLIC) follows by proof part (2) and an application of the induction hypothesis. All other cases contain a rewriting statement of the form $\Gamma; \Delta' \hookrightarrow \Gamma; \Delta''$ among their hypotheses. By Lemma B.8 (point 3) it follows that $\Gamma; \Delta \hookrightarrow \Gamma; \Delta''$, thus allowing us to immediately conclude by applying the original rule.
(5) $\mathcal{J} = T <: U$: By induction on the derivation of $\Gamma; \Delta' \vdash T <: U$, using the same reasoning as in the previous case.
(6) $\mathcal{J} = E : T$: By induction on the derivation of $\Gamma; \Delta' \vdash E : T$, using the same reasoning as in the previous cases.

□

The next technical lemma states that rewriting does not introduce free variables.

LEMMA B.10 (REWRITING AND VARIABLES). *If* $x \notin \mathit{dom}(\Gamma)$ *and* $\Gamma; \Delta \hookrightarrow \Gamma; \Delta'$*, then* $x \notin \mathit{fv}(\Delta')$.

PROOF. Immediate by Lemma B.5 (point 1), since $\Gamma; \Delta \hookrightarrow \Gamma; \Delta'$ implies $\Gamma; \Delta' \vdash \diamond$ by inverting rule (REWRITE). □

The next lemma is an expected property of the refinement stripping function $\psi$, i.e., that it removes all the refinement formulas from a type.

LEMMA B.11 (SOUNDNESS OF $\psi$). *For every type* $T$*, we have* $\mathit{forms}(x : \psi(T)) = \emptyset$.

PROOF. By induction on the structure of $T$. □

The next lemma states that the stripping function $\psi$ is idempotent, i.e., there is no purpose in stripping refinements twice from the same type.

LEMMA B.12 (IDEMPOTENT $\psi$). *For every type* $T$*, we have* $\psi(\psi(T)) = \psi(T)$.

PROOF. By induction on the structure of $T$. $\square$

### B.3. Properties of kinding and subtyping

The next result states that, whenever a typing environment can assign a kind to a type $T$, then it can be rewritten so as to be split in two distinct components: the first one is exponential and it is needed to kind-check the structural information $\psi(T)$, while the second one can be used to derive the refinement formulas $forms(x : T)$ when $T$ is tainted. This result is extensively used in the proofs, most likely to deal with the subtleties introduced by environment splitting.

LEMMA B.13 (BARE KINDS). *If $\Gamma; \Delta \vdash T :: k$, then there exist $!\Delta'$ and $\Delta''$ such that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta', \Delta''$ and $\Gamma; !\Delta' \vdash \psi(T) :: k$. Moreover, if $k = \mathsf{tnt}$, we can also require $\Delta'' \vdash forms(x : T)$ for any $x \notin dom(\Gamma)$.*

PROOF. By induction on the derivation of $\Gamma; \Delta \vdash T :: k$:

*Case* (KIND VAR): assume that $\Gamma; \Delta \vdash \alpha :: k$ by the premise $(\alpha :: k) \in \Gamma$ with $\Gamma; \Delta \vdash \diamond$. Since $forms(x : \alpha) = \emptyset$ and $\psi(\alpha) = \alpha$, we just need to show that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$ for some $!\Delta'$ such that $\Gamma; !\Delta' \vdash \alpha :: k$. We note that $\Gamma; \Delta \vdash \diamond$ implies $\Gamma; \emptyset \vdash \diamond$ by Lemma B.5 (point 1), hence $\Gamma; \emptyset \vdash \alpha :: k$ by (KIND VAR). Since $\Gamma; \Delta \hookrightarrow \Gamma; \emptyset$ by Lemma B.8 (point 1), this is the desired conclusion.

*Case* (KIND UNIT): assume that $\Gamma; \Delta \vdash \mathsf{unit} :: k$ by the premise $\Gamma; \Delta \vdash \diamond$. Since $forms(x : \mathsf{unit}) = \emptyset$ and $\psi(\mathsf{unit}) = \mathsf{unit}$, we just need to show that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$ for some $!\Delta'$ such that $\Gamma; !\Delta' \vdash \mathsf{unit} :: k$. We note that $\Gamma; \Delta \vdash \diamond$ implies $\Gamma; \emptyset \vdash \diamond$ by Lemma B.5 (point 1), hence $\Gamma; \emptyset \vdash \mathsf{unit} :: k$ by (KIND UNIT). Since $\Gamma; \Delta \hookrightarrow \Gamma; \emptyset$ by Lemma B.8 (point 1), this is the desired conclusion.

*Case* (KIND FUN): assume that $\Gamma; \Delta \vdash x : T \to U :: k$ by the premises $\Gamma; !\Delta_1 \vdash T :: \overline{k}$ and $\Gamma, x : \psi(T); !\Delta_2 \vdash U :: k$ with $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2$. Since $\psi(x : T \to U) = x : T \to U$ and $forms(x : (x : T \to U)) = \emptyset$, the conclusion is immediate.

*Case* (KIND REFINE PUBLIC): assume that $\Gamma; \Delta \vdash \{x : T \mid F\} :: \mathsf{pub}$ by the premises $\Gamma; \Delta \vdash \{x : T \mid F\}$ and $\Gamma; \Delta \vdash T :: \mathsf{pub}$. By inductive hypothesis $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, \Delta_2$ for some $!\Delta_1, \Delta_2$ such that $\Gamma; !\Delta_1 \vdash \psi(T) :: \mathsf{pub}$. Since $\psi(\{x : T \mid F\}) = \psi(T)$ by definition, we can conclude.

*Case* (KIND REFINE TAINTED): We know that $\Gamma; \Delta \vdash \{x : T \mid F\} :: \mathsf{tnt}$ by the premises $\Gamma; \Delta_1 \vdash \psi(\{x : T \mid F\}) :: \mathsf{tnt}$ and $\Gamma, x : \psi(\{x : T \mid F\}); \Delta_2 \vdash forms(x : \{x : T \mid F\})$ with $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$. We apply the inductive hypothesis to get $\Gamma; \Delta_1 \hookrightarrow \Gamma; !\Delta_{11}, \Delta_{12}$ for some $!\Delta_{11}$ and $\Delta_{12}$ such that $\Gamma; !\Delta_{11} \vdash \psi(\psi(\{x : T \mid F\})) :: \mathsf{tnt}$ and $\Delta_{12} \vdash forms(x : \psi(\{x : T \mid F\}))$. Note that the former judgement is equivalent to $\Gamma; !\Delta_{11} \vdash \psi(\{x : T \mid F\}) :: \mathsf{tnt}$ by Lemma B.12. By inverting (DERIVE) we have $\Delta_2 \vdash forms(x : \{x : T \mid F\})$. Since $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_{11}, \Delta_2$ by Lemma B.8, we can conclude.

The cases for rules (KIND PAIR), (KIND SUM) and (KIND REC) are identical to the case for (KIND FUN). $\square$

The next technical lemma is needed in the proof of Lemma B.19 below. It states that the assignment of a public kind does not depend on the refinement formulas associated to the type, but only on structural information.

LEMMA B.14 (BARE KINDS REVERSE). *If $\Gamma; \Delta \vdash T$ and $\Gamma; \Delta \vdash \psi(T) :: \mathsf{pub}$, then $\Gamma; \Delta \vdash T :: \mathsf{pub}$.*

PROOF. By induction on the structure of $T$. In most cases $T = \psi(T)$, allowing us to immediately conclude. In the case where $T = \{x : U \mid F\}$, assume that $\Gamma; \Delta \vdash \psi(\{x : U \mid F\}) :: \mathsf{pub}$ and $\Gamma; \Delta \vdash \{x : U \mid F\}$. We observe that the latter implies $\Gamma; \Delta \vdash U$. By definition we have $\psi(\{x : U \mid F\}) = \psi(U)$, hence by inductive hypothesis we get

$\Gamma; \Delta \vdash U :: \mathsf{pub}$. Since $\Gamma; \Delta \vdash \{x : U \mid F\}$ by hypothesis and $\Gamma; \Delta \vdash U :: \mathsf{pub}$, we conclude $\Gamma; \Delta \vdash \{x : U \mid F\} :: \mathsf{pub}$ by (KIND REFINE PUBLIC). □

The next result is similar in spirit to Lemma B.13, but it applies to subtyping. Again the goal is to identify a possible rewriting of the typing environment such that the structural subtyping relation and the refinement formulas can be proved separately. This is needed in a number of places to deal with the complications introduced by environment splitting.

LEMMA B.15 (BARE SUBTYPES). *If $\Gamma; \Delta \vdash T <: U$, then there exist $!\Delta'$ and $\Delta''$ such that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta', \Delta''$ and $\Gamma; !\Delta' \vdash \psi(T) <: \psi(U)$ and $\Delta'', \mathit{forms}(x : T) \vdash \mathit{forms}(x : U)$ for any $x \notin \mathit{dom}(\Gamma)$.*

PROOF. By induction on the derivation of $\Gamma; \Delta \vdash T <: U$:

*Case* (SUB REFL): assume that $\Gamma; \Delta \vdash T <: T$ by the premise $\Gamma; \Delta \vdash T$. Since $\Gamma; \emptyset \vdash \psi(T)$ by Lemma B.5 (point 3), we have $\Gamma; \emptyset \vdash \psi(T) <: \psi(T)$ by (SUB REFL). Moreover, we note that $\mathit{forms}(x : T) \vdash \mathit{forms}(x : T)$. This leads to the desired conclusion, since $\Gamma; \Delta \hookrightarrow \Gamma; \emptyset$ by Lemma B.8 (point 1).

*Case* (SUB PUB TNT): assume that $\Gamma; \Delta \vdash T <: U$ by the premises $\Gamma; \Delta_1 \vdash T :: \mathsf{pub}$ and $\Gamma; \Delta_2 \vdash U :: \mathsf{tnt}$ with $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$. We apply Lemma B.13 to $\Gamma; \Delta_1 \vdash T :: \mathsf{pub}$ and we get $\Gamma; \Delta_1 \hookrightarrow \Gamma; !\Delta_{11}, \Delta_{12}$ for some $!\Delta_{11}, \Delta_{12}$ such that $\Gamma; !\Delta_{11} \vdash \psi(T) :: \mathsf{pub}$. Then we apply Lemma B.13 to $\Gamma; \Delta_2 \vdash U :: \mathsf{tnt}$ and we get $\Gamma; \Delta_2 \hookrightarrow \Gamma; !\Delta_{21}, \Delta_{22}$ for some $!\Delta_{21}, \Delta_{22}$ such that $\Gamma; !\Delta_{21} \vdash \psi(U) :: \mathsf{tnt}$ and $\Gamma; \Delta_{22} \vdash \mathit{forms}(x : U)$. By an application of (SUB PUB TNT) we then get $\Gamma; !\Delta_{11}, !\Delta_{21} \vdash \psi(T) <: \psi(U)$. Now we notice that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_{11}, \Delta_{12}, !\Delta_{21}, \Delta_{22}$ by Lemma B.8, which implies $\Gamma; \Delta \hookrightarrow \Gamma; (!\Delta_{11}, !\Delta_{21}), \Delta_{22}$ again by Lemma B.8, hence we conclude.

*Case* (SUB FUN): assume that $\Gamma; \Delta \vdash y : U_1 \to U_2 <: y : U_3 \to U_4$ by the premises $\Gamma; !\Delta_1 \vdash U_3 <: U_1$ and $\Gamma, y : \psi(U_3); !\Delta_2 \vdash U_2 <: U_4$ with $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2$. Since $\psi(y : U_1 \to U_2) = y : U_1 \to U_2$ and $\psi(y : U_3 \to U_4) = y : U_3 \to U_4$ and $\mathit{forms}(x : (y : U_1 \to U_2)) = \mathit{forms}(x : (y : U_3 \to U_4)) = \emptyset$, the conclusion is immediate.

*Case* (SUB REFINE): assume that $\Gamma; \Delta \vdash T <: U$ by the premises $\Gamma; \Delta_1 \vdash \psi(T) <: \psi(U)$ and $\Gamma, y : \psi(T); \Delta_2, \mathit{forms}(y : T) \vdash \mathit{forms}(y : U)$ with $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$. We apply the inductive hypothesis to $\Gamma; \Delta_1 \vdash \psi(T) <: \psi(U)$ and we get that there exist $!\Delta_{11}, \Delta_{12}$ such that $\Gamma; !\Delta_{11} \vdash \psi(\psi(T)) <: \psi(\psi(U))$ and $\Gamma; \Delta_1 \hookrightarrow \Gamma; !\Delta_{11}, \Delta_{12}$. The former judgement is equivalent to $\Gamma; !\Delta_{11} \vdash \psi(T) <: \psi(U)$ by Lemma B.12, while by inverting rule (DERIVE) we have $\Delta_2, \mathit{forms}(y : T) \vdash \mathit{forms}(y : U)$; hence, to conclude we just note that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_{11}, \Delta_2$ by Lemma B.8.

The cases for rules (SUB PAIR), (SUB SUM) and (SUB POS REC) are identical to the case for (SUB FUN). □

The next technical lemma is needed in the proof of Lemma B.19 below. It states that only refinement formulas are relevant for many judgements of our type system, so we can always replace a purely structural type $\psi(T)$ with any other (well-formed) purely structural type $\psi(T')$ in the typing environment.

LEMMA B.16 (REPLACING UNREFINED BINDINGS). *For all $\mathcal{J} \in \{\diamond, U, F, U :: k, U <: U'\}$ it holds that if $\Gamma, x : \psi(T), \Gamma'; \Delta \vdash \mathcal{J}$ and $\Gamma; \emptyset \vdash \psi(T')$, then $\Gamma, x : \psi(T'), \Gamma'; \Delta \vdash \mathcal{J}$. Moreover, the depth of the two derivations is the same.*

PROOF. We prove all statements separately by induction on the derivation of $\Gamma, x : \psi(T), \Gamma'; \Delta \vdash \mathcal{J}$, making use of Lemma B.5 when needed. □

*Definition* B.17 (*Compartmental Notation for Environments*). Let $\Gamma[(\mu_i)^{i\in\{1,\ldots,n\}}]$ denote the environment obtained by inserting the entries $\mu_1,\ldots,\mu_n$ at fixed positions between the entries of the environment $\Gamma$.

The next technical lemma is needed in the proof of Lemma B.19 below. Intuitively, it states that kinding annotations for type variables do not play any role for many judgements of our type system.

LEMMA B.18 (TYPE VARIABLES AND KINDING). *For all* $\Gamma = \Gamma_0[(\alpha_i)^{i\in\{1,\ldots,n\}}]$ *and* $\hat{\Gamma} = \Gamma_0[(\alpha_i :: k_i)^{i\in\{1,\ldots,n\}}]$ *it holds that:*

(*1*) $dom(\Gamma) = dom(\hat{\Gamma})$*;*
(*2*) $\Gamma;\Delta \vdash \diamond$ *if and only if* $\hat{\Gamma};\Delta \vdash \diamond$*;*
(*3*) $\Gamma;\Delta \hookrightarrow \Gamma;\Delta'$ *if and only if* $\hat{\Gamma};\Delta \hookrightarrow \hat{\Gamma};\Delta'$*;*
(*4*) $\Gamma;\Delta \vdash T$ *if and only if* $\hat{\Gamma};\Delta \vdash T$*;*
(*5*) $\Gamma;\Delta \vdash F$ *if and only if* $\hat{\Gamma};\Delta \vdash F$*;*
(*6*) *If* $\Gamma;\Delta \vdash T :: k$*, then* $\hat{\Gamma};\Delta \vdash T :: k$.

PROOF. We proceed as follows:

(1) We note that $dom(\alpha_i) = dom(\alpha_i :: k_i)$ by the definition of $dom$ and we easily conclude.
(2) $\Gamma;\Delta$ and $\hat{\Gamma};\Delta$ only differ in $\alpha_i$ and $\alpha_i :: k_i$ respectively. The statement follows noting that $dom(\alpha_i) = \{\alpha_i\} = dom(\alpha_i :: k_i)$.
(3) By definition of (REWRITE), using (2).
(4) By definition of (TYPE), using (1) and (2).
(5) By definition of (DERIVE), using (1) and (2).
(6) By induction on the derivation of $\Gamma;\Delta \vdash T :: k$, using the previous statements.

□

The next lemma states that any subtype of a public type is public, while any supertype of a tainted type is tainted. This is needed to prove Lemma B.20 below.

LEMMA B.19 (PUBLIC DOWN/TAINTED UP). *For all environments* $\Gamma;\Delta$ *and types* $T, T'$ *it holds that:*

(*1*) *If* $\Gamma;\Delta \vdash T <: T'$ *and* $\Gamma;\Delta' \vdash T' :: \mathsf{pub}$*, then* $\Gamma;\Delta,\Delta' \vdash T :: \mathsf{pub}$.
(*2*) *If* $\Gamma;\Delta \vdash T <: T'$ *and* $\Gamma;\Delta' \vdash T :: \mathsf{tnt}$*, then* $\Gamma;\Delta,\Delta' \vdash T' :: \mathsf{tnt}$.

PROOF. The lemma is an instance (for $n = 0$) of the following more general statement: For all environments $\Gamma;\Delta$ and types $T, T'$ such that $\Gamma = \Gamma_0[(\alpha_i)^{i\in\{1,\ldots,n\}}]$ and $\hat{\Gamma} = \Gamma_0[(\alpha_i :: k_i)^{i\in\{1,\ldots,n\}}]$ it holds:

(1) If $\Gamma;\Delta \vdash T <: T'$ and $\hat{\Gamma};\Delta' \vdash T' :: \mathsf{pub}$, then $\hat{\Gamma};\Delta,\Delta' \vdash T :: \mathsf{pub}$.
(2) If $\Gamma;\Delta \vdash T <: T'$ and $\hat{\Gamma};\Delta' \vdash T :: \mathsf{tnt}$, then $\hat{\Gamma};\Delta,\Delta' \vdash T' :: \mathsf{tnt}$.

Both statements are proved by simultaneous induction on the derivation of $\Gamma;\Delta \vdash T <: T'$. We distinguish the last applied subtyping rule and we often implicitly appeal to Lemma B.5 and Lemma B.18. Notice in particular that, by using Lemma B.5 and Lemma B.18, we can derive both $\Gamma;\Delta,\Delta' \vdash \diamond$ and $\hat{\Gamma};\Delta,\Delta' \vdash \diamond$.

*Case* (SUB REFL): In this case $T = T'$, hence we know in the two cases that:
(1) $\hat{\Gamma};\Delta' \vdash T :: \mathsf{pub}$. As seen above, we know that $\hat{\Gamma};\Delta,\Delta' \vdash \diamond$, hence $\hat{\Gamma};\Delta,\Delta' \vdash T :: \mathsf{pub}$ follows by Lemma B.7.

(2) $\hat{\Gamma}; \Delta' \vdash T' :: \mathsf{tnt}$. Using the same reasoning as in the previous case we can conclude that $\hat{\Gamma}; \Delta, \Delta' \vdash T' :: \mathsf{tnt}$ follows by Lemma B.7.

*Case* (SUB PUB TNT): In this case it holds that $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ such that $\Gamma; \Delta_1 \vdash T :: \mathsf{pub}$ and $\Gamma; \Delta_2 \vdash T' :: \mathsf{tnt}$. Notice again that $\Gamma; \Delta, \Delta' \vdash \diamond$ as before.

In the proof of statement (1) we need to show that $\hat{\Gamma}; \Delta, \Delta' \vdash T :: \mathsf{pub}$. By Lemma B.8 we know that $\Gamma; \Delta, \Delta' \hookrightarrow \Gamma; \Delta_1$. We derive that $\Gamma; \Delta, \Delta' \vdash T :: \mathsf{pub}$ by an application of Lemma B.9. We apply Lemma B.18 to conclude that $\hat{\Gamma}; \Delta, \Delta' \vdash T :: \mathsf{pub}$.

In the proof of statement (2) we need to show that $\Gamma; \Delta, \Delta' \vdash T' :: \mathsf{tnt}$. By Lemma B.8 we know that $\Gamma; \Delta, \Delta' \hookrightarrow \Gamma; \Delta_2$. We conclude by an application of Lemma B.9 that $\Gamma; \Delta, \Delta' \vdash T' :: \mathsf{tnt}$. Using Lemma B.18 we can conclude that $\hat{\Gamma}; \Delta, \Delta' \vdash T' :: \mathsf{tnt}$.

*Case* (SUB REFINE): In this case we know that $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ such that $\Gamma; \Delta_1 \vdash \psi(T) <: \psi(T')$ and $\Gamma, y : \psi(T); \Delta_2, \mathit{forms}(y : T) \vdash \mathit{forms}(y : T')$.

We show both statements separately. We first note that by Lemma B.5 we know that $\Gamma; \emptyset \vdash T$ and $\Gamma; \emptyset \vdash T'$ and thus by Lemma B.18 $\hat{\Gamma}; \emptyset \vdash T$ and $\hat{\Gamma}; \emptyset \vdash T'$.

(1) By Lemma B.13 we know that there exist $\Delta_1', \Delta_2'$ such that:
— $\hat{\Gamma}; \Delta' \hookrightarrow \hat{\Gamma}; !\Delta_1', \Delta_2'$,
— $\hat{\Gamma}; !\Delta_1' \vdash \psi(T') :: \mathsf{pub}$.
We can apply the induction hypothesis to derive that:

$$\hat{\Gamma}; \Delta_1, !\Delta_1' \vdash \psi(T) :: \mathsf{pub}.$$

By Lemma B.14 we can immediately derive that:

$$\hat{\Gamma}; \Delta_1, !\Delta_1' \vdash T :: \mathsf{pub}.$$

We can derive that $\hat{\Gamma}; \Delta, \Delta' \hookrightarrow \hat{\Gamma}; \Delta_1, !\Delta_1'$ using Lemma B.8 in combination with Lemma B.18, hence we conclude $\hat{\Gamma}; \Delta, \Delta' \vdash T :: \mathsf{pub}$ by Lemma B.9.

(2) By Lemma B.13 we know that there exist $\Delta_1', \Delta_2'$ such that:
— $\hat{\Gamma}; \Delta' \hookrightarrow \hat{\Gamma}; !\Delta_1', \Delta_2'$,
— $\hat{\Gamma}; !\Delta_1' \vdash \psi(T) :: \mathsf{tnt}$, and
— $\Delta_2' \vdash \mathit{forms}(y : T)$ for some $y \notin \mathit{dom}(\Gamma)$.
We can apply the induction hypothesis to derive that:

$$\hat{\Gamma}; \Delta_1, !\Delta_1' \vdash \psi(T') :: \mathsf{tnt}.$$

If $\psi(T') = T'$, we observe that $\hat{\Gamma}; \Delta, \Delta' \hookrightarrow \hat{\Gamma}; \Delta_1, !\Delta_1'$ by Lemma B.8 in combination with Lemma B.18, hence we conclude $\hat{\Gamma}; \Delta, \Delta' \vdash T :: \mathsf{tnt}$ by Lemma B.9.
Otherwise, we know that $T'$ is refined. We stated that $\Gamma, y : \psi(T); \Delta_2, \mathit{forms}(y : T) \vdash \mathit{forms}(y : T')$, thus, by inverting (DERIVE), we know that $\Delta_2, \mathit{forms}(y : T) \vdash \mathit{forms}(y : T')$. Using Lemma B.3 we get:

$$\Delta_2', \Delta_2 \vdash \mathit{forms}(y : T'),$$

hence, by applying (DERIVE) and some simple observations, we know that $\hat{\Gamma}, y : \psi(T'); \Delta_2', \Delta_2 \vdash \mathit{forms}(y : T')$. By (KIND REFINE TAINTED) we then get:

$$\frac{\hat{\Gamma}; \Delta_1, !\Delta_1' \vdash \psi(T') :: \mathsf{tnt} \qquad \hat{\Gamma}, y : \psi(T'); \Delta_2', \Delta_2 \vdash \mathit{forms}(y : T')}{\hat{\Gamma}; !\Delta_1', \Delta_2', \Delta_2 \vdash T' :: \mathsf{tnt}}$$

By Lemma B.8 in combination with Lemma B.18, we know that $\hat{\Gamma}; \Delta, \Delta' \hookrightarrow \hat{\Gamma}; \Delta_1, \Delta_2, !\Delta_1', \Delta_2'$, hence we conclude $\hat{\Gamma}; \Delta, \Delta' \vdash T' :: \mathsf{tnt}$ by Lemma B.9.

*Case* (SUB SUM): In this case we know that $T = T_1 + T_2$ and $T' = T_1' + T_2'$ and $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2$ such that $\Gamma; !\Delta_i \vdash T_i <: T_i'$ for $i \in \{1, 2\}$.

(1) By the definition of the only applicable kinding rule (KIND SUM) we also know that $\hat{\Gamma};\Delta' \hookrightarrow \hat{\Gamma};!\Delta'_1,!\Delta'_2$ such that $\hat{\Gamma};!\Delta'_i \vdash T'_i :: \mathsf{pub}$ for $i \in \{1,2\}$. We apply the induction hypothesis twice and derive that $\hat{\Gamma};!\Delta_i,!\Delta'_i \vdash T_i :: \mathsf{pub}$. Since we know that $\hat{\Gamma};\Delta,\Delta' \hookrightarrow \hat{\Gamma};!\Delta_1,!\Delta_2,!\Delta'_1,!\Delta'_2 = \hat{\Gamma};!\Delta_1,!\Delta'_1,!\Delta_2,!\Delta'_2$ by Lemma B.8 and Lemma B.18, we conclude $\hat{\Gamma};\Delta,\Delta' \vdash T :: \mathsf{pub}$ by an application of (KIND SUM).

(2) Analogous to the case for statement (1).

*Case* (SUB POS REC): We know that $T = \mu\alpha.U$ and $T' = \mu\alpha.U'$ and $\Gamma;\Delta \hookrightarrow \Gamma;!\Delta_1$ such that $\Gamma,\alpha;!\Delta_1 \vdash U <: U'$ and $\alpha$ occurs only positively in $U$ and $U'$.

(1) By the definition of the only applicable kinding rule (KIND REC) we also know that $\hat{\Gamma};\Delta' \hookrightarrow \hat{\Gamma};!\Delta'_1$ such that $\hat{\Gamma},\alpha :: \mathsf{pub};!\Delta'_1 \vdash U' :: \mathsf{pub}$. We define $\alpha_{n+1} \triangleq \alpha$ and $\Gamma' \triangleq \Gamma,\alpha = \Gamma_0[(\alpha_i)^{i\in\{1,\dots,n+1\}}]$. Furthermore, we define $k_{n+1} \triangleq \mathsf{pub}$ and $\hat{\Gamma}' \triangleq \hat{\Gamma},\alpha :: \mathsf{pub} = \Gamma_0[(\alpha_i :: k_i)^{i\in\{1,\dots,n+1\}}]$. We can thus apply the induction hypothesis and derive that $\hat{\Gamma}';!\Delta_1,!\Delta'_1 \vdash U :: \mathsf{pub}$, which is equivalent to $\hat{\Gamma},\alpha :: \mathsf{pub};!\Delta_1,!\Delta'_1 \vdash U :: \mathsf{pub}$. Since we know that $\hat{\Gamma};\Delta,\Delta' \hookrightarrow \hat{\Gamma};!\Delta_1,!\Delta'_1$ by Lemma B.8 and Lemma B.18, we conclude $\hat{\Gamma};\Delta,\Delta' \vdash \mu\alpha.U :: \mathsf{pub}$ by an application of (KIND REC).

(2) Analogous to the case for statement (1).

*Case* (SUB PAIR): In this case $T = x : T_1 * T_2$ and $T' = x : T'_1 * T'_2$. We know that $\Gamma;\Delta \hookrightarrow \Gamma;!\Delta_1,!\Delta_2$ such that $\Gamma;!\Delta_1 \vdash T_1 <: T'_1$ and $\Gamma,x:\psi(T_1);!\Delta_2 \vdash T_2 <: T'_2$.

(1) By the only applicable kinding rule (KIND PAIR), we have $\hat{\Gamma};\Delta' \hookrightarrow \hat{\Gamma};!\Delta'_1,!\Delta'_2$ such that $\hat{\Gamma};!\Delta'_1 \vdash T'_1 :: \mathsf{pub}$ and $\hat{\Gamma},x:\psi(T'_1);!\Delta'_2 \vdash T'_2 :: \mathsf{pub}$.
We apply the induction hypothesis to derive that:

$$\hat{\Gamma};!\Delta_1,!\Delta'_1 \vdash T_1 :: \mathsf{pub}.$$

We apply Lemma B.16 to transform $\hat{\Gamma},x:\psi(T'_1);!\Delta'_2 \vdash T'_2 :: \mathsf{pub}$ into:

$$\hat{\Gamma},x:\psi(T_1);!\Delta'_2 \vdash T'_2 :: \mathsf{pub},$$

allowing us to apply the induction hypothesis a second time to derive that:

$$\hat{\Gamma},x:\psi(T_1);!\Delta_2,!\Delta'_2 \vdash T_2 :: \mathsf{pub}.$$

We conclude $\hat{\Gamma};\Delta,\Delta' \vdash T :: \mathsf{pub}$ by an application of (KIND PAIR), since we know that $\hat{\Gamma};\Delta,\Delta' \hookrightarrow \hat{\Gamma};!\Delta_1,!\Delta'_1,!\Delta_2,!\Delta'_2$ by Lemma B.8 and Lemma B.18.

(2) Analogous to the case for statement (1).

*Case* (SUB FUN): In this case $T = x : T_1 \to T_2$ and $T' = x : T'_1 \to T'_2$. We know that $\Gamma;\Delta \hookrightarrow \Gamma;!\Delta_1,!\Delta_2$ such that $\Gamma;!\Delta_1 \vdash T'_1 <: T_1$ and $\Gamma,x:\psi(T'_1);!\Delta_2 \vdash T_2 <: T'_2$.

(1) By the only applicable kinding rule (KIND FUN), we have $\hat{\Gamma};\Delta' \hookrightarrow \hat{\Gamma};!\Delta'_1,!\Delta'_2$ such that $\hat{\Gamma};!\Delta'_1 \vdash T'_1 :: \mathsf{tnt}$ and $\hat{\Gamma},x:\psi(T'_1);!\Delta'_2 \vdash T'_2 :: \mathsf{pub}$.
We apply the induction hypothesis (2) to derive that:

$$\hat{\Gamma};!\Delta_1,!\Delta'_1 \vdash T_1 :: \mathsf{tnt}.$$

We apply the induction hypothesis (1) to derive that:

$$\hat{\Gamma},x:\psi(T'_1);!\Delta_2,!\Delta'_2 \vdash T_2 :: \mathsf{pub}.$$

We apply Lemma B.16 to transform $\hat{\Gamma},x:\psi(T'_1);!\Delta_2,!\Delta'_2 \vdash T_2 :: \mathsf{pub}$ into:

$$\hat{\Gamma},x:\psi(T_1);!\Delta_2,!\Delta'_2 \vdash T_2 :: \mathsf{pub}.$$

We conclude $\hat{\Gamma};\Delta,\Delta' \vdash T :: \mathsf{pub}$ by an application of (KIND FUN), using the fact that $\hat{\Gamma};\Delta,\Delta' \hookrightarrow \hat{\Gamma};!\Delta_1,!\Delta'_1,!\Delta_2,!\Delta'_2$ by Lemma B.8 and Lemma B.18.

(2) Analogous to the case for statement (1).

□

The next result is central to proving the transitivity of our subtyping relation. It establishes a standard characterization of public and tainted kinds: a type is public iff it is a subtype of Un, while it is tainted iff it is a supertype of Un.

LEMMA B.20 (PUBLIC TAINTED). *For all environments $\Gamma; \Delta$ and types $T$ we have:*

(1) $\Gamma; \Delta \vdash T :: \mathsf{pub}$ *if and only if* $\Gamma; \Delta \vdash T <: \mathsf{Un}$.
(2) $\Gamma; \Delta \vdash T :: \mathsf{tnt}$ *if and only if* $\Gamma; \Delta \vdash \mathsf{Un} <: T$.

PROOF. By definition $\mathsf{Un} \triangleq \mathsf{unit}$ and thus by (KIND UNIT) it holds that $\Gamma; \emptyset \vdash \mathsf{Un} :: \mathsf{pub}$ and $\Gamma; \emptyset \vdash \mathsf{Un} :: \mathsf{tnt}$. We can immediately prove the forward implication by applying the subtyping rule (SUB PUB TNT), since $\Gamma; \Delta \hookrightarrow \Gamma; \Delta$ by Lemma B.8. The reverse implication follows immediately by Lemma B.19. □

The next technical lemma details a relationship between the stripping function $\psi$ and the subtyping relation. It is invoked only once in the proof of transitivity for the subtyping relation.

LEMMA B.21 (SUBTYPING AND $\psi$). *The following statements hold true:*

(1) *If* $\Gamma; \emptyset \vdash T$*, then* $\Gamma; \emptyset \vdash T <: \psi(T)$.
(2) *If* $\Gamma; \Delta \vdash \psi(T) <: U$ *and* $\Gamma; \emptyset \vdash T$*, then* $\Gamma; \Delta \vdash T <: U$.

PROOF. We proceed as follows:

(1) By induction on the structure of $T$:
 — Whenever $T = \psi(T)$, we can conclude by an application of (SUB REFL).
 — Otherwise, we know that $T = \{x : U \mid F\}$. We know that $\Gamma; \emptyset \vdash T$, hence $\Gamma; \emptyset \vdash \psi(T)$ by Lemma B.5 (point 3). Applying (SUB REFL) lets us derive that $\Gamma; \emptyset \vdash \psi(T) <: \psi(T)$, which is equivalent to $\Gamma; \emptyset \vdash \psi(T) <: \psi(\psi(T))$ by Lemma B.12. Furthermore, we know that $\mathit{forms}(y : \psi(T)) = \emptyset$ by Lemma B.11, hence we have $\Gamma, x : \psi(T); \mathit{forms}(y : T) \vdash \mathit{forms}(y : \psi(T))$. We thus conclude $\Gamma; \emptyset \vdash T <: \psi(T)$ by an application of (SUB REFINE).
(2) By induction on the derivation of $\Gamma; \Delta \vdash \psi(T) <: U$. We distinguish upon the last applied subtyping rule:
 — In the case where the last applied rule was (SUB FUN), (SUB PAIR), (SUB SUM), or (SUB POS REC) we know that $T = \psi(T)$ and we can immediately conclude.
 — (SUB REFL): In this case we know that $U = \psi(T)$. We can thus conclude by an application of statement (1) and Lemma B.7.
 — (SUB PUB TNT): In this case we know that there exist $\Delta_1, \Delta_2$ such that $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ and $\Gamma; \Delta_1 \vdash \psi(T) :: \mathsf{pub}$ and $\Gamma; \Delta_2 \vdash U :: \mathsf{tnt}$. By Lemma B.14 we thus know that $\Gamma; \Delta_1 \vdash T :: \mathsf{pub}$, allowing us to conclude by an application of (SUB PUB TNT).
 — (SUB REFINE): In this case we know that $U$ must be refined. Furthermore, we know that there must exist $\Delta_1, \Delta_2$ such that $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ and $\Gamma; \Delta_1 \vdash \psi(\psi(T)) <: \psi(U)$ and $\Gamma, x : \psi(\psi(T)); \Delta_2, \mathit{forms}(x : \psi(T)) \vdash \mathit{forms}(x : U)$. Note that by Lemma B.12 we know that $\psi(\psi(T)) = \psi(T)$ and by Lemma B.11 we know that $\mathit{forms}(x : \psi(T)) = \emptyset$. Thus, it follows that $\Gamma; \Delta_1 \vdash \psi(T) <: \psi(U)$ and $\Gamma, x : \psi(T); \Delta_2 \vdash \mathit{forms}(x : U)$. We can apply Lemma B.7 to derive that $\Gamma, x : \psi(T); \Delta_2, \mathit{forms}(x : T) \vdash \mathit{forms}(x : U)$. This allows us to conclude by an application of (SUB REFINE).

□

We are finally ready to prove the transitivity of the subtyping relation. This is a standard formulation for an affine setting.

LEMMA B.22 (TRANSITIVITY). *If* $\Gamma;\Delta \vdash T <: T'$ *and* $\Gamma;\Delta' \vdash T' <: T''$*, then* $\Gamma;\Delta,\Delta' \vdash T <: T''$.

PROOF. By induction on the sum of the depth of the derivations of the antecedent judgements. We proceed by case analysis on the last subtyping rule $R_1$ applied in the derivation $\Gamma;\Delta \vdash T <: T'$ and the last applied rule $R_2$ in the derivation of $\Gamma;\Delta' \vdash T' <: T''$. We first note that by Lemma B.5 it must be the case that $\Gamma;\Delta \vdash \diamond$ and $\Gamma;\Delta' \vdash \diamond$ and thus by Lemma B.6 it holds that $\Gamma;\Delta,\Delta' \vdash \diamond$.

*Case* $R_1 =$ (SUB REFL): Since in this case $T = T'$, we can immediately conclude by applying Lemma B.7 to $\Gamma;\Delta' \vdash T' <: T''$.

*Case* $R_2 =$ (SUB REFL): Since in this case $T' = T''$, we can immediately conclude by applying Lemma B.7 to $\Gamma;\Delta \vdash T <: T'$.

*Case* $R_1 =$ (SUB PUB TNT): By definition of (SUB PUB TNT) it follows that $\Gamma;\Delta_1 \vdash T :: \mathsf{pub}$, $\Gamma;\Delta_2 \vdash T' :: \mathsf{tnt}$, where $\Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2$. We can apply Lemma B.19 to derive that $\Gamma;\Delta',\Delta_2 \vdash T'' :: \mathsf{tnt}$ and since we know that $\Gamma;\Delta,\Delta' \vdash \diamond$ and $\Gamma;\Delta,\Delta' \hookrightarrow \Gamma;\Delta_1,\Delta_2,\Delta'$ by Lemma B.8 we apply rule (SUB PUB TNT) to conclude.

*Case* $R_2 =$ (SUB PUB TNT): By definition of (SUB PUB TNT) it follows that $\Gamma;\Delta'_1 \vdash T' :: \mathsf{pub}$, $\Gamma;\Delta'_2 \vdash T'' :: \mathsf{tnt}$, where $\Gamma;\Delta' \hookrightarrow \Gamma;\Delta'_1,\Delta'_2$. We can apply Lemma B.19 to derive that $\Gamma;\Delta,\Delta'_1 \vdash T :: \mathsf{pub}$ and since we know that $\Gamma;\Delta,\Delta' \vdash \diamond$ and $\Gamma;\Delta,\Delta' \hookrightarrow \Gamma;\Delta,\Delta'_1,\Delta'_2$ we apply rule (SUB PUB TNT) to conclude.

*Case* $R_1 = R_2 =$ (SUB SUM): Follows immediately by applying the induction hypothesis twice to the premises of the applied rule (SUB SUM) and then applying (SUB SUM) to the resulting statements.

*Case* $R_1 = R_2 =$ (SUB POS REC): Follows immediately by applying the induction hypothesis to the premise of the applied rule (SUB POS REC) and then applying (SUB POS REC) to the resulting statement.

*Case* $R_1 =$ (SUB REFINE): In this case we know that $\Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2$ such that $\Gamma;\Delta_1 \vdash \psi(T) <: \psi(T')$ and $(\Gamma;\Delta_2) \bullet y : T \vdash \mathit{forms}(y : T')$.

We distinguish all possible rules $R_2$, that are not captured by previous cases:

— $R_2$ is either (SUB FUN), (SUB PAIR), (SUB SUM), or (SUB POS REC): In this case we know that $\psi(T') = T'$ and we can immediately apply the induction hypothesis to derive that:

$$\Gamma;\Delta_1,\Delta' \vdash \psi(T) <: T''.$$

By Lemma B.21 it follows that:

$$\Gamma;\Delta_1,\Delta' \vdash T <: T''.$$

By Lemma B.8 we know that $\Gamma;\Delta,\Delta' \hookrightarrow \Gamma;\Delta_1,\Delta'$, allowing us to conclude by an application of Lemma B.9.

— $R_2 =$ (SUB REFINE): In this case we know that $\Gamma;\Delta' \hookrightarrow \Gamma;\Delta'_1,\Delta'_2$ such that $\Gamma;\Delta'_1 \vdash \psi(T') <: \psi(T'')$ and $(\Gamma;\Delta'_2) \bullet y : T' \vdash \mathit{forms}(y : T'')$.

We can apply the induction hypothesis to $\Gamma;\Delta_1 \vdash \psi(T) <: \psi(T')$ and $\Gamma;\Delta'_1 \vdash \psi(T') <: \psi(T'')$, leading to:

$$\Gamma;\Delta_1,\Delta'_1 \vdash \psi(T) <: \psi(T'').$$

By the definition of "$\bullet$", inverting rule (DERIVE), we know that $\Delta_2, \mathit{forms}(y : T) \vdash \mathit{forms}(y : T')$ and $\Delta'_2, \mathit{forms}(y : T') \vdash \mathit{forms}(y : T'')$. Using Lemma B.3 we can derive that $\Delta_2, \Delta_2', \mathit{forms}(y : T) \vdash \mathit{forms}(y : T'')$. By applying rule (DERIVE) and Lemma B.16, we can then get:

$$(\Gamma; \Delta_2, \Delta_2') \bullet y : T \vdash \mathit{forms}(y : T'').$$

We also know by definition of (SUB REFINE) that $T$ and/or $T'$ refined and $T'$ and/or $T''$ refined. This implies that either $T'$ is the only refined type or at least one type in $\{T, T''\}$ is refined. In the latter case we can immediately conclude by an application of (SUB REFINE). In the former case we know that $\psi(T) = T$ and $\psi(T'') = T''$. Since $\Gamma; \Delta, \Delta' \hookrightarrow \Gamma; \Delta_1, \Delta_1'$ by Lemma B.8 and $\Gamma; \Delta_1, \Delta_1' \vdash \psi(T) <: \psi(T'') = T <: T''$, we conclude $\Gamma; \Delta, \Delta' \vdash T <: T''$ by Lemma B.9.

*Case* $R_2 =$ (SUB REFINE): In this case we know that $\Gamma; \Delta' \hookrightarrow \Gamma; \Delta_1', \Delta_2'$ such that $\Gamma; \Delta_1' \vdash \psi(T') <: \psi(T'')$ and $(\Gamma; \Delta_2') \bullet y : T' \vdash \mathit{forms}(y : T'')$.
Note that all possible rules $R_1$ that are not captured by previous cases (SUB FUN), (SUB PAIR), (SUB SUM), or (SUB POS REC) entail that $T = \psi(T)$ and $T' = \psi(T')$, and $T''$ must be refined by definition of (SUB REFINE).
In particular, this means that we can apply the induction hypothesis to $\Gamma; \Delta \vdash T <: T'$ and $\Gamma; \Delta_1' \vdash \psi(T') <: \psi(T'')$, yielding:

$$\Gamma; \Delta, \Delta_1' \vdash \psi(T) <: \psi(T'').$$

By inverting (DERIVE) we have $\Delta_2', \mathit{forms}(y : T') \vdash \mathit{forms}(y : T'')$. By Lemma B.11 we know that $\mathit{forms}(y : T') = \mathit{forms}(y : T) = \emptyset$, hence $\Delta_2', \mathit{forms}(y : T) \vdash \mathit{forms}(y : T'')$. By applying rule (DERIVE) and Lemma B.16, we can then get:

$$\Gamma, y : \psi(T); \Delta_2', \mathit{forms}(y : T) \vdash \mathit{forms}(y : T'').$$

We conclude by an application of (SUB REFINE).

*Case* $R_1 = R_2 =$ (SUB FUN): In this case $T = x : U \to V$, $T' = x : U' \to V'$, and $T'' = x : U'' \to V''$.
Furthermore, there must exist $\Delta_1, \Delta_2, \Delta_1', \Delta_2'$ such that:
— $\Gamma; \Delta \vdash \Gamma; !\Delta_1, !\Delta_2$,
— $\Gamma; !\Delta_1 \vdash U' <: U$,
— $\Gamma, x : \psi(U'); !\Delta_2 \vdash V <: V'$,
— $\Gamma; \Delta' \vdash \Gamma; !\Delta_1', !\Delta_2'$,
— $\Gamma; !\Delta_1' \vdash U'' <: U'$, and
— $\Gamma, x : \psi(U''); !\Delta_2' \vdash V' <: V''$.
We note that by applying Lemma B.16 to the third statement we get $\Gamma, x : \psi(U''); !\Delta_2 \vdash V <: V'$, where the depth of the derivation has not changed. We apply the inductive hypothesis to $\Gamma; !\Delta_1' \vdash U'' <: U'$ and $\Gamma; !\Delta_1 \vdash U' <: U$, and we get:

$$\Gamma; !\Delta_1', !\Delta_1 \vdash U'' <: U.$$

We apply the inductive hypothesis to $\Gamma, x : \psi(U''); !\Delta_2 \vdash V <: V'$ and $\Gamma, x : \psi(U''); !\Delta_2' \vdash V' <: V''$, and we get:

$$\Gamma, x : \psi(U''); !\Delta_2, !\Delta_2' \vdash V <: V''.$$

The conclusions $\Gamma; \Delta \vdash T <: T''$ follows by (SUB FUN).

*Case* $R_1 = R_2 =$ (SUB PAIR): Completely analogous to the previous case.

No other combination of rules is possible. □

### B.4. Properties of substitution

The next result establishes for value typing judgements a property we already showed for kinding and subtyping judgements. Namely, if a typing environment $\Gamma; \Delta$ can assign a type $T$ to a value $M$, then it can be rewritten into two distinct typing environments: an exponential environment $\Gamma; !\Delta'$ where $M$ is assigned the structural type $\psi(T)$ and a possibly non-exponential environment $\Gamma; \Delta''$ where the refinement formulas $\mathit{forms}(x : T)$ can be proved on the value $M$.

LEMMA B.23 (BARE TYPES). *Let $fv(M) = \emptyset$. If $\Gamma; \Delta \vdash M : T$, then there exist $!\Delta'$ and $\Delta''$ such that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta', \Delta''$ and $\Gamma; !\Delta' \vdash M : \psi(T)$ and $\Delta'' \vdash \mathit{forms}(x : T)\{M/x\}$ for any $x \notin dom(\Gamma)$.*

PROOF. By induction on the derivation of $\Gamma; \Delta \vdash M : T$:

*Case* (VAL VAR): assume that $\Gamma; \Delta \vdash y : T$ by the premise $(y : T) \in \Gamma$ with $\Gamma; \Delta \vdash \diamond$. We have $T = \psi(T)$ by Lemma B.5, hence $\mathit{forms}(x : T) = \emptyset$ by Lemma B.11 and we just need to show that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$ for some $!\Delta'$ such that $\Gamma; !\Delta' \vdash y : T$. Now we note that $\Gamma; \Delta \vdash \diamond$ implies $\Gamma; \emptyset \vdash \diamond$ by Lemma B.5, hence $\Gamma; \emptyset \vdash y : T$ by (VAL VAR). This leads to the desired conclusion, since $\Gamma; \Delta \hookrightarrow \Gamma; \emptyset$ by Lemma B.8.

*Case* (VAL UNIT): assume that $\Gamma; \Delta \vdash () : \mathsf{unit}$ by the premise $\Gamma; \Delta \vdash \diamond$. Since $\psi(\mathsf{unit}) = \mathsf{unit}$ and $\mathit{forms}(x : \mathsf{unit}) = \emptyset$, we just need to show that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$ for some $!\Delta'$ such that $\Gamma; !\Delta' \vdash () : \mathsf{unit}$. By Lemma B.5 we have $\Gamma; \emptyset \vdash \diamond$, hence $\Gamma; \emptyset \vdash () : \mathsf{unit}$ by (VAL UNIT). This leads to the desired conclusion, since $\Gamma; \Delta \hookrightarrow \Gamma; \emptyset$ by Lemma B.8.

*Case* (VAL FUN): assume that $\Gamma; \Delta \vdash \lambda y. E : y : U_1 \to U_2$ by the premise $(\Gamma; !\Delta') \bullet y : U_1 \vdash E : U_2$ with $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$. Since $\psi(y : U_1 \to U_2) = y : U_1 \to U_2$ and $\mathit{forms}(x : (y : U_1 \to U_2)) = \emptyset$, the conclusion is immediate.

*Case* (VAL REFINE): assume that $\Gamma; \Delta \vdash M : \{x : U \mid F\}$ by the premises $\Gamma; \Delta_1 \vdash M : U$ and $\Gamma; \Delta_2 \vdash F\{M/x\}$ with $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$. Notice that $\Gamma; \Delta_2 \vdash F\{M/x\}$ implies $\Delta_2 \vdash F\{M/x\}$ by inverting rule (DERIVE). By inductive hypothesis $\Gamma; \Delta_1 \hookrightarrow \Gamma; !\Delta_{11}, \Delta_{12}$ with $\Gamma; !\Delta_{11} \vdash M : \psi(U)$ and $\Delta_{12} \vdash \mathit{forms}(x : U)\{M/x\}$. Notice that the former is equivalent to $\Gamma; !\Delta_{11} \vdash M : \psi(\{x : U \mid F\})$ by definition. Now by applying ($\otimes$-RIGHT) we get $\Delta_{12}, \Delta_2 \vdash \mathit{forms}(x : U)\{M/x\} \otimes F\{M/x\}$, which is equivalent to $\Delta_{12}, \Delta_2 \vdash \mathit{forms}(x : \{x : U \mid F\})\{M/x\}$. Since $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_{11}, (\Delta_{12}, \Delta_2)$ by Lemma B.8, we can conclude.

*Case* (EXP SUBSUM): assume that $\Gamma; \Delta \vdash M : T$ by the premises $\Gamma; \Delta_1 \vdash M : U$ and $\Gamma; \Delta_2 \vdash U <: T$ with $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$. By inductive hypothesis $\Gamma; \Delta_1 \hookrightarrow \Gamma; !\Delta_{11}, \Delta_{12}$ with $\Gamma; !\Delta_{11} \vdash M : \psi(U)$ and $\Delta_{12} \vdash \mathit{forms}(x : U)\{M/x\}$. By the premise $\Gamma; \Delta_2 \vdash U <: T$ and Lemma B.15, we have $\Gamma; \Delta_2 \hookrightarrow \Gamma; !\Delta_{21}, \Delta_{22}$ with $\Gamma; !\Delta_{21} \vdash \psi(U) <: \psi(T)$ and $\Delta_{22}, \mathit{forms}(x : U) \vdash \mathit{forms}(x : T)$. Now we note that $x \notin dom(\Gamma)$ implies $x \notin fv(\Delta_{22})$ by Lemma B.10. Given that logical entailment is closed under substitution of closed values for variables by Lemma B.1, we get:

$$\Delta_{22}, \mathit{forms}(x : U)\{M/x\} \vdash \mathit{forms}(x : T)\{M/x\},$$

hence by Lemma B.3 we have:

$$\Delta_{12}, \Delta_{22} \vdash \mathit{forms}(x : T)\{M/x\}.$$

Moreover, by (EXP SUBSUM) we get $\Gamma; !\Delta_{11}, !\Delta_{21} \vdash M : \psi(T)$, hence we conclude, since $\Gamma; \Delta \hookrightarrow \Gamma; (!\Delta_{11}, !\Delta_{21}), (\Delta_{12}, \Delta_{22})$ by Lemma B.8.

The cases for rules (VAL PAIR), (VAL INL), (VAL INR) and (VAL FOLD) are identical to the case for (VAL FUN). □

The next lemma states that multiplicative conjunctions occurring in refinement types can be equivalently broken into their atomic components. This is needed in the proof of Lemma B.25 below.

LEMMA B.24 ($\otimes$ SUB). *If $\Gamma; \Delta \vdash \{x : T \mid F_1 \otimes F_2\}$, then $\Gamma; \emptyset \vdash \{x : T \mid F_1 \otimes F_2\} <:> \{x : \{x : T \mid F_1\} \mid F_2\}$.*

PROOF. By applying (SUB REFINE), using simple observations. □

The next result is a very convenient lemma, which is needed in the proof of our substitution lemma. It essentially states that, if a typing environment $\Gamma; \Delta$ can assign a type $T$ to a value $M$, then it can be rewritten into two distinct typing environments: an exponential environment $\Gamma; !\Delta'$ where $M$ is assigned the structural type $\psi(T)$ and a possibly non-exponential environment $\Gamma; \Delta''$ where $M$ is assigned the original type $T$. Hence, purely structural typing judgements can be proved arbitrarily many times. This is again needed to deal with the subtleties introduced by environment splitting.

LEMMA B.25 (AFFINE TYPING). *If $\Gamma; \Delta \vdash M : T$, then there exist $!\Delta'$ and $\Delta''$ such that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta', \Delta''$ and $\Gamma; !\Delta' \vdash M : \psi(T)$ and $\Gamma; \Delta'' \vdash M : T$.*

PROOF. Let $\Gamma; \Delta \vdash M : T$ and consider any $x \notin dom(\Gamma)$. By Lemma B.23 there exist $!\Delta'$ and $\Delta''$ such that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta', \Delta''$ and $\Gamma; !\Delta' \vdash M : \psi(T)$ and $\Delta'' \vdash forms(x : T)\{M/x\}$. By Lemma B.8 we have $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta', !\Delta', \Delta''$. Now we note that $\Gamma; !\Delta', \Delta'' \vdash M : \{x : \psi(T) \mid forms(x : T)\}$ by (VAL REFINE), hence $\Gamma; !\Delta', \Delta'' \vdash M : T$ by using (EXP SUBSUM) in combination with Lemma B.24. Hence, we proved $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta', (!\Delta', \Delta'')$ with $\Gamma; !\Delta' \vdash M : \psi(T)$ and $\Gamma; !\Delta', \Delta'' \vdash M : T$. □

The next simple lemma states that, if a value $M$ is assigned a refinement type $T$, then the refinement formulas $forms(x : T)$ can be proved on $M$ from the formulas in the typing environment.

LEMMA B.26 (FORMULAS). *If $\Gamma; \Delta \vdash M : T$ and $x \notin dom(\Gamma)$, then $\Delta \vdash forms(x : T)\{M/x\}$.*

PROOF. Since $\Gamma; \Delta \vdash M : T$ and $x \notin dom(\Gamma)$, we apply Lemma B.23 and we get that there exist $!\Delta', \Delta''$ such that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta', \Delta''$ and $\Delta'' \vdash forms(x : T)\{M/x\}$. By inverting (REWRITE) we know that $\Delta \vdash !\Delta', \Delta''$. By multiple applications of (WEAK) we get $!\Delta', \Delta'' \vdash forms(x : T)\{M/x\}$, hence $\Delta \vdash forms(x : T)\{M/x\}$ by Lemma B.3. □

The next lemma establishes some basic syntactic properties of substitution.

LEMMA B.27 (BASIC SUBSTITUTION). *The following statements hold true:*

(1) *For every type $T$, we have $\psi(T)\{M/x\} = \psi(T\{M/x\})$.*
(2) *If $x \neq y$, then $forms(y : T)\{M/x\} = forms(y : T\{M/x\})$.*

PROOF. Point (1) is proved by induction on the structure of $T$, while point (2) follows by definition of $forms$ and standard syntactic properties of substitution. □

Finally, we can state and prove our substitution lemma, showing that typing is preserved by substitution of closed values for variables with the same type. The statement is complicated by the necessity to join different environments, but the formulation is consistent with standard presentations of substructural type systems.

LEMMA B.28 (SUBSTITUTION). *Suppose that $\Gamma; \Delta \vdash M : U$ and $fv(M) = \emptyset$. The following statements hold true:*

(1) *If $(\Gamma; \Delta') \bullet x : U \bullet (\Gamma'; \Delta'') \vdash \diamond$, then $\Gamma, \Gamma'\{M/x\}; \Delta, (\Delta', \Delta'')\{M/x\} \vdash \diamond$.*
(2) *If $(\Gamma; \Delta') \bullet x : U \bullet (\Gamma'; \Delta'') \vdash F$, then $\Gamma, \Gamma'\{M/x\}; \Delta, (\Delta', \Delta'')\{M/x\} \vdash F\{M/x\}$.*
(3) *If $(\Gamma; \Delta') \bullet x : U \bullet (\Gamma'; \Delta'') \hookrightarrow \Gamma, x : \psi(U), \Gamma'; \Delta^*$, then $\Gamma, \Gamma'\{M/x\}; \Delta, (\Delta', \Delta'')\{M/x\} \hookrightarrow \Gamma, \Gamma'\{M/x\}; \Delta^*\{M/x\}$.*
(4) *If $(\Gamma; \Delta') \bullet x : U \bullet (\Gamma'; \Delta'') \vdash T$, then $\Gamma, \Gamma'\{M/x\}; \Delta, (\Delta', \Delta'')\{M/x\} \vdash T\{M/x\}$.*
(5) *If $(\Gamma; \Delta') \bullet x : U \bullet (\Gamma'; \Delta'') \vdash T :: k$, then $\Gamma, \Gamma'\{M/x\}; \Delta, (\Delta', \Delta'')\{M/x\} \vdash T\{M/x\} :: k$.*
(6) *If $(\Gamma; \Delta') \bullet x : U \bullet (\Gamma'; \Delta'') \vdash T <: T'$, then $\Gamma, \Gamma'\{M/x\}; \Delta, (\Delta', \Delta'')\{M/x\} \vdash T\{M/x\} <: T'\{M/x\}$.*

(7) *If* $(\Gamma;\Delta') \bullet x : U \bullet (\Gamma';\Delta'') \vdash E : T$*, then* $\Gamma, \Gamma'\{M/x\}; \Delta, (\Delta', \Delta'')\{M/x\} \vdash E\{M/x\} : T\{M/x\}$.

PROOF. The proof is by simultaneous induction on the derivation of the antecedent judgements:

(1) Rule (ENV EMPTY) cannot be applied. For the other two rules the conclusion follows by inductive hypothesis, using Lemma B.5 and standard syntactic properties of substitution.

(2) Let $(\Gamma;\Delta') \bullet x : U \bullet (\Gamma';\Delta'') \vdash F$. The previous typing environment is equivalent to $\Gamma, x : \psi(U), \Gamma'; \Delta', \mathit{forms}(x : U), \Delta''$, thus by inverting rule (DERIVE) we have $(\Gamma;\Delta') \bullet x : U \bullet (\Gamma';\Delta'') \vdash \diamond$ and $\mathit{fnfv}(F) \subseteq \mathit{dom}(\Gamma, x : \psi(U), \Gamma')$ and $\Delta', \mathit{forms}(x : U), \Delta'' \vdash F$.
By inductive hypothesis we have $\Gamma, \Gamma'\{M/x\}; \Delta, (\Delta', \Delta'')\{M/x\} \vdash \diamond$. Since $\Gamma;\Delta \vdash M : U$ implies $\mathit{fnfv}(M) \subseteq \mathit{dom}(\Gamma)$ by Lemma B.5, it is easy to observe that $\mathit{fnfv}(F\{M/x\}) \subseteq \mathit{dom}(\Gamma, \Gamma'\{M/x\})$. Given that logical entailment is closed under substitution of closed values for variables by Lemma B.1, we have:

$$(\Delta', \mathit{forms}(x : U), \Delta'')\{M/x\} \vdash F\{M/x\}.$$

Now we note that by Lemma B.26 we have $\Delta \vdash \mathit{forms}(x : U)\{M/x\}$, hence $\Delta, (\Delta', \Delta'')\{M/x\} \vdash F\{M/x\}$ by Lemma B.3.
The conclusion $\Gamma, \Gamma'\{M/x\}; \Delta, (\Delta', \Delta'')\{M/x\} \vdash F\{M/x\}$ then follows by applying (DERIVE).

(3) Let $(\Gamma;\Delta') \bullet x : U \bullet (\Gamma';\Delta'') \hookrightarrow \Gamma, x : \psi(U), \Gamma'; \Delta^*$. We first note that the environment $(\Gamma;\Delta') \bullet x : U \bullet (\Gamma';\Delta'')$ is equivalent to $\Gamma, x : \psi(U), \Gamma'; \Delta', \mathit{forms}(x : U), \Delta''$, thus by inverting rule (REWRITE) we have $(\Gamma;\Delta') \bullet x : U \bullet (\Gamma';\Delta'') \vdash \diamond$ and $\Delta', \mathit{forms}(x : U), \Delta'' \vdash \Delta^*$ and $(\Gamma;\emptyset) \bullet x : \psi(U) \bullet (\Gamma';\Delta^*) \vdash \diamond$.
We apply the inductive hypothesis to $(\Gamma;\Delta') \bullet x : U \bullet (\Gamma';\Delta'') \vdash \diamond$ and we get $\Gamma, \Gamma'\{M/x\}; \Delta, (\Delta', \Delta'')\{M/x\} \vdash \diamond$.
Given that logical entailment is closed under substitution of closed values for variables by Lemma B.1, we have:

$$(\Delta', \mathit{forms}(x : U), \Delta'')\{M/x\} \vdash \Delta^*\{M/x\},$$

Now we note that by Lemma B.26 we have $\Delta \vdash \mathit{forms}(x : U)\{M/x\}$, hence $\Delta, (\Delta', \Delta'')\{M/x\} \vdash \Delta^*\{M/x\}$ by Lemma B.3.
By Lemma B.25 we have $\Gamma;\Delta \hookrightarrow \Gamma; !\Delta_1, \Delta_2$ for some $!\Delta_1, \Delta_2$ such that $\Gamma; !\Delta_1 \vdash M : \psi(U)$ and $\Gamma;\Delta_2 \vdash M : U$. We then apply the inductive hypothesis to $(\Gamma;\emptyset) \bullet x : \psi(U) \bullet (\Gamma';\Delta^*) \vdash \diamond$ and we get $\Gamma, \Gamma'\{M/x\}; !\Delta_1, \Delta^*\{M/x\} \vdash \diamond$. By Lemma B.5 this implies $\Gamma, \Gamma'\{M/x\}; \Delta^*\{M/x\} \vdash \diamond$, hence we conclude by applying (REWRITE).

(4) We just need to consider rule (TYPE). The conclusion follows by inverting the rule, using point (1), Lemma B.5 and standard syntactic properties of substitution.

(5) Rules (KIND VAR) and (KIND UNIT) use point (1). Rule (KIND REFINE PUBLIC) uses point (3) and the inductive hypothesis. The rules involving both logical rewriting and splitting are the most interesting, we show (KIND PAIR) as an example.
Assume then that $(\Gamma;\Delta') \bullet x : U \bullet (\Gamma';\Delta'') \vdash y : T_1 * T_2 :: k$ by the premises $\Gamma, x : \psi(U), \Gamma'; !\Delta_1 \vdash T_1 :: k$ and $\Gamma, x : \psi(U), \Gamma', y : \psi(T_1); !\Delta_2 \vdash T_2 :: k$ with $(\Gamma;\Delta') \bullet x : U \bullet (\Gamma';\Delta'') \hookrightarrow \Gamma, x : \psi(U), \Gamma'; !\Delta_1, !\Delta_2$.
We note that we can state the two premises as $(\Gamma; !\Delta_1) \bullet x : \psi(U) \bullet (\Gamma';\emptyset) \vdash T_1 :: k$ and $(\Gamma; !\Delta_2) \bullet x : \psi(U) \bullet (\Gamma', y : \psi(T_1);\emptyset) \vdash T_2 :: k$. By Lemma B.25 in combination with Lemma B.8 we have that $\Gamma;\Delta \vdash M : U$ implies $\Gamma;\Delta \hookrightarrow \Gamma; !\Delta_1', !\Delta_1', \Delta_2'$ for some $!\Delta_1'$ and $\Delta_2'$ such that $\Gamma; !\Delta_1' \vdash M : \psi(U)$ and $\Gamma;\Delta_2' \vdash M : U$.

We now apply the inductive hypothesis twice, and get:

$$\Gamma, \Gamma'\{M/x\}; !\Delta_1', !\Delta_1\{M/x\} \vdash T_1\{M/x\} :: k,$$

and:

$$\Gamma, \Gamma'\{M/x\}, y : \psi(T_1)\{M/x\}; !\Delta_1', !\Delta_2\{M/x\} \vdash T_2\{M/x\} :: k.$$

Notice that, by Lemma B.27, the latter is equivalent to:

$$\Gamma, \Gamma'\{M/x\}, y : \psi(T_1\{M/x\}); !\Delta_1', !\Delta_2\{M/x\} \vdash T_2\{M/x\} :: k.$$

We then proceed by considering the premise $(\Gamma; \Delta') \bullet x : U \bullet (\Gamma'; \Delta'') \hookrightarrow \Gamma, x : \psi(U), \Gamma'; !\Delta_1, !\Delta_2$. We apply the inductive hypothesis (point 3) there and we get $\Gamma, \Gamma'\{M/x\}; \Delta_2', (\Delta', \Delta'')\{M/x\} \hookrightarrow \Gamma, \Gamma'\{M/x\}; (!\Delta_1, !\Delta_2)\{M/x\}$.
Now we note that:

$$\Gamma, \Gamma'\{M/x\}; \Delta, (\Delta', \Delta'')\{M/x\} \hookrightarrow \Gamma, \Gamma'\{M/x\}; (!\Delta_1', !\Delta_1\{M/x\}), (!\Delta_1', !\Delta_2\{M/x\}),$$

hence we can conclude by applying (KIND PAIR).

(6) Rule (SUB REFL) uses point (4). Rule (SUB PUB TNT) uses point (5). The remaining cases mostly rely on the same arguments applied to prove the case (KIND PAIR) of the previous point. We show (SUB REFINE) as an example case.
Assume then that $(\Gamma; \Delta') \bullet x : U \bullet (\Gamma'; \Delta'') \vdash T_1 <: T_2$ by the premises $\Gamma, x : \psi(U), \Gamma'; \Delta_1 \vdash \psi(T_1) <: \psi(T_2)$ and $(\Gamma, x : \psi(U), \Gamma'; \Delta_2) \bullet y : T_1 \vdash \mathit{forms}(y : T_2)$ with $(\Gamma; \Delta') \bullet x : U \bullet (\Gamma'; \Delta'') \hookrightarrow \Gamma, x : \psi(U), \Gamma'; \Delta_1, \Delta_2$.
We note that we can state the two premises as $(\Gamma; \Delta_1) \bullet x : \psi(U) \bullet (\Gamma'; \emptyset) \vdash \psi(T_1) <: \psi(T_2)$ and $(\Gamma; \Delta_2) \bullet x : \psi(U) \bullet (\Gamma', y : \psi(T_1); \mathit{forms}(y : T_1)) \vdash \mathit{forms}(y : T_2)$. By Lemma B.25 in combination with Lemma B.8 we have that $\Gamma; \Delta \vdash M : U$ implies $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1', !\Delta_1', \Delta_2'$ for some $!\Delta_1'$ and $\Delta_2'$ such that $\Gamma; !\Delta_1' \vdash M : \psi(U)$ and $\Gamma; \Delta_2' \vdash M : U$.
We now apply the inductive hypothesis twice, and get:

$$\Gamma, \Gamma'\{M/x\}; !\Delta_1', \Delta_1\{M/x\} \vdash \psi(T_1)\{M/x\} <: \psi(T_2)\{M/x\},$$

and:

$$\Gamma, \Gamma'\{M/x\}, y : \psi(T_1)\{M/x\}; !\Delta_1', (\Delta_2, \mathit{forms}(y : T_1))\{M/x\} \vdash \mathit{forms}(y : T_2)\{M/x\}.$$

By Lemma B.27, the former is equivalent to:

$$\Gamma, \Gamma'; !\Delta_1', \Delta_1\{M/x\} \vdash \psi(T_1\{M/x\}) <: \psi(T_2\{M/x\}),$$

while the latter is equivalent to:

$$\Gamma, \Gamma', y : \psi(T_1\{M/x\}); !\Delta_1', \Delta_2\{M/x\}, \mathit{forms}(y : T_1\{M/x\}) \vdash \mathit{forms}(y : T_2\{M/x\}).$$

We then proceed by considering the premise $(\Gamma; \Delta') \bullet x : U \bullet (\Gamma'; \Delta'') \hookrightarrow \Gamma, x : \psi(U), \Gamma'; \Delta_1, \Delta_2$. We apply the inductive hypothesis (point 3) there and we get $\Gamma, \Gamma'\{M/x\}; \Delta_2', (\Delta', \Delta'')\{M/x\} \hookrightarrow \Gamma, \Gamma'\{M/x\}; (\Delta_1, \Delta_2)\{M/x\}$.
Now we note that:

$$\Gamma, \Gamma'\{M/x\}; \Delta, (\Delta', \Delta'')\{M/x\} \hookrightarrow \Gamma, \Gamma'\{M/x\}; (!\Delta_1', \Delta_1\{M/x\}), (!\Delta_1', \Delta_2\{M/x\}),$$

hence we can conclude by applying (SUB REFINE).

(7) All cases follow by the previous points and the inductive hypothesis, using standard syntactic properties of substitution and replicating the same arguments as before.

□

### B.5. Inversion lemmas

The next result is a standard bound weakening lemma: any occurrence of a type $T$ in the typing environment can be safely replaced with a subtype $T'$.

LEMMA B.29 (BOUND WEAK). *Let* $\Gamma; \Delta \vdash T' <: T$. *If* $\Gamma, x : \psi(T), \Gamma'; \Delta', \mathit{forms}(x : T) \vdash \mathcal{J}$, *then* $\Gamma, x : \psi(T'), \Gamma'; \Delta, \Delta', \mathit{forms}(x : T') \vdash \mathcal{J}$.

PROOF. For each judgement $\mathcal{J}$ the proof proceeds by induction on the derivation of $\Gamma, x : \psi(T), \Gamma'; \Delta', \mathit{forms}(x : T) \vdash \mathcal{J}$. We frequently use the fact that $\mathit{dom}(\Gamma, x : \psi(T), \Gamma') = \mathit{dom}(\Gamma, x : \psi(T'), \Gamma')$. Furthermore, we often use Lemma B.7 implicitly.

(1) $\mathcal{J} = \diamond$: The induction proof uses the fact that by Lemma B.5 we know that $\mathit{fnfv}(T') \subseteq \mathit{dom}(\Gamma)$ and $\mathit{fnfv}(\Delta) \subseteq \mathit{dom}(\Gamma)$.
(2) $\mathcal{J} = U$: By Lemma B.5 we know that $\mathit{fnfv}(U) \subseteq \mathit{dom}(\Gamma, x : \psi(T), \Gamma')$, which means that also $\mathit{fnfv}(U) \subseteq \mathit{dom}(\Gamma, x : \psi(T'), \Gamma')$. We conclude by applying statement (1) and rule (TYPE).
(3) $\mathcal{J} = F$: The proof makes use of statement (1), Lemma B.5, and the fact that $\mathit{dom}(\Gamma, x : \psi(T), \Gamma') = \mathit{dom}(\Gamma, x : \psi(T'), \Gamma')$. Furthermore, it applies Lemma B.15 to $\Gamma; \Delta \vdash T' <: T$ (showing that formulas in $\Delta$ and $T'$ entail those in $T$) in combination with Lemma B.3 to conclude.
(4) $\mathcal{J} = U :: k$: The proof makes uses of the previous statements. It also uses Lemma B.16 to show that replacing $x : \psi(T)$ by $x : \psi(T')$ is safe. It applies Lemma B.15 to $\Gamma; \Delta \vdash T' <: T$ (showing that formulas in $\Delta$ and $T'$ entail those in $T$) in combination with Lemma B.3 to conclude.
(5) $\mathcal{J} = U <: U'$: The proof uses similar reasoning as the proof of statement (4) and makes use of the previous statements.
(6) $\mathcal{J} = E : U$: The proof makes use of the previous statements and relies on Lemma B.15 and Lemma B.3.

□

We now present two technical lemmas which are needed to establish the inversion result for iso-recursive type constructors.

LEMMA B.30 (TYPE VARIABLES AND KINDING). *If* $\Gamma, \alpha, \Gamma'; \Delta \vdash T :: k$, *then* $\alpha \notin \mathit{fnfv}(T)$.

PROOF. By induction on the derivation of $\Gamma, \alpha, \Gamma'; \Delta \vdash T :: k$.
The case (KIND UNIT) follows immediately, since $\mathit{fnfv}(\mathsf{unit}) = \emptyset$. The case (KIND VAR) implies that $T = \beta$ for some type variable $\beta$ and $\beta :: k \in (\Gamma, \alpha, \Gamma')$. It must be the case that $\beta \neq \alpha$, since $\Gamma, \alpha, \Gamma'; \Delta \vdash T :: k$ implies $\Gamma, \alpha, \Gamma'; \Delta \vdash \diamond$ by Lemma B.5 and having both $\alpha$ and $\alpha :: k$ in the same environment would violate the well-formedness conditions enforced by (TYPE ENV ENTRY), given that $\mathit{dom}(\alpha) = \mathit{dom}(\alpha :: k)$. The case (KIND REFINE TAINTED) follows by an application of the induction hypothesis to the first premise of the kinding rule, using the fact that $\alpha \in \mathit{fnfv}(T)$ if and only if $\alpha \in \mathit{fnfv}(\psi(T))$. The remaining cases follow by the induction hypothesis. □

LEMMA B.31 (TYPE SUBSTITUTION). *For all* $T, T'$ *such that* $T = \psi(T)$ *and* $T' = \psi(T')$ *it holds that:*

(1) *If* $\Gamma, \alpha, \Gamma'; \Delta \vdash \mathcal{J}$ *and* $\Gamma; \Delta' \vdash T$, *then* $\Gamma, (\Gamma'\{T/\alpha\}); \Delta, \Delta' \vdash \mathcal{J}\{T/\alpha\}$.
(2) *If* $\Gamma, \alpha :: k, \Gamma'; \Delta \vdash \diamond$ *and* $\Gamma; \Delta' \vdash T :: k$, *then* $\Gamma, (\Gamma'\{T/\alpha\}); \Delta, \Delta' \vdash \diamond$.
(3) *If* $\Gamma, \alpha :: k, \Gamma'; \Delta \vdash U$ *and* $\Gamma; \Delta' \vdash T :: k$, *then* $\Gamma, (\Gamma'\{T/\alpha\}); \Delta, \Delta' \vdash U\{T/\alpha\}$.
(4) *If* $\Gamma, \alpha :: k, \Gamma'; \Delta \vdash U :: k'$ *and* $\Gamma; \Delta' \vdash T :: k$, *then* $\Gamma, (\Gamma'\{T/\alpha\}); \Delta, \Delta' \vdash U\{T/\alpha\} :: k'$.
(5) *We have:*

— *If* $\Gamma, \alpha, \Gamma'; \emptyset \vdash U$ *and* $\alpha$ *only occurs positively in* $U$ *and* $\Gamma; !\Delta \vdash T <: T'$*, then* $\Gamma, (\Gamma'\{T/\alpha\}); !\Delta \vdash U\{T/\alpha\} <: U\{T'/\alpha\}$.
— *If* $\Gamma, \alpha, \Gamma'; \emptyset \vdash U$ *and* $\alpha$ *only occurs negatively in* $U$ *and* $\Gamma; !\Delta \vdash T <: T'$*, then* $\Gamma, (\Gamma'\{T/\alpha\}); !\Delta \vdash U\{T'/\alpha\} <: U\{T/\alpha\}$.

(*6*) *If* $\Gamma, \alpha, \Gamma'; \Delta \vdash U <: U'$ *and* $\alpha$ *only occurs positively in* $U, U'$ *and* $\Gamma; \Delta' \vdash T <: T'$*, then* $\Gamma, (\Gamma'\{T/\alpha\}); \Delta, \Delta' \vdash U\{T/\alpha\} <: U'\{T'/\alpha\}$.

PROOF. We would like to note that the core statements of this lemma are points (4) and (6), the other points are just needed to prove them. In particular, point (1) is often used in the proof of the later statements; point (2) is used in the proof of point (3), which in turn is used in the proof of point (4); point (5) is used in the proof of point (6). We provide a proof sketch below.

(1) By induction on the derivation of $\Gamma, \alpha, \Gamma'; \Delta \vdash \mathcal{J}$, making use of Lemma B.5 and Lemma B.6.
(2) By induction on the derivation of $\Gamma, \alpha :: k, \Gamma'; \Delta \vdash \diamond$, making use of Lemma B.5 and Lemma B.6.
(3) By the definition of (TYPE), using point (2), Lemma B.5 and Lemma B.6.
(4) Since we know that $T = \psi(T)$, we can apply Lemma B.13 in combination with Lemma B.8 to show that there exists a $\Delta''$ such that $\Gamma; \Delta' \hookrightarrow \Gamma; !\Delta''$ and $\Gamma; !\Delta'' \vdash T :: k$.
We now prove the following modified statement: if $\Gamma, \alpha :: k, \Gamma'; \Delta \vdash U :: k'$ and $\Gamma; !\Delta'' \vdash T :: k$, then $\Gamma, (\Gamma'\{T/\alpha\}); \Delta, !\Delta'' \vdash U\{T/\alpha\} :: k'$.
The proof proceeds by induction on the derivation of $\Gamma, \alpha :: k, \Gamma'; \Delta \vdash U :: k'$, using point (3) and making use of Lemma B.8 whenever needed to perform the rewriting $\Gamma; !\Delta'' \hookrightarrow \Gamma; !\Delta'', !\Delta''$ and apply the inductive hypothesis twice.
The conclusion then follows by Lemma B.9.
(5) We prove both points simultaneously by induction on the structure of $U$.
(6) Since we know that $T = \psi(T)$ and $T' = \psi(T')$, we can apply Lemma B.15 in combination with Lemma B.8 to show that there exists a $\Delta''$ such that $\Gamma; \Delta' \hookrightarrow \Gamma; !\Delta''$ and $\Gamma; !\Delta'' \vdash T <: T'$.
We now prove the following modified statement: if $\Gamma, \alpha, \Gamma'; \Delta \vdash U <: U'$ and $\alpha$ only occurs positively in $U, U'$ and $\Gamma; !\Delta'' \vdash T <: T'$, then $\Gamma, (\Gamma'\{T/\alpha\}); \Delta, !\Delta'' \vdash U\{T/\alpha\} <: U'\{T'/\alpha\}$.
The proof proceeds by induction on the derivation of $\Gamma, \alpha, \Gamma'; \Delta \vdash U <: U'$, using point (5) and making use of Lemma B.8 whenever needed to perform the rewriting $\Gamma; !\Delta'' \hookrightarrow \Gamma; !\Delta'', !\Delta''$ and apply the inductive hypothesis twice.
The conclusion then follows by Lemma B.9.

□

We can finally state and prove a number of inversion results for the constructed values of our framework. The goal is showing that the elementary components of these constructed values have indeed the expected types. There is a substantial amount of work to do, but the technical details are mostly standard.

LEMMA B.32 (INVERSION FOR FUNCTIONS). *The following statements hold:*

(*1*) *If* $\Gamma; \Delta \vdash \lambda x. E : V$*, then there exist* $\Delta_1, \Delta_2, T, U$ *such that* $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ *and* $\Gamma; \Delta_1 \vdash \lambda x. E : x : T \to U$ *(by a top-level application of* VAL FUN*) and* $\Gamma; \Delta_2 \vdash x : T \to U <: \psi(V)$.
(*2*) *If* $\Gamma; \Delta \vdash x : T \to U <: x : T' \to U'$*, then there exist* $\Delta_1, \Delta_2$ *such that* $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2$ *and* $\Gamma; !\Delta_1 \vdash T' <: T$ *and* $\Gamma, x : \psi(T'); !\Delta_2 \vdash U <: U'$.

(3) *If* $\Gamma; \Delta \vdash \lambda x.\, E : x : T \to U$*, then there exists a* $\Delta'$ *such that* $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$ *and* $(\Gamma; !\Delta') \bullet x : T \vdash E : U$.
(4) *If* $\Gamma; \Delta \vdash \lambda x.\, E : x : T \to U$*, then* $(\Gamma; \Delta) \bullet x : T \vdash E : U$.

PROOF. We show the four statements separately, using the first two results in the proof of the third.

(1) By induction on the derivation of $\Gamma; \Delta \vdash \lambda x.\, E : V$. We know that $\Gamma; \Delta \vdash \lambda x.\, E : V$. We distinguish three cases, depending on the last applied typing rule:

*Case* (VAL FUN): In this case we know that $V = x : T \to U$ for some $T, U$, hence $\psi(V) = V$. Since $\Delta; \emptyset \vdash \psi(V)$ by Lemma B.5, we immediately derive $\Gamma; \emptyset \vdash x : T \to U <: \psi(V)$ by (SUB REFL). Since $\Gamma; \Delta \hookrightarrow \Gamma; \Delta$ by Lemma B.8, we can conclude.

*Case* (VAL REFINE): In this case we know that $V = \{y : V' \mid F\}$ and $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ for some $\Delta_1, \Delta_2$ such that $\Gamma; \Delta_1 \vdash \lambda x.\, E : V'$ and $\Gamma; \Delta_2 \vdash F\{\lambda x.\, E/y\}$.
We can apply the induction hypothesis to $\Gamma; \Delta_1 \vdash \lambda x.\, E : V'$, letting us derive that there exist $\Delta_{11}, \Delta_{12}, T, U$ such that:
— $\Gamma; \Delta_1 \hookrightarrow \Gamma; \Delta_{11}, \Delta_{12}$,
— $\Gamma; \Delta_{11} \vdash \lambda x.\, E : x : T \to U$ by a top-level application of (VAL FUN), and
— $\Gamma; \Delta_{12} \vdash x : T \to U <: \psi(V')$.
By the definition of $\psi$ we know that $\psi(V) = \psi(V')$, thus we know that:
— $\Gamma; \Delta_{11} \vdash \lambda x.\, E : x : T \to U$ by a top-level application of (VAL FUN), and
— $\Gamma; \Delta_{12} \vdash x : T \to U <: \psi(V)$.
Since $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_{11}, \Delta_{12}$ by Lemma B.8, we can conclude.

*Case* (EXP SUBSUM): In this case we know that there exist $\Delta_1, \Delta_2$ such that $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ and $\Gamma; \Delta_1 \vdash \lambda x.\, E : V'$ and $\Gamma; \Delta_2 \vdash V' <: V$.
We can apply the induction hypothesis to $\Gamma; \Delta_1 \vdash \lambda x.\, E : V'$, letting us derive that there exist $\Delta_{11}, \Delta_{12}, T, U$ such that:
— $\Gamma; \Delta_1 \hookrightarrow \Gamma; \Delta_{11}, \Delta_{12}$,
— $\Gamma; \Delta_{11} \vdash \lambda x.\, E : x : T \to U$ by a top-level application of (VAL FUN), and
— $\Gamma; \Delta_{12} \vdash x : T \to U <: \psi(V')$.
We apply Lemma B.15 to $\Gamma; \Delta_2 \vdash V' <: V$ and we get that there exist $!\Delta_{21}, \Delta_{22}$ such that $\Gamma; \Delta_2 \hookrightarrow \Gamma; !\Delta_{21}, \Delta_{22}$ and $\Gamma; !\Delta_{21} \vdash \psi(V') <: \psi(V)$. Since $\Gamma; \Delta_2 \hookrightarrow \Gamma; !\Delta_{21}$ by Lemma B.8 point 1, we have $\Gamma; \Delta_2 \vdash \psi(V') <: \psi(V)$ by Lemma B.9. By transitivity of the subtyping relation (Lemma B.22) we thus have:

$$\Gamma; \Delta_{12}, \Delta_2 \vdash x : T \to U <: \psi(V),$$

which allows us to conclude.

(2) By induction on the derivation of $\Gamma; \Delta \vdash x : T \to U <: x : T' \to U'$. We implicitly use Lemma B.5 whenever needed. We distinguish three cases, depending on the last applied subtyping rule:

*Case* (SUB REFL): In this case we know that $T = T'$ and $U = U'$ and conclude by two applications of (SUB REFL) that $\Gamma; \emptyset \vdash T' <: T$ and $\Gamma; \emptyset \vdash U <: U'$. Using suitable alpha-renaming and Lemma B.7 to extend $\Gamma$ with $x : \psi(T')$ in the second judgement, we can conclude, since $\Gamma; \Delta \hookrightarrow \Gamma; \emptyset$ by Lemma B.8.

*Case* (SUB FUN): The statement follows immediately by the premises of the subtyping rule.

*Case* (SUB PUB TNT): In this case we know that $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ for some $\Delta_1, \Delta_2$ such that $\Gamma; \Delta_1 \vdash x : T \to U :: \mathsf{pub}$ and $\Gamma; \Delta_2 \vdash x : T' \to U' :: \mathsf{tnt}$.
By the only applicable kinding rule (KIND FUN) it follows that there exist $\Delta_{11}, \Delta_{12}$ and $\Delta_{21}, \Delta_{22}$ such that $\Gamma; \Delta_1 \hookrightarrow \Gamma; \Delta_{11}, \Delta_{12}$ and $\Gamma; \Delta_2 \hookrightarrow \Gamma; \Delta_{21}, \Delta_{22}$ such that $\Gamma; !\Delta_{11} \vdash T :: \mathsf{tnt}$ and $\Gamma; !\Delta_{21} \vdash T' :: \mathsf{pub}$ and $\Gamma, x : \psi(T); !\Delta_{12} \vdash U :: \mathsf{pub}$ and $\Gamma, x : \psi(T'); !\Delta_{22} \vdash U' :: \mathsf{tnt}$.

Applying (SUB PUB TNT) to $\Gamma; !\Delta_{11} \vdash T :: \mathsf{tnt}$ and $\Gamma; !\Delta_{21} \vdash T' :: \mathsf{pub}$ yields:

$$\Gamma; !\Delta_{11}, !\Delta_{21} \vdash T' <: T.$$

We apply Lemma B.16 to $\Gamma, x : \psi(T); !\Delta_{12} \vdash U :: \mathsf{pub}$ and we get:

$$\Gamma, x : \psi(T'); !\Delta_{12} \vdash U :: \mathsf{pub}.$$

Applying (SUB PUB TNT) to $\Gamma, x : \psi(T'); !\Delta_{12} \vdash U :: \mathsf{pub}$ and $\Gamma, x : \psi(T'); !\Delta_{22} \vdash U' :: \mathsf{tnt}$ yields:

$$\Gamma, x : \psi(T'); !\Delta_{12}, !\Delta_{22} \vdash U <: U',$$

thus allowing us to conclude.

(3) We know that $\Gamma; \Delta \vdash \lambda x. E : x : T \to U$ and $\psi(x : T \to U) = x : T \to U$ by definition. We apply part (1) and derive that there exist $\Delta_1, \Delta_2, T', U'$ such that:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$,
— $\Gamma; \Delta_1 \vdash \lambda x. E : x : T' \to U'$ by a top-level application of (VAL FUN), and
— $\Gamma; \Delta_2 \vdash x : T' \to U' <: x : T \to U$.
By the definition of (VAL FUN) the second statement lets us derive that:

$$\Gamma, x : \psi(T'); !\Delta'_1, \mathit{forms}(x : T') \vdash E : U',$$

for some $\Delta'_1$ such that $\Gamma; \Delta_1 \hookrightarrow \Gamma; !\Delta'_1$, which is equivalent to $(\Gamma; !\Delta'_1) \bullet x : T' \vdash E : U'$. Applying part (2) to the third statement yields that there exist $\Delta_{21}, \Delta_{22}$ such that:
— $\Gamma; \Delta_2 \hookrightarrow \Gamma; !\Delta_{21}, !\Delta_{22}$,
— $\Gamma; !\Delta_{21} \vdash T <: T'$, and
— $\Gamma, x : \psi(T); !\Delta_{22} \vdash U' <: U$.
Applying Lemma B.16 to the latter yields:

$$\Gamma, x : \psi(T'); !\Delta_{22} \vdash U' <: U.$$

We apply (EXP SUBSUM) to $(\Gamma; !\Delta'_1) \bullet x : T' \vdash E : U'$ and $\Gamma, x : \psi(T'); !\Delta_{22} \vdash U' <: U$, which leads to:

$$(\Gamma; !\Delta'_1, !\Delta_{22}) \bullet x : T' \vdash E : U.$$

Applying Lemma B.29 to the latter statement and $\Gamma; !\Delta_{21} \vdash T <: T'$ lets us derive:

$$(\Gamma; !\Delta'_1, !\Delta_{22}, !\Delta_{21}) \bullet x : T \vdash E : U.$$

Since $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'_1, !\Delta_{22}, !\Delta_{21}$ by Lemma B.8, we can conclude.

(4) Follows immediately from statement (3) by an application of Lemma B.9.

□

LEMMA B.33 (INVERSION FOR PAIRS). *The following statements hold:*

(1) *If* $\Gamma; \Delta \vdash (M, N) : V$*, then there exist* $\Delta_1, \Delta_2, T, U$ *such that* $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ *and* $\Gamma; \Delta_1 \vdash (M, N) : x : T * U$ *(by a top-level application of* VAL PAIR*) and* $\Gamma; \Delta_2 \vdash x : T * U <: \psi(V)$.

(2) *If* $\Gamma; \Delta \vdash x : T * U <: x : T' * U'$*, then there exist* $\Delta_1, \Delta_2$ *such that* $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2$ *and* $\Gamma; !\Delta_1 \vdash T <: T'$ *and* $\Gamma, x : \psi(T); !\Delta_2 \vdash U <: U'$.

(3) *If* $\Gamma; \Delta \vdash (M, N) : x : T * U$*, then there exist* $\Delta_1, \Delta_2$ *such that* $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2$ *and* $\Gamma; !\Delta_1 \vdash M : T$ *and* $\Gamma; !\Delta_2 \vdash N : U\{M/x\}$.

PROOF. We show the three statements separately, using the first two results in the proof of the third.

(1) By induction on the derivation of $\Gamma; \Delta \vdash (M, N) : V$. The proof is analogous to that of Lemma B.32, part (1).

(2) By induction on the derivation of $\Gamma; \Delta \vdash x : T * U <: x : T' * U'$. The proof is analogous to that of Lemma B.32, part (2).
(3) We know that $\Gamma; \Delta \vdash (M, N) : x : T * U$ and that $\psi(x : T * U) = x : T * U$. We apply part (1) and derive that there exist $\Delta_1, \Delta_2, T', U'$ such that:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$,
— $\Gamma; \Delta_1 \vdash (M, N) : x : T' * U'$ by a top-level application of (VAL PAIR), and
— $\Gamma; \Delta_2 \vdash x : T' * U' <: x : T * U$.
By the definition of (VAL PAIR) the second statement lets us derive:

$$\Gamma; !\Delta_{11} \vdash M : T',$$

and:

$$\Gamma; !\Delta_{12} \vdash N : U'\{M/x\},$$

for some $\Delta_{11}, \Delta_{12}$ such that $\Gamma; \Delta_1 \hookrightarrow \Gamma; !\Delta_{11}, !\Delta_{12}$.
We can also apply part (2) to the third statement, which let us derive that there exist $\Delta_{21}, \Delta_{22}$ such that:
— $\Gamma; \Delta_2 \hookrightarrow \Gamma; !\Delta_{21}, !\Delta_{22}$,
— $\Gamma; !\Delta_{21} \vdash T' <: T$, and
— $\Gamma, x : \psi(T'); !\Delta_{22} \vdash U' <: U$.
We apply (EXP SUBSUM) to $\Gamma; !\Delta_{11} \vdash M : T'$ and $\Gamma; !\Delta_{21} \vdash T' <: T$, which yields:

$$\Gamma; !\Delta_{11}, !\Delta_{21} \vdash M : T.$$

We know that $\Gamma, x : \psi(T'); !\Delta_{22} \vdash U' <: U$, which by applying Lemma B.7 implies that $(\Gamma; !\Delta_{22}) \bullet x : T' \vdash U' <: U$ (we implicitly use the definition of "$\bullet$").
Since $\Gamma; !\Delta_{11} \vdash M : T'$, we can apply Lemma B.28 to the latter statement and derive:

$$\Gamma; !\Delta_{11}, (!\Delta_{22}\{M/x\}) \vdash U'\{M/x\} <: U\{M/x\}.$$

Note, however, that since $x \notin dom(\Gamma)$ and $\Gamma; \Delta_2 \hookrightarrow \Gamma; !\Delta_{21}, !\Delta_{22}$, we know that $x \notin fv(\Delta_{22})$ by Lemma B.10. Thus, the previous judgement is equivalent to:

$$\Gamma; !\Delta_{11}, !\Delta_{22} \vdash U'\{M/x\} <: U\{M/x\}.$$

We apply (EXP SUBSUM) to $\Gamma; !\Delta_{12} \vdash N : U'\{M/x\}$ and $\Gamma; !\Delta_{11}, !\Delta_{22} \vdash U'\{M/x\} <: U\{M/x\}$, which leads to:

$$\Gamma; !\Delta_{12}, !\Delta_{11}, !\Delta_{22} \vdash N : U\{M/x\}.$$

Using Lemma B.8 we know that:

$$\Gamma; \Delta \hookrightarrow \Gamma; (!\Delta_{11}, !\Delta_{21}), (!\Delta_{12}, !\Delta_{11}, !\Delta_{22}),$$

which allows us to conclude.

□

LEMMA B.34 (INVERSION FOR SUM CONSTRUCTORS). *The following statements hold:*

(1) *Let* $h \in \{\mathsf{inl}, \mathsf{inr}\}$. *If* $\Gamma; \Delta \vdash h\ M : V$, *then there exist* $\Delta_1, \Delta_2, T, U$ *such that* $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ *and* $\Gamma; \Delta_1 \vdash h\ M : T + U$ *(by a top-level application of* VAL H*) and* $\Gamma; \Delta_2 \vdash T + U <: \psi(V)$.
(2) *If* $\Gamma; \Delta \vdash T + U <: T' + U'$, *then there exist* $\Delta_1, \Delta_2$ *such that* $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2$ *and* $\Gamma; !\Delta_1 \vdash T <: T'$ *and*
(3) *If* $\Gamma; \Delta \vdash \mathsf{inl}\ M : T + U$, *then there exist* $!\Delta$ *such that* $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta$ *and* $\Gamma; !\Delta \vdash M : T$ *and* $\Gamma; !\Delta \vdash U$.

(*4*) *If* $\Gamma;\Delta \vdash \mathsf{inr}\ M : T+U$*, then there exist* $\Delta'$ *such that* $\Gamma;\Delta \hookrightarrow \Gamma;!\Delta'$ *and* $\Gamma;!\Delta' \vdash M : U$ *and* $\Gamma;!\Delta' \vdash T$.
(*5*) *If* $\Gamma;\Delta \vdash \mathsf{inl}\ M : T+U$*, then* $\Gamma;\Delta \vdash M : T$.
(*6*) *If* $\Gamma;\Delta \vdash \mathsf{inr}\ M : T+U$*, then* $\Gamma;\Delta \vdash M : U$.

PROOF. We show the six statements separately, using the first results in the proof of the later ones.

(1) By induction on the derivation of $\Gamma;\Delta \vdash h\ M : V$. The proof is analogous to that of Lemma B.32, part (1).
(2) By induction on the derivation of $\Gamma;\Delta \vdash T+U <: T'+U'$. The proof is analogous to that of Lemma B.32, part (2).
(3) We know that $\Gamma;\Delta \vdash \mathsf{inl}\ M : T+U$ and that $\psi(T+U) = T+U$. We apply part (1) and derive that there exist $\Delta_1, \Delta_2, T', U'$ such that:
— $\Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2$,
— $\Gamma;\Delta_1 \vdash \mathsf{inl}\ M : T'+U'$ by a top-level application of (VAL INL), and
— $\Gamma;\Delta_2 \vdash T'+U' <: T+U$.
By the definition of (VAL INL) the second statement lets us derive that:
$$\Gamma;!\Delta_1' \vdash M : T',$$
and:
$$\Gamma;!\Delta_1' \vdash U',$$
for some $\Delta_1'$ such that $\Gamma;\Delta_1 \hookrightarrow \Gamma;!\Delta_1'$.
Applying part (2) to the third statement yields that there exist $\Delta_{21}, \Delta_{22}$ such that:
— $\Gamma;\Delta_2 \hookrightarrow \Gamma;!\Delta_{21},!\Delta_{22}$,
— $\Gamma;!\Delta_{21} \vdash T' <: T$, and
— $\Gamma;!\Delta_{22} \vdash U' <: U$.
We apply (EXP SUBSUM) to $\Gamma;!\Delta_1' \vdash M : T'$ and $\Gamma;\Delta_{21} \vdash T' <: T$, which leads to:
$$\Gamma;!\Delta_1',!\Delta_{21} \vdash M : T.$$
Furthermore, by Lemma B.5 we know that:
$$\Gamma;!\Delta_{22} \vdash U.$$
Using Lemma B.7 we can derive that:
$$\Gamma;!\Delta_1',!\Delta_{21},!\Delta_{22} \vdash M : T,$$
and:
$$\Gamma;!\Delta_1',!\Delta_{21},!\Delta_{22} \vdash U.$$
Since $\Gamma;\Delta \hookrightarrow \Gamma;!\Delta_1',!\Delta_{21},!\Delta_{22}$ by Lemma B.8, we can conclude.
(4) The proof follows analogously to that of statement (3).
(5) The statement follows immediately by an application of statement (3) and Lemma B.9.
(6) The statement follows immediately by an application of statement (4) and Lemma B.9.

□

LEMMA B.35 (INVERSION FOR RECURSIVE CONSTRUCTORS). *The following statements hold:*

(*1*) *If* $\Gamma;\Delta \vdash \mathsf{fold}\ M : V$*, then there exist* $\Delta_1, \Delta_2, T$ *such that* $\Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2$ *and* $\Gamma;\Delta_1 \vdash \mathsf{fold}\ M : \mu\alpha.T$ *(by a top-level application of* VAL FOLD*) and* $\Gamma;\Delta_2 \vdash \mu\alpha.T <: \psi(V)$.

(2) *If* $\Gamma; \Delta \vdash \mu\alpha.T <: \mu\alpha.T'$*, then there exists* $\Delta'$ *such that* $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$ *and* $\Gamma; !\Delta' \vdash T\{\mu\alpha.T/\alpha\} <: T'\{\mu\alpha.T'/\alpha\}$.
(3) *If* $\Gamma; \Delta \vdash \mathsf{fold}\ M : \mu\alpha.T$*, then there exist* $\Delta'$ *such that* $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$ *and* $\Gamma; !\Delta' \vdash M : T\{\mu\alpha.T/\alpha\}$.
(4) *If* $\Gamma; \Delta \vdash \mathsf{fold}\ M : \mu\alpha.T$*, then* $\Gamma; \Delta \vdash M : T\{\mu\alpha.T/\alpha\}$.

PROOF. We show the four statements separately, using the first two results in the proof of the third.

(1) By induction on the derivation of $\Gamma; \Delta \vdash \mathsf{fold}\ M : V$. The proof is analogous to that of Lemma B.32, part (1).
(2) By induction on the derivation of $\Gamma; \Delta \vdash \mu\alpha.T <: \mu\alpha.T'$. We implicitly use Lemma B.5 whenever needed. We distinguish three cases, depending on the last applied subtyping rule:

*Case* (SUB REFL): In this case we know that $T = T'$ and thus we have $T\{\mu\alpha.T/\alpha\} = T'\{\mu\alpha.T'/\alpha\}$. By an application of (SUB REFL) we have $\Gamma; \emptyset \vdash T\{\mu\alpha.T/\alpha\} <: T'\{\mu\alpha.T'/\alpha\}$. Since $\Gamma; \Delta \hookrightarrow \Gamma; \emptyset$ by Lemma B.8, we can conclude.

*Case* (SUB POS REC): By the premises of the subtyping rule we know that:

$$\Gamma, \alpha; !\Delta' \vdash T <: T'$$

for some $\Delta'$ such that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$. Moreover, we know that $\alpha$ occurs only positively in $T, T'$. Since $\Gamma; !\Delta' \hookrightarrow \Gamma; !\Delta'$ by Lemma B.8, we can apply (SUB POS REC) to derive that:

$$\Gamma; !\Delta' \vdash \mu\alpha.T <: \mu\alpha.T'.$$

By point (6) of Lemma B.31, we then get:

$$\Gamma; !\Delta', !\Delta' \vdash T\{\mu\alpha.T/\alpha\} <: T'\{\mu\alpha.T'/\alpha\}.$$

Since $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta', !\Delta'$ by Lemma B.8, we can conclude.

*Case* (SUB PUB TNT): In this case we know that $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ for some $\Delta_1, \Delta_2$ such that $\Gamma; \Delta_1 \vdash \mu\alpha.T :: \mathsf{pub}$ and $\Gamma; \Delta_2 \vdash \mu\alpha.T' :: \mathsf{tnt}$.
By the only applicable kinding rule (KIND REC) it follows that there exist $\Delta_1', \Delta_2'$ such that $\Gamma; \Delta_1 \hookrightarrow \Gamma; !\Delta_1'$ and $\Gamma; \Delta_2 \hookrightarrow \Gamma; !\Delta_2'$ with $\Gamma, \alpha :: \mathsf{pub}; !\Delta_1' \vdash T :: \mathsf{pub}$ and $\Gamma, \alpha :: \mathsf{tnt}; !\Delta_2' \vdash T' :: \mathsf{tnt}$.
Since $\Gamma; !\Delta_1' \hookrightarrow \Gamma; !\Delta_1'$ and $\Gamma; !\Delta_2' \hookrightarrow \Gamma; !\Delta_2'$ by Lemma B.8, we can apply (KIND REC) to derive that $\Gamma; !\Delta_1' \vdash \mu\alpha.T :: \mathsf{pub}$ and $\Gamma; !\Delta_2' \vdash \mu\alpha.T' :: \mathsf{tnt}$.
By part (4) of Lemma B.31 we then get $\Gamma; !\Delta_1', !\Delta_1' \vdash T\{\mu\alpha.T/\alpha\} :: \mathsf{pub}$ and $\Gamma; !\Delta_2'!, !\Delta_2' \vdash T'\{\mu\alpha.T'/\alpha\} :: \mathsf{tnt}$, hence we can apply (SUB PUB TNT) to get:

$$\Gamma; !\Delta_1', !\Delta_1', !\Delta_2', !\Delta_2' \vdash T\{\mu\alpha.T/\alpha\} <: T'\{\mu\alpha.T'/\alpha\}.$$

Since $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1', !\Delta_1', !\Delta_2', !\Delta_2'$ by Lemma B.8, we can conclude.

(3) We know that $\Gamma; \Delta \vdash \mathsf{fold}\ M : \mu\alpha.T$ and that $\psi(\mu\alpha.T) = \mu\alpha.T$. We apply part (1) and derive that there exist $\Delta_1, \Delta_2, T'$ such that:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$,
— $\Gamma; \Delta_1 \vdash \mathsf{fold}\ M : \mu\alpha.T'$ by a top-level application of (VAL FOLD), and
— $\Gamma; \Delta_2 \vdash \mu\alpha.T' <: \mu\alpha.T$.
By the definition of (VAL FOLD) the second statement lets us derive that:

$$\Gamma; !\Delta_1' \vdash M : T'\{\mu\alpha.T'/\alpha\},$$

for some $\Delta_1'$ such that $\Gamma; \Delta_1 \hookrightarrow \Gamma; !\Delta_1'$.
Applying part (2) to the third statement yields that there exists $\Delta_2'$ such that:
— $\Gamma; \Delta_2 \hookrightarrow \Gamma; !\Delta_2'$,
— $\Gamma; !\Delta_2' \vdash T'\{\mu\alpha.T'/\alpha\} <: T\{\mu\alpha.T/\alpha\}$.

We apply (EXP SUBSUM) to $\Gamma; !\Delta'_1 \vdash M : T'\{\mu\alpha.T'/\alpha\}$ and $\Gamma; !\Delta'_2 \vdash T'\{\mu\alpha.T'/\alpha\} <: T\{\mu\alpha.T/\alpha\}$, which leads to:

$$\Gamma; !\Delta'_1, !\Delta'_2 \vdash M : T\{\mu\alpha.T/\alpha\},$$

Since $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'_1, !\Delta'_2$ by Lemma B.8, we can conclude.

(4) We can immediately conclude by an application of statement (3) and Lemma B.9.

□

### B.6. Properties of extraction

We first present some simple, but useful properties of the extraction relation.

LEMMA B.36 (EXTRACTION AND FREE VALUES). *If $E \rightsquigarrow^{\widetilde{a}} [\Delta \mid D]$, then $fnfv(\Delta) \cup fnfv(D) \subseteq fnfv(E)$.*

PROOF. By induction on the derivation of $E \rightsquigarrow [\Delta \mid D]$. □

LEMMA B.37 (EXTENDING EXTRACTION). *If $E \rightsquigarrow^{\widetilde{b}} [\Delta \mid D]$ and $a \notin fn(E)$, then $E \rightsquigarrow^{a,\widetilde{b}} [\Delta \mid D]$.*

PROOF. By induction on the derivation of $E \rightsquigarrow^{\widetilde{b}} [\Delta \mid D]$. □

LEMMA B.38 (RESTRICTING EXTRACTION). *If $E \rightsquigarrow^{\widetilde{a}} [\Delta \mid D]$ and $E \rightsquigarrow^{\widetilde{b}} [\Delta' \mid D']$ with $\{\widetilde{b}\} \subseteq \{\widetilde{a}\}$, then $D \rightsquigarrow^{\widetilde{b}} [\Delta'' \mid D']$, where $\Delta' = \Delta, \Delta''$.*

PROOF. By induction on the structure of $E$:

*Case* $E = \mathsf{assume}\ F$ with $F \neq \mathbf{1}$ and $fn(F) \cap \{\widetilde{a}\} = \emptyset$: we have $E \rightsquigarrow^{\widetilde{a}} [F \mid \mathsf{assume}\ \mathbf{1}]$ by (EXTR ASSUME). Since $\{\widetilde{b}\} \subseteq \{\widetilde{a}\}$, we know that $fn(F) \cap \{\widetilde{b}\} = \emptyset$, hence we have $E \rightsquigarrow^{\widetilde{b}} [F \mid \mathsf{assume}\ \mathbf{1}]$ by (EXTR ASSUME). We know that $\mathsf{assume}\ \mathbf{1} \rightsquigarrow^{\widetilde{b}} [\emptyset \mid \mathsf{assume}\ \mathbf{1}]$ by (EXTR EXP), which allows us to conclude.

*Case* $E = \mathsf{assume}\ F$ with $F \neq \mathbf{1}$ and $fn(F) \cap \{\widetilde{a}\} \neq \emptyset$: we have $E \rightsquigarrow^{\widetilde{a}} [\emptyset \mid \mathsf{assume}\ F]$ by (EXTR EXP). Now we distinguish two cases: if $fn(F) \cap \{\widetilde{b}\} \neq \emptyset$, then we also have $E \rightsquigarrow^{\widetilde{b}} [\emptyset \mid \mathsf{assume}\ F]$ by (EXTR EXP), i.e., we have $\mathsf{assume}\ F \rightsquigarrow^{\widetilde{b}} [\emptyset \mid \mathsf{assume}\ F]$ and we conclude. Otherwise, whenever $fn(F) \cap \{\widetilde{b}\} = \emptyset$, we have $E \rightsquigarrow^{\widetilde{b}} [F \mid \mathsf{assume}\ \mathbf{1}]$ by (EXTR ASSUME), i.e., we have $\mathsf{assume}\ F \rightsquigarrow^{\widetilde{b}} [F \mid \mathsf{assume}\ \mathbf{1}]$ and we conclude again.

*Case* $E = E_1 \upharpoonright E_2$: we know by the definition of the only applicable extraction rule (EXTR FORK) that:

— $E \rightsquigarrow^{\widetilde{a}} [\Delta_1, \Delta_2 \mid D_1 \upharpoonright D_2]$ and
— $E \rightsquigarrow^{\widetilde{b}} [\Delta'_1, \Delta'_2 \mid D'_1 \upharpoonright D'_2]$, where
— $E_i \rightsquigarrow^{\widetilde{a}} [\Delta_i \mid D_i]$ and
— $E_i \rightsquigarrow^{\widetilde{b}} [\Delta'_i \mid D'_i]$ for $i \in \{1, 2\}$.

By applying the induction hypothesis to the latter two statements we know that there exist $\Delta''_1, \Delta''_2$ such that:

$$D_i \rightsquigarrow^{\widetilde{b}} [\Delta''_i \mid D'_i],$$

where $\Delta'_i = \Delta_i, \Delta''_i$ for $i \in \{1, 2\}$. By (EXP FORK) we can conclude that:

$$D_1 \upharpoonright D_2 \rightsquigarrow^{\widetilde{b}} [\Delta''_1, \Delta''_2 \mid D'_1 \upharpoonright D'_2],$$

where $\Delta'_1, \Delta'_2 = \Delta_1, \Delta''_1, \Delta_2, \Delta''_1 = \Delta_1, \Delta_2, \Delta''_1, \Delta''_2$.

*Case* $E$ is a restriction or let: in this case both $E \leadsto^{\widetilde{a}} [\Delta \mid D]$ and $E \leadsto^{\widetilde{b}} [\Delta' \mid D']$ must have been derived by a top-level application of the same extraction rule $\mathcal{R}$. We apply the induction hypothesis to the premise of the extraction rule $\mathcal{R}$ and conclude by applying $\mathcal{R}$ to the result, similarly to the previous case of forks.

*Case* $E$ has a different form: in this case both $E \leadsto^{\widetilde{a}} [\emptyset \mid E]$ and $E \leadsto^{\widetilde{b}} [\emptyset \mid E]$ by (EXTR EXP), so we immediately conclude.

□

LEMMA B.39 (TRANSITIVITY OF EXTRACTION). *Let $E \leadsto^{\widetilde{b}} [\Delta' \mid E']$ and $E' \leadsto^{\widetilde{c}} [\Delta'' \mid E'']$, where $\{\widetilde{c}\} \subseteq \{\widetilde{b}\}$, then $E \leadsto^{\widetilde{c}} [\Delta', \Delta'' \mid E'']$.*

PROOF. By induction on the structure of $E$:

*Case* $E = \mathsf{assume}\ F$, where $F \neq \mathbf{1}$ and $fn(F) \cap \{\widetilde{b}\} = \emptyset$. In this case we know, by definition of the only applicable extraction rule (EXTR ASSUME), that $E \leadsto^{\widetilde{b}} [\Delta' \mid E']$ with $\Delta' = F$ and $E' = \mathsf{assume}\ \mathbf{1}$. It immediately follows by the only applicable extraction rule (EXTR EXP) that $E' \leadsto^{\widetilde{c}} [\Delta'' \mid E'']$ with $\Delta'' = \emptyset$ and $E'' = \mathsf{assume}\ \mathbf{1}$. Since we know that $\{\widetilde{c}\} \subseteq \{\widetilde{b}\}$ and $fn(F) \cap \{\widetilde{b}\} = \emptyset$, we know that $fn(F) \cap \{\widetilde{c}\} = \emptyset$. We can thus apply (EXTR ASSUME) to derive $E \leadsto^{\widetilde{c}} [F \mid \mathsf{assume}\ \mathbf{1}]$ and conclude.

*Case* $E$ is a restriction, fork, or let: in this case both $E \leadsto^{\widetilde{b}} [\Delta' \mid E']$ and $E' \leadsto^{\widetilde{c}} [\Delta'' \mid E'']$ must have been derived by a top-level application of the same extraction rule $\mathcal{R}$. We apply the induction hypothesis to the premise(s) of the extraction rule $\mathcal{R}$ and conclude by applying $\mathcal{R}$ to the result(s).

*Case* $E$ has a different form: In this case we know that $E \leadsto^{\widetilde{b}} [\emptyset \mid E']$ with $E' = E$. Since we know that $E' \leadsto^{\widetilde{c}} [\Delta' \mid E'']$, it immediately follows that $E \leadsto^{\widetilde{c}} [\Delta' \mid E'']$ and we conclude.

□

LEMMA B.40 (IDEMPOTENT EXTRACTION). *If $E \leadsto^{\widetilde{a}} [\Delta \mid D]$, then $D \leadsto^{\widetilde{a}} [\emptyset \mid D]$.*

PROOF. By induction on the derivation of $E \leadsto^{\widetilde{a}} [\Delta \mid D]$. □

The next result shows that heating preserves logic: if $E \Rrightarrow E'$, then the formulas extracted from $E$ are exactly the same of the formulas extracted from $E'$. Moreover, the purged expressions $D$ and $D'$ obtained after extracting the assumptions from $E$ and $E'$ respectively are again related by heating. All this information is needed to show that heating preserves typing (Lemma B.46 below). In the following proofs we often write $E \leadsto [\Delta \mid D]$ whenever $E \leadsto^{\widetilde{a}} [\Delta \mid D]$ for some $\widetilde{a}$ clear from the context.

LEMMA B.41 (HEATING PRESERVES LOGIC). *If $E \Rrightarrow E'$ and $E \leadsto^{\widetilde{a}} [\Delta \mid D]$, then $E' \leadsto^{\widetilde{a}} [\Delta \mid D']$ for some $D'$ such that $D \Rrightarrow D'$. Moreover, the depth of the derivation of $D \Rrightarrow D'$ equals that of $E \Rrightarrow E'$.*

PROOF. By induction on the derivation of $E \Rrightarrow E'$:

*Case* (HEAT REFL): the case is trivial.

*Case* (HEAT TRANS): assume $E \Rrightarrow E''$ by the premises $E \Rrightarrow E'$ and $E' \Rrightarrow E''$. Assume further $E \leadsto [\Delta \mid D]$. We apply the induction hypothesis on $E \Rrightarrow E'$ and we get $E' \leadsto [\Delta \mid D']$ with $D \Rrightarrow D'$. We then apply the induction hypothesis on $E' \Rrightarrow E''$ and we get $E'' \leadsto [\Delta \mid D'']$ with $D' \Rrightarrow D''$. Since $D \Rrightarrow D''$ by (HEAT TRANS), we can conclude.

*Case* (HEAT LET): assume $\mathsf{let}\ x = E\ \mathsf{in}\ E'' \Rrightarrow \mathsf{let}\ x = E'\ \mathsf{in}\ E''$ by the premise $E \Rrightarrow E'$. Assume further $\mathsf{let}\ x = E\ \mathsf{in}\ E'' \rightsquigarrow [\Delta \mid \mathsf{let}\ x = D\ \mathsf{in}\ E'']$, which must be derived by the premise $E \rightsquigarrow [\Delta \mid D]$. We apply the induction hypothesis and we get $E' \rightsquigarrow [\Delta \mid D']$ with $D \Rrightarrow D'$. Hence, we have $\mathsf{let}\ x = E'\ \mathsf{in}\ E'' \rightsquigarrow [\Delta \mid \mathsf{let}\ x = D'\ \mathsf{in}\ E'']$ by (EXTR LET) and the conclusion follows by observing that $\mathsf{let}\ x = D\ \mathsf{in}\ E'' \Rrightarrow \mathsf{let}\ x = D'\ \mathsf{in}\ E''$ by (HEAT LET).

*Case* (HEAT RES): assume $(\nu a)E \Rrightarrow (\nu a)E'$ by the premise $E \Rrightarrow E'$. Assume further $(\nu a)E \rightsquigarrow^{\widetilde{b}} [\Delta \mid (\nu a)D]$, which must be derived by the premise $E \rightsquigarrow^{a,\widetilde{b}} [\Delta \mid D]$. We apply the induction hypothesis and we get $E' \rightsquigarrow^{a,\widetilde{b}} [\Delta \mid D']$ with $D \Rrightarrow D'$. Hence, we have $(\nu a)E' \rightsquigarrow^{\widetilde{b}} [\Delta \mid (\nu a)D']$ by (EXTR RES) and the conclusion follows by observing that $(\nu a)D \Rrightarrow (\nu a)D'$ by (HEAT RES).

*Case* (HEAT FORK 1): assume $E \upharpoonright E'' \Rrightarrow E' \upharpoonright E''$ by the premise $E \Rrightarrow E'$. Assume further $E \upharpoonright E'' \rightsquigarrow [\Delta, \Delta'' \mid D \upharpoonright D'']$, which must be derived by the premises $E \rightsquigarrow [\Delta \mid D]$ and $E'' \rightsquigarrow [\Delta'' \mid D'']$. By inductive hypothesis $E' \rightsquigarrow [\Delta \mid D']$ with $D \Rrightarrow D'$. Hence, we have $E' \upharpoonright E'' \rightsquigarrow [\Delta, \Delta'' \mid D' \upharpoonright D'']$ and the conclusion follows by observing that $D \upharpoonright D'' \Rrightarrow D' \upharpoonright D''$ by (HEAT FORK 1).

*Case* (HEAT FORK 2): the case is analogous to (HEAT FORK 1).

*Case* (HEAT FORK ()): assume $() \upharpoonright E \Rrightarrow E$. Let $E \rightsquigarrow [\Delta \mid D]$, we have $() \upharpoonright E \rightsquigarrow [\Delta \mid () \upharpoonright D]$. Since $() \upharpoonright D \Rrightarrow D$ by (HEAT FORK ()), we can conclude. The other direction is analogous.

*Case* (HEAT MSG ()): assume $a!M \Rrightarrow a!M \upharpoonright ()$. We have $a!M \rightsquigarrow [\emptyset \mid a!M]$ and $a!M \upharpoonright () \rightsquigarrow [\emptyset \mid a!M \upharpoonright ()]$, hence the conclusion follows by (HEAT MSG ()).

*Case* (HEAT ASSUME ()): let $\mathsf{assume}\ F \Rrightarrow \mathsf{assume}\ F \upharpoonright ()$. We have two possibilities: either $\mathsf{assume}\ F \rightsquigarrow [F \mid \mathsf{assume}\ \mathbf{1}]$ or $\mathsf{assume}\ F \rightsquigarrow [\emptyset \mid \mathsf{assume}\ F]$. In the first case we also have $\mathsf{assume}\ F \upharpoonright () \rightsquigarrow [F \mid \mathsf{assume}\ \mathbf{1} \upharpoonright ()]$, while in the second case we have $\mathsf{assume}\ F \upharpoonright () \rightsquigarrow [\emptyset \mid \mathsf{assume}\ F \upharpoonright ()]$. In both cases we can conclude by (HEAT ASSUME ()).

*Case* (HEAT ASSERT ()): let $\mathsf{assert}\ F \Rrightarrow \mathsf{assert}\ F \upharpoonright ()$. We have $\mathsf{assert}\ F \rightsquigarrow [\emptyset \mid \mathsf{assert}\ F]$ and $\mathsf{assert}\ F \upharpoonright () \rightsquigarrow [\emptyset \mid \mathsf{assert}\ F \upharpoonright ()]$, hence the conclusion follows by (HEAT ASSERT ()).

*Case* (HEAT RES FORK 1): assume $E \upharpoonright (\nu a)E' \Rrightarrow (\nu a)(E \upharpoonright E')$ with $a \notin fn(E)$. The only possible extraction derivation is the following:

$$\textsc{Extr Fork}\ \dfrac{E \rightsquigarrow^{\widetilde{b}} [\Delta \mid D] \qquad \dfrac{E' \rightsquigarrow^{a,\widetilde{b}} [\Delta' \mid D']}{(\nu a)E' \rightsquigarrow^{\widetilde{b}} [\Delta' \mid (\nu a)D']}\ \textsc{Extr Res}}{E \upharpoonright (\nu a)E' \rightsquigarrow^{\widetilde{b}} [\Delta, \Delta' \mid D \upharpoonright (\nu a)D']}$$

Since $a \notin fn(E)$, we can apply Lemma B.37 and get $E \rightsquigarrow^{a,\widetilde{b}} [\Delta \mid D]$. Hence, we can construct the following derivation:

$$\dfrac{\textsc{Extr Fork}\ \dfrac{E \rightsquigarrow^{a,\widetilde{b}} [\Delta \mid D] \qquad E' \rightsquigarrow^{a,\widetilde{b}} [\Delta' \mid D']}{E \upharpoonright E' \rightsquigarrow^{a,\widetilde{b}} [\Delta, \Delta' \mid D \upharpoonright D']}}{(\nu a)(E \upharpoonright E') \rightsquigarrow^{\widetilde{b}} [\Delta, \Delta' \mid (\nu a)(D \upharpoonright D')]}\ \textsc{Extr Res}$$

Since $a \notin fn(E)$ implies $a \notin fn(D)$ by Lemma B.36, we have $D \upharpoonright (\nu a)D' \Rrightarrow (\nu a)(D \upharpoonright D')$ by (HEAT RES FORK 1) and we conclude.

*Case* (HEAT RES FORK 2): the case is analogous to (HEAT RES FORK 1).

*Case* (HEAT RES LET): assume $\mathsf{let}\ x = (\nu a)E\ \mathsf{in}\ E' \Rrightarrow (\nu a)(\mathsf{let}\ x = E\ \mathsf{in}\ E')$ with $a \notin fn(E')$. The only possible extraction derivation is the following:

$$\text{EXTR LET}\ \dfrac{\dfrac{E \rightsquigarrow^{a,\widetilde{b}} [\Delta \mid D]}{(\nu a)E \rightsquigarrow^{\widetilde{b}} [\Delta \mid (\nu a)D]}\ \text{EXTR RES}}{\mathsf{let}\ x = (\nu a)E\ \mathsf{in}\ E' \rightsquigarrow^{\widetilde{b}} [\Delta \mid \mathsf{let}\ x = (\nu a)D\ \mathsf{in}\ E']}$$

Hence, we can construct the following derivation:

$$\dfrac{\text{EXTR LET}\ \dfrac{E \rightsquigarrow^{a,\widetilde{b}} [\Delta \mid D]}{\mathsf{let}\ x = E\ \mathsf{in}\ E' \rightsquigarrow^{a,\widetilde{b}} [\Delta \mid \mathsf{let}\ x = D\ \mathsf{in}\ E']}}{(\nu a)(\mathsf{let}\ x = E\ \mathsf{in}\ E') \rightsquigarrow^{\widetilde{b}} [\Delta \mid (\nu a)(\mathsf{let}\ x = D\ \mathsf{in}\ E')]}\ \text{EXTR RES}$$

Since $a \notin fn(E')$ implies $a \notin fn(D)$ by Lemma B.36, we have $\mathsf{let}\ x = (\nu a)D\ \mathsf{in}\ E' \Rrightarrow (\nu a)(\mathsf{let}\ x = D\ \mathsf{in}\ E')$ by (HEAT RES LET) and we conclude.

*Case* (HEAT FORK ASSOC): assume $(E \upharpoonright E') \upharpoonright E'' \Rrightarrow E \upharpoonright (E' \upharpoonright E'')$. The only possible extraction derivation is the following:

$$\text{EXP FORK}\ \dfrac{\text{EXP FORK}\ \dfrac{E \rightsquigarrow [\Delta \mid D] \qquad E' \rightsquigarrow [\Delta' \mid D']}{E \upharpoonright E' \rightsquigarrow [\Delta, \Delta' \mid D \upharpoonright D']} \qquad E'' \rightsquigarrow [\Delta'' \mid D'']}{(E \upharpoonright E') \upharpoonright E'' \rightsquigarrow [\Delta, \Delta', \Delta'' \mid (D \upharpoonright D') \upharpoonright D'']}$$

Hence, we can construct the following derivation:

$$\text{EXP FORK}\ \dfrac{E \rightsquigarrow [\Delta \mid D] \qquad \dfrac{E' \rightsquigarrow [\Delta' \mid D'] \qquad E'' \rightsquigarrow [\Delta'' \mid D'']}{E' \upharpoonright E'' \rightsquigarrow [\Delta', \Delta'' \mid D' \upharpoonright D'']}\ \text{EXP FORK}}{E \upharpoonright (E' \upharpoonright E'') \rightsquigarrow [\Delta, \Delta', \Delta'' \mid D \upharpoonright (D' \upharpoonright D'')]}$$

We observe that $(D \upharpoonright D') \upharpoonright D'' \Rrightarrow D \upharpoonright (D' \upharpoonright D'')$ by (HEAT FORK ASSOC) to conclude. The other direction is analogous.

*Case* (HEAT FORK COMM): assume $(E \upharpoonright E') \upharpoonright E'' \Rrightarrow (E' \upharpoonright E) \upharpoonright E''$. The only possible extraction derivation is the following:

$$\text{EXP FORK}\ \dfrac{\text{EXP FORK}\ \dfrac{E \rightsquigarrow [\Delta \mid D] \qquad E' \rightsquigarrow [\Delta' \mid D']}{E \upharpoonright E' \rightsquigarrow [\Delta, \Delta' \mid D \upharpoonright D']} \qquad E'' \rightsquigarrow [\Delta'' \mid D'']}{(E \upharpoonright E') \upharpoonright E'' \rightsquigarrow [\Delta, \Delta', \Delta'' \mid (D \upharpoonright D') \upharpoonright D'']}$$

Hence, we can construct the following derivation:

$$\text{EXP FORK}\ \dfrac{\text{EXP FORK}\ \dfrac{E' \rightsquigarrow [\Delta' \mid D'] \qquad E \rightsquigarrow [\Delta \mid D]}{E' \upharpoonright E \rightsquigarrow [\Delta, \Delta' \mid D' \upharpoonright D]} \qquad E'' \rightsquigarrow [\Delta'' \mid D'']}{(E' \upharpoonright E) \upharpoonright E'' \rightsquigarrow [\Delta, \Delta', \Delta'' \mid (D' \upharpoonright D) \upharpoonright D'']}$$

where we note that the order of the formulas is immaterial, since we interpret the $\Delta$'s as multisets. We observe that $(D \upharpoonright D') \upharpoonright D'' \Rrightarrow (D' \upharpoonright D) \upharpoonright D''$ by (HEAT FORK COMM) to conclude. The other direction is analogous.

*Case* (HEAT FORK LET): assume $\mathsf{let}\ x = (E_1 \upharpoonright E_2)\ \mathsf{in}\ E_3 \Rrightarrow E_1 \upharpoonright (\mathsf{let}\ x = E_2\ \mathsf{in}\ E_3)$. We have $\mathsf{let}\ x = (E_1 \upharpoonright E_2)\ \mathsf{in}\ E_3 \rightsquigarrow [\Delta_1, \Delta_2 \mid \mathsf{let}\ x = (D_1 \upharpoonright D_2)\ \mathsf{in}\ E_3]$ with $E_1 \rightsquigarrow [\Delta_1 \mid D_1]$ and $E_2 \rightsquigarrow [\Delta_2 \mid D_2]$. In fact, the only possible extraction derivation is the following:

$$\text{EXTR LET}\ \dfrac{\text{EXTR FORK}\ \dfrac{E_1 \rightsquigarrow [\Delta_1 \mid D_1] \qquad E_2 \rightsquigarrow [\Delta_2 \mid D_2]}{E_1 \upharpoonright E_2 \rightsquigarrow [\Delta_1, \Delta_2 \mid D_1 \upharpoonright D_2]}}{\mathsf{let}\ x = (E_1 \upharpoonright E_2)\ \mathsf{in}\ E_3 \rightsquigarrow [\Delta_1, \Delta_2 \mid \mathsf{let}\ x = (D_1 \upharpoonright D_2)\ \mathsf{in}\ E_3]}$$

Hence, we can construct the following derivation:

$$\textsc{Extr Fork}\ \dfrac{E_1 \rightsquigarrow [\Delta_1 \mid D_1] \qquad \dfrac{E_2 \rightsquigarrow [\Delta_2 \mid D_2]}{\mathsf{let}\ x = E_2\ \mathsf{in}\ E_3 \rightsquigarrow [\Delta_2 \mid \mathsf{let}\ x = D_2\ \mathsf{in}\ E_3]}\ \textsc{Extr Let}}{E_1 \upharpoonright (\mathsf{let}\ x = E_2\ \mathsf{in}\ E_3) \rightsquigarrow [\Delta_1, \Delta_2 \mid D_1 \upharpoonright (\mathsf{let}\ x = D_2\ \mathsf{in}\ E_3)]}$$

Since $\mathsf{let}\ x = (D_1 \upharpoonright D_2)\ \mathsf{in}\ E_3 \Rrightarrow D_1 \upharpoonright (\mathsf{let}\ x = D_2\ \mathsf{in}\ E_3)$ by (HEAT FORK LET), we can conclude. The other direction is analogous, since we can invert the construction and transform the second derivation, which is the only possible one, into the first one.

□

The next lemma is in the same spirit of Lemma B.41, but it predicates over the reduction relation rather than on heating and it is slightly more complicated. This is needed in the proof of Subject Reduction (Theorem B.48 below).

LEMMA B.42 (REDUCTION PRESERVES LOGIC). *If $E \to E'$ and $E \rightsquigarrow^{\widetilde{a}} [\Delta \mid D]$, then $D \to D'$ and $E' \rightsquigarrow^{\widetilde{a}} [\Delta, \Delta' \mid D'']$ for some $D', D'', \Delta'$ such that $D' \rightsquigarrow^{\widetilde{a}} [\Delta' \mid D^*]$ with $D^* \Rrightarrow D''$. Moreover, the depth of the derivation of $D \to D'$ equals that of $E \to E'$.*

PROOF. By induction on the derivation of $E \to E'$. We note that, whenever $E \rightsquigarrow [\emptyset \mid E]$, the conclusion is trivial, hence we focus on the remaining cases:

*Case* (RED LET): assume $\mathsf{let}\ x = E_1\ \mathsf{in}\ E_2 \to \mathsf{let}\ x = E_1'\ \mathsf{in}\ E_2$ with $E_1 \to E_1'$ and $\mathsf{let}\ x = E_1\ \mathsf{in}\ E_2 \rightsquigarrow [\Delta_1 \mid \mathsf{let}\ x = D_1\ \mathsf{in}\ E_2]$ with $E_1 \rightsquigarrow [\Delta_1 \mid D_1]$. By induction hypothesis $D_1 \to D_1'$ and $E_1' \rightsquigarrow [\Delta_1, \Delta_1' \mid D']$ with $D_1' \rightsquigarrow [\Delta_1' \mid D_1'']$ and $D_1'' \Rrightarrow D'$. We then have $\mathsf{let}\ x = D_1\ \mathsf{in}\ E_2 \to \mathsf{let}\ x = D_1'\ \mathsf{in}\ E_2$ by (RED LET). Now we observe that $\mathsf{let}\ x = E_1'\ \mathsf{in}\ E_2 \rightsquigarrow [\Delta_1, \Delta_1' \mid \mathsf{let}\ x = D'\ \mathsf{in}\ E_2]$ and $\mathsf{let}\ x = D_1'\ \mathsf{in}\ E_2 \rightsquigarrow [\Delta_1' \mid \mathsf{let}\ x = D_1''\ \mathsf{in}\ E_2]$, so we conclude by (HEAT LET).

*Case* (RED RES): assume $(\nu a)E \to (\nu a)E'$ with $E \to E'$ and $(\nu a)E \rightsquigarrow^{\widetilde{b}} [\Delta_1 \mid (\nu a)D_1]$ with $E \rightsquigarrow^{a,\widetilde{b}} [\Delta_1 \mid D_1]$. By induction hypothesis $D_1 \to D_1'$ and $E' \rightsquigarrow^{a,\widetilde{b}} [\Delta_1, \Delta_1' \mid D']$ with $D_1' \rightsquigarrow^{a,\widetilde{b}} [\Delta_1' \mid D_1'']$ and $D_1'' \Rrightarrow D'$. We then have $(\nu a)D_1 \to (\nu a)D_1'$ by (RED RES). Now we observe that $(\nu a)E' \rightsquigarrow^{\widetilde{b}} [\Delta_1, \Delta_1' \mid (\nu a)D']$ and $(\nu a)D_1' \rightsquigarrow^{\widetilde{b}} [\Delta_1' \mid (\nu a)D_1'']$, so we conclude by (HEAT RES).

*Case* (RED FORK 1): assume $E_1 \upharpoonright E_2 \to E_1' \upharpoonright E_2$ with $E_1 \to E_1'$ and $E_1 \upharpoonright E_2 \rightsquigarrow [\Delta_1, \Delta_2 \mid D_1 \upharpoonright D_2]$ with $E_1 \rightsquigarrow [\Delta_1 \mid D_1]$, $E_2 \rightsquigarrow [\Delta_2 \mid D_2]$. By induction hypothesis $D_1 \to D_1'$ and $E_1' \rightsquigarrow [\Delta_1, \Delta_1' \mid D']$ with $D_1' \rightsquigarrow [\Delta_1' \mid D_1'']$ and $D_1'' \Rrightarrow D'$. We then have $D_1 \upharpoonright D_2 \to D_1' \upharpoonright D_2$ by (RED FORK 1). Now we observe that $E_1' \upharpoonright E_2 \rightsquigarrow [\Delta_1, \Delta_1', \Delta_2 \mid D' \upharpoonright D_2]$ and $D_1' \upharpoonright D_2 \rightsquigarrow [\Delta_1' \mid D_1'' \upharpoonright D_2]$, since $D_2 \rightsquigarrow [\emptyset \mid D_2]$ by Lemma B.40. Thus, we conclude by (HEAT FORK 1).

*Case* (RED FORK 2): analogous to the previous case.

*Case* (RED HEAT): assume $E \to E'$ by the premises $E \Rrightarrow E_A$, $E_A \to E_B$, $E_B \Rrightarrow E'$. Assume further $E \rightsquigarrow [\Delta_1 \mid E_1]$. By Lemma B.41 we have $E_A \rightsquigarrow [\Delta_1 \mid E_A']$ with $E_1 \Rrightarrow E_A'$. By inductive hypothesis we get $E_A' \to E_B'$ and $E_B \rightsquigarrow [\Delta_1, \Delta_1' \mid D_B]$ with $E_B' \rightsquigarrow [\Delta_1' \mid E_B'']$ and $E_B'' \Rrightarrow D_B$. Again by Lemma B.41 we have $E' \rightsquigarrow [\Delta_1, \Delta_1' \mid E'']$ with $D_B \Rrightarrow E''$. Since we can derive $E_1 \to E_B'$ by (RED HEAT) and $E_B'' \Rrightarrow E''$ by (HEAT TRANS), we can conclude.

□

### B.7. Proof of subject reduction

In the proof of Lemmas B.44, B.45, B.46 and Theorem B.48 below we rely on an observation about the structure of the type derivations to simplify the formal reasoning and carry out the proofs. First, we consider an alternative formulation of typing for values, presented in Table XXV, which removes the non-structural rule (VAL REFINE). We also assume to keep the original typing rules for expressions.

**Table XXV** Alternative rules for typing values

$$\text{VAL VAR REFINE}\ \frac{(x:T)\in\Gamma \qquad \Gamma;\Delta\vdash F\{x/y\}}{\Gamma;\Delta\vdash^{\mathsf{alt}} x:\{y:T\mid F\}}$$

$$\text{VAL UNIT REFINE}\ \frac{\Gamma;\Delta\vdash F\{()/y\}}{\Gamma;\Delta\vdash^{\mathsf{alt}} ():\{y:\mathsf{unit}\mid F\}}$$

$$\text{VAL FUN REFINE}\ \frac{(\Gamma;!\Delta_1)\bullet x:T\vdash^{\mathsf{alt}} E:U \quad \Gamma;\Delta_2\vdash F\{\lambda x.\,E/y\} \quad \Gamma;\Delta\hookrightarrow\Gamma;!\Delta_1,\Delta_2}{\Gamma;\Delta\vdash^{\mathsf{alt}}\lambda x.\,E:\{y:x:T\to U\mid F\}}$$

$$\text{VAL PAIR REFINE}\ \frac{\Gamma;!\Delta_1\vdash^{\mathsf{alt}} M:T \quad \Gamma;!\Delta_2\vdash^{\mathsf{alt}} N:U\{M/x\} \quad \Gamma;\Delta_3\vdash F\{(M,N)/y\} \quad \Gamma;\Delta\hookrightarrow\Gamma;!\Delta_1,!\Delta_2,\Delta_3}{\Gamma;\Delta\vdash^{\mathsf{alt}}(M,N):\{y:x:T*U\mid F\}}$$

$$\text{VAL INL REFINE}\ \frac{\Gamma;!\Delta_1\vdash^{\mathsf{alt}} M:T \quad \Gamma;!\Delta_1\vdash U \qquad \Gamma;\Delta_2\vdash F\{\mathsf{inl}\ M/y\} \quad \Gamma;\Delta\hookrightarrow\Gamma;!\Delta_1,\Delta_2}{\Gamma;\Delta\vdash^{\mathsf{alt}}\mathsf{inl}\ M:\{y:T+U\mid F\}}$$

$$\text{VAL INR REFINE}\ \frac{\Gamma;!\Delta_1\vdash^{\mathsf{alt}} M:U \quad \Gamma;!\Delta_1\vdash T \qquad \Gamma;\Delta_2\vdash F\{\mathsf{inr}\ M/y\} \quad \Gamma;\Delta\hookrightarrow\Gamma;!\Delta_1,\Delta_2}{\Gamma;\Delta\vdash^{\mathsf{alt}}\mathsf{inr}\ M:\{y:T+U\mid F\}}$$

$$\text{VAL FOLD REFINE}\ \frac{\Gamma;!\Delta_1\vdash^{\mathsf{alt}} M:T\{\mu a.\,T/\alpha\} \quad \Gamma;\Delta_2\vdash F\{\mathsf{fold}\ M/y\} \quad \Gamma;\Delta\hookrightarrow\Gamma;!\Delta_1,\Delta_2}{\Gamma;\Delta\vdash^{\mathsf{alt}}\mathsf{fold}\ M:\{y:\mu\alpha.\,T\mid F\}}$$

We can show that the original and the alternative formulation coincide.

LEMMA B.43 (ALTERNATIVE TYPING). *$\Gamma;\Delta\vdash E:T$ if and only if $\Gamma;\Delta\vdash^{\mathsf{alt}} E:T$.*

PROOF. We show both directions independently:

($\Rightarrow$) By induction on the derivation of $\Gamma;\Delta\vdash E:T$:

*Case* (VAL VAR): let $\Gamma;\Delta\vdash x:T$ by the premises $\Gamma;\Delta\vdash\diamond$ and $(x:T)\in\Gamma$. We can construct the following type derivation:

$$\text{EXP SUBSUM}\ \frac{\text{VAL VAR REFINE}\ \dfrac{(x:T)\in\Gamma \qquad \Gamma;\Delta\vdash\mathbf{1}}{\Gamma;\Delta\vdash^{\mathsf{alt}} x:\{y:T\mid\mathbf{1}\}} \qquad \text{SUB REFINE}\ \dfrac{\Gamma;\emptyset\vdash\psi(T)<:\psi(T) \qquad \Gamma,y:\psi(T);\mathbf{1}\vdash\mathbf{1}}{\Gamma;\emptyset\vdash\{y:T\mid\mathbf{1}\}<:T}}{\Gamma;\Delta\vdash^{\mathsf{alt}} x:T}$$

*Case* (VAL REFINE): let $\Gamma;\Delta\vdash M:\{x:T\mid F\}$ by the premises $\Gamma;\Delta_1\vdash M:T$ and $\Gamma;\Delta_2\vdash F\{M/x\}$. By inductive hypothesis $\Gamma;\Delta_1\vdash^{\mathsf{alt}} M:T$. By inspection of the alternative typing rules, this judgement can be derived only though an application of a structural rule after an arbitrary number of applications of (EXP SUBSUM), hence in the type derivation there must be an instance of one of the alternative type rules $\mathcal{R}$ of the form:

$$\mathcal{R}\ \frac{(\ldots) \qquad \Gamma;\Delta^*\vdash F'\{M/x\} \qquad \Gamma;\Delta'\hookrightarrow\Gamma;(\ldots),\Delta^*}{\Gamma;\Delta'\vdash^{\mathsf{alt}} M:\{x:U\mid F'\}}$$

where $\Gamma;\Delta_1\hookrightarrow\Gamma;\Delta',\Delta''$ and $\Gamma;\Delta''\vdash\{x:U\mid F'\}<:T$. (Notice that in this process we appeal to the transitivity of both the subtyping relation, proved in Lemma B.22, and the environment rewriting relation, proved in Lemma B.8.) Since $\Gamma;\Delta^*\vdash F'\{M/x\}$ and $\Gamma;\Delta_2\vdash F\{M/x\}$, we know that $\Gamma;\Delta^*,\Delta_2\vdash(F'\otimes F)\{M/x\}$ by ($\otimes$-RIGHT), so we have:

$$\mathcal{R}\ \frac{(\ldots) \qquad \Gamma;\Delta^*,\Delta_2\vdash(F'\otimes F)\{M/x\} \qquad \Gamma;\Delta',\Delta_2\hookrightarrow\Gamma;(\ldots),\Delta^*,\Delta_2}{\Gamma;\Delta',\Delta_2\vdash^{\mathsf{alt}} M:\{x:U\mid F'\otimes F\}}$$

Now we note that $\Gamma; \Delta'' \vdash \{x : U \mid F'\} <: T$ implies $\Gamma; \Delta'' \vdash \psi(U) <: \psi(T)$ by Lemma B.15 in combination with Lemma B.9. Hence, we also have:

$$\text{SUB REFINE}\ \frac{\Gamma; \Delta'' \vdash \psi(U) <: \psi(T) \qquad \Gamma, x : \psi(U); F' \otimes F \vdash F}{\Gamma; \Delta'' \vdash \{x : U \mid F' \otimes F\} <: \{x : T \mid F\}}$$

Hence, $\Gamma; \Delta', \Delta_2, \Delta'' \vdash^{\mathsf{alt}} M : \{x : T \mid F\}$ by (EXP SUBSUM). Since we have $\Gamma; \Delta \hookrightarrow \Gamma; \Delta', \Delta'', \Delta_2$ by Lemma B.8, we conclude $\Gamma; \Delta \vdash^{\mathsf{alt}} M : \{x : T \mid F\}$ by a variant of Lemma B.9 predicating over the alternative typing relation.

For all the other rules for values, the proof strategy is similar to the case of (VAL VAR). The cases for expressions which are not values are immediate, since the two formulations share the same rules.

($\Leftarrow$) By induction on the derivation of $\Gamma; \Delta \vdash^{\mathsf{alt}} E : T$:

*Case* (VAL VAR REFINE): let $\Gamma; \Delta \vdash^{\mathsf{alt}} x : \{y : T \mid F\}$ by the premises $(x : T) \in \Gamma$ and $\Gamma; \Delta \vdash F\{x/y\}$. The latter implies $\Gamma; \Delta \vdash \diamond$ by Lemma B.5, hence $\Gamma; \emptyset \vdash \diamond$ again by Lemma B.5 and we can conclude as follows:

$$\text{VAL REFINE}\ \frac{\text{VAL VAR}\ \dfrac{\Gamma; \emptyset \vdash \diamond \qquad (x : T) \in \Gamma}{\Gamma; \emptyset \vdash x : T} \qquad \Gamma; \Delta \vdash F\{x/y\}}{\Gamma; \Delta \vdash x : \{y : T \mid F\}}$$

*Case* (VAL FUN REFINE): let $\Gamma; \Delta \vdash^{\mathsf{alt}} \lambda x. E : \{y : x : T \to U \mid F\}$ by the premises $(\Gamma; !\Delta_1) \bullet x : T \vdash^{\mathsf{alt}} E : U$ and $\Gamma; \Delta_2 \vdash F\{\lambda x. E/y\}$ with $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, \Delta_2$. By inductive hypothesis $(\Gamma; !\Delta_1) \bullet x : T \vdash E : U$, hence we can conclude as follows:

$$\text{VAL REFINE}\ \frac{\text{VAL FUN}\ \dfrac{(\Gamma; !\Delta_1) \bullet x : T \vdash E : U \qquad \Gamma; !\Delta_1 \hookrightarrow \Gamma; !\Delta_1}{\Gamma; !\Delta_1 \vdash \lambda x. E : x : T \to U} \qquad \Gamma; \Delta_2 \vdash F\{\lambda x. E/y\}}{\Gamma; \Delta \vdash \lambda x. E : \{y : x : T \to U \mid F\}}$$

The case for (VAL UNIT REFINE) is similar to the case for (VAL VAR REFINE). For all the other rules for values, the proof strategy is similar to the case of (VAL FUN REFINE). The cases for expressions which are not values are immediate, since the two formulations share the same rules.

□

Now the idea is to appeal to the transitivity of both the subtyping relation (Lemma B.22) and the environment rewriting relation (Lemma B.8) to rearrange the structure of any type derivation constructed under the alternative typing rules. Namely, we observe that for any expression $E$ the general form of such a type derivation is as follows:

$$\frac{\dfrac{\dfrac{\Gamma; \Delta_1 \vdash^{\mathsf{alt}} E : T_1 \qquad \Gamma; \Delta_2 \vdash T_1 <: T_2 \qquad \Gamma; \Delta_3 \hookrightarrow \Gamma; \Delta_1, \Delta_2}{\vdots}}{\Gamma; \Delta_{2n-1} \vdash^{\mathsf{alt}} E : T_{2n-1}} \qquad \Gamma; \Delta_{2n} \vdash T_{2n-1} <: T \qquad \Gamma; \Delta \hookrightarrow \Gamma; \Delta_{2n-1}, \Delta_{2n}}{\Gamma; \Delta \vdash^{\mathsf{alt}} E : T}$$

where the last rule applied to derive $\Gamma; \Delta_1 \vdash^{\mathsf{alt}} E : T_1$ is not (EXP SUBSUM). Without loss of generality, we reorganize the derivation as follows:

$$\frac{\Gamma; \Delta_1 \vdash^{\mathsf{alt}} E : T_1 \qquad \Gamma; \Delta^* \vdash T_1 <: T \qquad \Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta^*}{\Gamma; \Delta \vdash^{\mathsf{alt}} E : T}$$

with $\Delta^* = \Delta_2, \Delta_4, \ldots, \Delta_{2n}$. Notice that also derivations which do not use rule (EXP SUBSUM) can be rearranged as detailed, since the subtyping relation is reflexive. Moreover, given that original typing and alternative typing coincide by Lemma B.43, we note that the previous transformation can be applied to any type derivation.

Now we can show that extraction preserves typing: this is needed to show that heating preserves typing (Lemma B.46 below).

LEMMA B.44 (EXTRACTION PRESERVES TYPING). *If* $\Gamma; \Delta \vdash E : T$ *and* $E \rightsquigarrow^{\widetilde{a}} [\Delta' \mid E']$, *then* $\Gamma; \Delta, \Delta' \vdash E' : T$.

PROOF. By a case analysis on the structure of $E$:

*Case* $E$ is any expression such that $E \rightsquigarrow [\emptyset \mid E]$: the conclusion is trivial.

*Case* $E = \mathsf{assume}\ F$ with $F \neq \mathbf{1}$ and $fn(F) \cap \{\widetilde{a}\} = \emptyset$: we have $E \rightsquigarrow^{\widetilde{a}} [F \mid \mathsf{assume}\ \mathbf{1}]$ and $\Gamma; \Delta \vdash \mathsf{assume}\ F : T$. The typing judgement must follow by an instance of (EXP ASSUME) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash \mathsf{assume}\ F : U$ and $\Gamma; \Delta_B \vdash U <: T$ with $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$ and $\Gamma; \Delta_A, F \vdash \mathsf{assume}\ \mathbf{1} : U$. The conclusion $\Gamma; \Delta, F \vdash \mathsf{assume}\ \mathbf{1} : T$ follows by (EXP SUBSUM).

*Case* $E = (\nu a)D$: we have $E \rightsquigarrow^{\widetilde{b}} [\Delta' \mid (\nu a)D']$ with $D \rightsquigarrow^{a,\widetilde{b}} [\Delta' \mid D']$ and $\Gamma; \Delta \vdash (\nu a)D : T$. The typing judgement must follow by an instance of (EXP RES) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash (\nu a)D : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $D \rightsquigarrow^{a} [\Delta'' \mid D'']$
— $\Gamma, a \updownarrow T'; \Delta_A, \Delta'' \vdash D'' : V$
By Lemma B.38 we know that $D' \rightsquigarrow^{a} [\Delta''' \mid D'']$, for some $\Delta'''$ such that $\Delta'' = \Delta', \Delta'''$. We can then construct the following type derivation:

$$\dfrac{\textsc{Exp Res}\ \dfrac{D' \rightsquigarrow^{a} [\Delta''' \mid D''] \qquad \Gamma, a \updownarrow T'; \Delta_A, \overbrace{\Delta', \Delta'''}^{\Delta''} \vdash D'' : V}{\Gamma; \Delta_A, \Delta' \vdash (\nu a)D' : V} \qquad \Gamma; \Delta_B \vdash V <: T}{\Gamma; \Delta, \Delta' \vdash (\nu a)D' : T}\ \textsc{Exp Subsum}$$

*Case* $E = \mathsf{let}\ x = E_1\ \mathsf{in}\ E_2$: we have $E \rightsquigarrow^{\widetilde{a}} [\Delta' \mid \mathsf{let}\ x = D'\ \mathsf{in}\ E_2]$ with $E_1 \rightsquigarrow^{\widetilde{a}} [\Delta' \mid D']$ and $\Gamma; \Delta \vdash \mathsf{let}\ x = E_1\ \mathsf{in}\ E_2 : T$. The typing judgement must follow by an instance of (EXP LET) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash \mathsf{let}\ x = E_1\ \mathsf{in}\ E_2 : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $E_1 \rightsquigarrow^{\emptyset} [\Delta'' \mid D'']$
— $\Gamma; \Delta_A, \Delta'' \hookrightarrow \Gamma; \Delta_1, \Delta_2$
— $\Gamma; \Delta_1 \vdash D'' : U$
— $(\Gamma; \Delta_2) \bullet x : U \vdash E_2 : V$
By Lemma B.38 we know that $D' \rightsquigarrow^{\emptyset} [\Delta''' \mid D'']$, for some $\Delta'''$ such that $\Delta'' = \Delta', \Delta'''$. Hence, we have $\Gamma; \Delta_A, \Delta', \Delta''' \hookrightarrow \Gamma; \Delta_1, \Delta_2$ and we can then construct the following type derivation:

$$\dfrac{\textsc{Exp Let}\ \dfrac{D' \rightsquigarrow^{\emptyset} [\Delta''' \mid D''] \qquad \Gamma; \Delta_1 \vdash D'' : U \qquad (\Gamma; \Delta_2) \bullet x : U \vdash E_2 : V}{\Gamma; \Delta_A, \Delta' \vdash \mathsf{let}\ x = D'\ \mathsf{in}\ E_2 : V} \qquad \Gamma; \Delta_B \vdash V <: T}{\Gamma; \Delta, \Delta' \vdash \mathsf{let}\ x = D'\ \mathsf{in}\ E_2 : T}\ \textsc{Exp Subsum}$$

*Case* $E = E_1 \upharpoonright E_2$: similar to the previous case.

□

Similarly to the previous result, we can also show that *inverting* an extraction preserves typing: again, this is needed to prove that heating preserves typing (Lemma B.46 below).

LEMMA B.45 (INVERTING EXTRACTION PRESERVES TYPING). *Let* $E \rightsquigarrow^{\widetilde{b}} [\Delta' \mid E']$. *If* $\Gamma; \Delta, \Delta' \vdash E' : T$, *then* $\Gamma; \Delta \vdash E : T$.

PROOF. By a case analysis on the structure of $E$:

*Case* $E$ is any expression such that $E \rightsquigarrow [\emptyset \mid E]$: the conclusion is trivial.

*Case* $E = \mathsf{assume}\ F$ with $F \neq \mathbf{1}$ and $fn(F) \cap \{\widetilde{b}\} = \emptyset$: we know that $E \rightsquigarrow^{\widetilde{b}} [F \mid \mathsf{assume}\ \mathbf{1}]$ and $\Gamma; \Delta, F \vdash \mathsf{assume}\ \mathbf{1} : T$. The conclusion $\Gamma; \Delta \vdash \mathsf{assume}\ F : T$ immediately follows by (EXP ASSUME).

*Case* $E = (\nu a)D$: we have $E \rightsquigarrow^{\widetilde{b}} [\Delta' \mid (\nu a)D']$ with $D \rightsquigarrow^{a,\widetilde{b}} [\Delta' \mid D']$ and $\Gamma; \Delta, \Delta' \vdash (\nu a)D' : T$. The typing judgement must follow by an instance of (EXP RES) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash (\nu a)D' : V$ by a top-level application of (EXP RES) and $\Gamma; \Delta_B \vdash V <: T$ with $\Gamma; \Delta, \Delta' \hookrightarrow \Gamma; \Delta_A, \Delta_B$.
Since we know that $\Gamma; \Delta_A \vdash (\nu a)D' : V$ by (EXP RES), it must be the case that $D' \rightsquigarrow^{a} [\Delta'' \mid D'']$ and $\Gamma, a \updownarrow W; \Delta_A, \Delta'' \vdash D'' : V$ with $a \notin fn(V)$.
By Lemma B.39 we know that $D \rightsquigarrow^{a,\widetilde{b}} [\Delta' \mid D']$ and $D' \rightsquigarrow^{a} [\Delta'' \mid D'']$ imply:

$$D \rightsquigarrow^{a} [\Delta', \Delta'' \mid D''].$$

By Lemma B.7 we know that $\Gamma; \Delta_B \vdash V <: T$ implies:

$$\Gamma, a \updownarrow W; \Delta_B \vdash V <: T.$$

Applying (EXP SUBSUM) to the latter and $\Gamma, a \updownarrow W; \Delta_A, \Delta'' \vdash D'' : V$, we get:

$$\Gamma, a \updownarrow W; \Delta_A, \Delta'', \Delta_B \vdash D'' : T.$$

We observe that $\Gamma, a \updownarrow W; \Delta, \Delta', \Delta'' \hookrightarrow \Gamma, a \updownarrow W; \Delta_A, \Delta'', \Delta_B$, so we can apply Lemma B.9 and get:

$$\Gamma, a \updownarrow W; \Delta, \Delta', \Delta'' \vdash D'' : T.$$

Finally, we note that $a \notin fn(T)$ by applying Lemma B.5 to $\Gamma; \Delta_B \vdash V <: T$, hence we conclude $\Gamma; \Delta \vdash (\nu a)D : T$ by an application of (EXP RES).

*Case* $E = \mathsf{let}\ x = E_1\ \mathsf{in}\ E_2$: We know that $E \rightsquigarrow^{\widetilde{b}} [\Delta' \mid \mathsf{let}\ x = D_1\ \mathsf{in}\ E_2]$, where $E_1 \rightsquigarrow^{\widetilde{b}} [\Delta' \mid D_1]$ and $\Gamma; \Delta, \Delta' \vdash \mathsf{let}\ x = D_1\ \mathsf{in}\ E_2 : T$.
The typing judgement must follow by an instance of (EXP LET) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash \mathsf{let}\ x = D_1\ \mathsf{in}\ E_2 : V$ by a top-level application of (EXP LET) and $\Gamma; \Delta_B \vdash V <: T$ with $\Gamma; \Delta, \Delta' \hookrightarrow \Gamma; \Delta_A, \Delta_B$.
Since we know that $\Gamma; \Delta_A \vdash \mathsf{let}\ x = D_1\ \mathsf{in}\ E_2 : V$ by (EXP LET), it must be the case that $D_1 \rightsquigarrow^{\emptyset} [\Delta'' \mid D_1']$ and $\Gamma; \Delta_1 \vdash D_1' : W$ and $(\Gamma; \Delta_2) \bullet x : W \vdash E_2 : V$, for some $\Delta_1, \Delta_2$ such that $\Gamma; \Delta_A, \Delta'' \hookrightarrow \Gamma; \Delta_1, \Delta_2$.
By Lemma B.39 we know that $E_1 \rightsquigarrow^{\widetilde{b}} [\Delta' \mid D_1]$ and $D_1 \rightsquigarrow^{\emptyset} [\Delta'' \mid D_1']$ imply:

$$E_1 \rightsquigarrow^{\emptyset} [\Delta', \Delta'' \mid D_1'].$$

By Lemma B.7 we know that $\Gamma; \Delta_B \vdash V <: T$ implies:

$$\Gamma, x : \psi(W); \Delta_B \vdash V <: T.$$

Applying (EXP SUBSUM) to the latter and $(\Gamma; \Delta_2) \bullet x : W \vdash E_2 : V$, we get:

$$(\Gamma; \Delta_2, \Delta_B) \bullet x : W \vdash E_2 : T.$$

We conclude $\Gamma; \Delta \vdash \mathsf{let}\ x = E_1\ \mathsf{in}\ E_2 : T$ by applying (EXP LET) to the collected statements:
— $E_1 \rightsquigarrow^{\emptyset} [\Delta', \Delta'' \mid D_1']$
— $\Gamma; \Delta_1 \vdash D_1' : W$

— $(\Gamma; \Delta_2, \Delta_B) \bullet x : W \vdash E_2 : T$, and
— $\Gamma; \Delta, \Delta', \Delta'' \hookrightarrow \Gamma; \Delta_1, (\Delta_2, \Delta_B)$, which holds by Lemma B.8.
*Case* $E = E_1 \upharpoonright E_2$: similar to the previous case.

□

The next result, sometimes called Subject Heating, shows that typing is preserved by heating. This is needed in the proof of the Subject Reduction theorem, since the reduction relation is closed under heating.

LEMMA B.46 (HEATING PRESERVES TYPING). *If* $\Gamma; \Delta \vdash E : T$ *and* $E \Rrightarrow E'$*, then* $\Gamma; \Delta \vdash E' : T$.

PROOF. By induction on the derivation of $E \Rrightarrow E'$:

*Case* (HEAT REFL): the case is trivial.
*Case* (HEAT TRANS): assume $E \Rrightarrow E''$ by the premises $E \Rrightarrow E'$ and $E' \Rrightarrow E''$. Assume further that $\Gamma; \Delta \vdash E : T$. We apply the inductive hypothesis twice and we conclude $\Gamma; \Delta \vdash E'' : T$.
*Case* (HEAT LET): assume $\mathsf{let}\ x = E\ \mathsf{in}\ E'' \Rrightarrow \mathsf{let}\ x = E'\ \mathsf{in}\ E''$ by the premise $E \Rrightarrow E'$. Assume further that $\Gamma; \Delta \vdash \mathsf{let}\ x = E\ \mathsf{in}\ E'' : T$, which must follow by an instance of (EXP LET) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash \mathsf{let}\ x = E\ \mathsf{in}\ E'' : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $E \rightsquigarrow [\Delta' \mid D]$
— $\Gamma; \Delta, \Delta' \hookrightarrow \Gamma; \Delta_1, \Delta_2$
— $\Gamma; \Delta_1 \vdash D : U$
— $(\Gamma; \Delta_2) \bullet x : U \vdash E'' : V$
By Lemma B.41 we know that $E \Rrightarrow E'$ implies $E' \rightsquigarrow [\Delta' \mid D']$ with $D \Rrightarrow D'$. Since Lemma B.41 is depth-preserving, we can apply the inductive hypothesis and get $\Gamma; \Delta_1 \vdash D' : U$, hence the conclusion $\Gamma; \Delta \vdash \mathsf{let}\ x = E'\ \mathsf{in}\ E'' : T$ follows by applying (EXP LET) and (EXP SUBSUM).
*Case* (HEAT RES): assume $(\nu a)E \Rrightarrow (\nu a)E'$ by the premise $E \Rrightarrow E'$. Assume further that $\Gamma; \Delta \vdash (\nu a)E : T$. The typing judgement must follow by an instance of (EXP RES) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash (\nu a)E : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $E \rightsquigarrow^a [\Delta' \mid D]$
— $\Gamma, a \updownarrow T'; \Delta, \Delta' \vdash D : V$
By Lemma B.41 we know that $E \Rrightarrow E'$ implies $E' \rightsquigarrow^a [\Delta' \mid D']$ with $D \Rrightarrow D'$. Since Lemma B.41 is depth-preserving, we can apply the inductive hypothesis and get $\Gamma, a \updownarrow T'; \Delta, \Delta' \vdash D' : V$, hence the conclusion $\Gamma; \Delta \vdash (\nu a)E' : T$ follows by applying (EXP RES) and (EXP SUBSUM).
*Case* (HEAT FORK 1): assume $E \upharpoonright E'' \Rrightarrow E' \upharpoonright E''$ by the premise $E \Rrightarrow E'$. Assume further that $\Gamma; \Delta \vdash E \upharpoonright E'' : T$. The judgement must follow by an instance of (EXP FORK) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash E \upharpoonright E'' : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $E \rightsquigarrow [\Delta' \mid D]$
— $E'' \rightsquigarrow [\Delta'' \mid D'']$
— $\Gamma; \Delta_A, \Delta', \Delta'' \hookrightarrow \Gamma; \Delta_1, \Delta_2$
— $\Gamma; \Delta_1 \vdash D : U$
— $\Gamma; \Delta_2 \vdash D'' : V$

By Lemma B.41 we know that $E \Rrightarrow E'$ implies $E' \rightsquigarrow [\Delta' \mid D']$ with $D \Rrightarrow D'$. Since Lemma B.41 is depth-preserving, we can apply the inductive hypothesis and get $\Gamma; \Delta_1 \vdash D' : U$, hence the conclusion $\Gamma; \Delta \vdash E' \Lsh E'' : T$ follows by applying (EXP FORK) and (EXP SUBSUM).

*Case* (HEAT FORK 2): the case is analogous to HEAT FORK 1.

*Case* (HEAT FORK ()): assume $() \Lsh E \Rrightarrow E$ with $\Gamma; \Delta \vdash () \Lsh E : T$. The judgement must follow by an instance of (EXP FORK) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash () \Lsh E : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $() \rightsquigarrow [\emptyset \mid ()]$
— $E \rightsquigarrow [\Delta' \mid E']$
— $\Gamma; \Delta_A, \Delta' \hookrightarrow \Gamma; \Delta_1, \Delta_2$
— $\Gamma; \Delta_1 \vdash () : U$
— $\Gamma; \Delta_2 \vdash E' : V$

Notice that both $\Gamma; \Delta_1 \vdash \diamond$ and $\Gamma; \Delta_2 \vdash \diamond$ by Lemma B.5, thus $\Gamma; \Delta_1, \Delta_2 \vdash \diamond$ by Lemma B.6. By Lemma B.7 we then know that $\Gamma; \Delta_2 \vdash E' : V$ implies $\Gamma; \Delta_1, \Delta_2 \vdash E' : V$, hence we have $\Gamma; \Delta_A, \Delta' \vdash E' : V$ by Lemma B.9 and this implies $\Gamma; \Delta_A \vdash E : V$ by Lemma B.45. The conclusion $\Gamma; \Delta \vdash E : T$ follows by (EXP SUBSUM).

Assume now $E \Rrightarrow () \Lsh E$ with $\Gamma; \Delta \vdash E : T$. The judgement must follow by an instance of a structural rule after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash E : V$ and $\Gamma; \Delta_B \vdash V <: T$ with $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$. By Lemma B.5 we know that $\Gamma; \Delta_A \vdash E : V$ implies $\Gamma; \Delta_A \vdash \diamond$, hence $\Gamma; \emptyset \vdash \diamond$ again by Lemma B.5 and $\Gamma; \emptyset \vdash () : \mathsf{unit}$ by (VAL UNIT). Let then $E \rightsquigarrow [\Delta' \mid E']$: since $\Gamma; \Delta_A \vdash E : V$, we have $\Gamma; \Delta_A, \Delta' \vdash E' : V$ by Lemma B.44. Hence, we have:
— $() \rightsquigarrow [\emptyset \mid ()]$
— $E \rightsquigarrow [\Delta' \mid E']$
— $\Gamma; \emptyset \vdash () : \mathsf{unit}$
— $\Gamma; \Delta_A, \Delta' \vdash E' : V$

which imply $\Gamma; \Delta_A \vdash () \Lsh E : V$ by (EXP FORK). The conclusion $\Gamma; \Delta \vdash () \Lsh E : T$ follows by (EXP SUBSUM).

*Case* (HEAT MSG ()): let $a!M \Rrightarrow a!M \Lsh ()$ with $\Gamma; \Delta \vdash a!M : T$. The judgement must follow by an instance of (EXP SEND) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash a!M : \mathsf{unit}$ and $\Gamma; \Delta_B \vdash \mathsf{unit} <: T$ with $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$. By Lemma B.5 we know that $\Gamma; \Delta_A \vdash a!M : \mathsf{unit}$ implies $\Gamma; \Delta_A \vdash \diamond$, hence $\Gamma; \emptyset \vdash \diamond$ again by Lemma B.5 and $\Gamma; \emptyset \vdash () : \mathsf{unit}$ by (VAL UNIT). Thus, we have:
— $a!M \rightsquigarrow [\emptyset \mid a!M]$
— $() \rightsquigarrow [\emptyset \mid ()]$
— $\Gamma; \Delta_A \vdash a!M : \mathsf{unit}$
— $\Gamma; \emptyset \vdash () : \mathsf{unit}$

which imply $\Gamma; \Delta_A \vdash a!M \Lsh () : \mathsf{unit}$ by (EXP FORK). Hence, the conclusion $\Gamma; \Delta \vdash a!M \Lsh () : T$ follows by (EXP SUBSUM).

*Case* (HEAT ASSUME ()): let $\mathsf{assume}\ F \Rrightarrow \mathsf{assume}\ F \Lsh ()$ with $\Gamma; \Delta \vdash \mathsf{assume}\ F : T$. We distinguish two cases. Let $F = \mathbf{1}$, then $\Gamma; \Delta \vdash \mathsf{assume}\ \mathbf{1} : T$ must follow by an instance of (EXP TRUE) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash \mathsf{assume}\ \mathbf{1} : \mathsf{unit}$ and $\Gamma; \Delta_B \vdash \mathsf{unit} <: T$ with $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$. Now notice that $\mathsf{assume}\ \mathbf{1} \rightsquigarrow [\emptyset \mid \mathsf{assume}\ \mathbf{1}]$ and $() \rightsquigarrow [\emptyset \mid ()]$, hence we can construct the following type derivation:

$$\dfrac{\text{EXP FORK}\ \dfrac{\Gamma; \Delta_A \vdash \mathsf{assume}\ \mathbf{1} : \mathsf{unit} \qquad \dfrac{\Gamma; \emptyset \vdash \diamond}{\Gamma; \emptyset \vdash () : \mathsf{unit}}\ \text{VAL UNIT}}{\Gamma; \Delta_A \vdash \mathsf{assume}\ \mathbf{1} \Lsh () : \mathsf{unit}} \qquad \Gamma; \Delta_B \vdash \mathsf{unit} <: T}{\Gamma; \Delta \vdash \mathsf{assume}\ \mathbf{1} \Lsh () : T}\ \text{EXP SUBSUM}$$

Let now $F \neq \mathbf{1}$, then $\Gamma;\Delta \vdash \mathsf{assume}\ F : T$ must follow by an instance of (EXP ASSUME) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma;\Delta_A \vdash \mathsf{assume}\ F : V$ and $\Gamma;\Delta_B \vdash V <: T$ with $\Gamma;\Delta \hookrightarrow \Gamma;\Delta_A,\Delta_B$ and $\Gamma;\Delta_A, F \vdash \mathsf{assume}\ \mathbf{1} : V$. The latter must have been derived by an instance of (EXP TRUE) after an instance of (EXP SUBSUM), hence we have $\Gamma;\Delta_1 \vdash \mathsf{assume}\ \mathbf{1} : \mathsf{unit}$ and $\Gamma;\Delta_2 : \mathsf{unit} <: V$ with $\Gamma;\Delta_A, F \hookrightarrow \Gamma;\Delta_1,\Delta_2$. Now notice that $\mathsf{assume}\ F \rightsquigarrow [F \mid \mathsf{assume}\ \mathbf{1}]$ and $() \rightsquigarrow [\emptyset \mid ()]$, hence we can construct the following type derivation:

$$\dfrac{\dfrac{\dfrac{\Gamma;\emptyset \vdash \diamond}{\Gamma;\emptyset \vdash \mathsf{assume}\ \mathbf{1} : \mathsf{unit}}\ \text{Exp True} \qquad \dfrac{\dfrac{\Gamma;\Delta_1 \vdash \diamond}{\Gamma;\Delta_1 \vdash () : \mathsf{unit}}\ \text{Val Unit} \qquad \Gamma;\Delta_2 \vdash \mathsf{unit} <: V}{\Gamma;\Delta_A, F \vdash () : V}\ \text{Exp Subsum}}{\Gamma;\Delta_A \vdash \mathsf{assume}\ F \Rsh () : V}\ \text{Exp Fork} \qquad \Gamma;\Delta_B \vdash V <: T}{\Gamma;\Delta \vdash \mathsf{assume}\ F \Rsh () : T}\ \text{Exp Subsum}$$

*Case* (HEAT ASSERT ()): the case is analogous to (HEAT MSG ()).

*Case* (HEAT RES FORK 1): let $E' \Rsh (\nu a)E \Rrightarrow (\nu a)(E' \Rsh E)$ with $a \notin fn(E')$. Assume further $\Gamma;\Delta \vdash E' \Rsh (\nu a)E : T$. The judgement must follow by an instance of (EXP FORK) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma;\Delta_A \vdash E' \Rsh (\nu a)E : V$ and $\Gamma;\Delta_B \vdash V <: T$ with:

— $\Gamma;\Delta \hookrightarrow \Gamma;\Delta_A,\Delta_B$
— $E' \rightsquigarrow^{\emptyset} [\Delta' \mid D']$
— $(\nu a)E \rightsquigarrow^{\emptyset} [\Delta'' \mid (\nu a)D]$ with $E \rightsquigarrow^{a} [\Delta'' \mid D]$
— $\Gamma;\Delta_A,\Delta',\Delta'' \hookrightarrow \Gamma;\Delta_1,\Delta_2$
— $\Gamma;\Delta_1 \vdash D' : U$
— $\Gamma;\Delta_2 \vdash (\nu a)D : V$

The latter judgement must follow by an instance of (EXP RES) after an instance of (EXP SUBSUM). Notice that $E \rightsquigarrow^{a} [\Delta'' \mid D]$ implies $D \rightsquigarrow^{a} [\emptyset \mid D]$ by Lemma B.40, hence we simply have $\Gamma, a \updownarrow T';\Delta_{21} \vdash D : U$ and $\Gamma;\Delta_{22} \vdash U <: V$ with $\Gamma;\Delta_2 \hookrightarrow \Gamma;\Delta_{21},\Delta_{22}$. We also notice that $E' \rightsquigarrow^{\emptyset} [\Delta' \mid D']$ and $a \notin fn(E')$ imply $E' \rightsquigarrow^{a} [\Delta' \mid D']$ by Lemma B.37, hence $E' \Rsh E \rightsquigarrow^{a} [\Delta',\Delta'' \mid D' \Rsh D]$ by (EXTR FORK). Moreover, we know that $E' \rightsquigarrow^{\emptyset} [\Delta' \mid D']$ implies $D' \rightsquigarrow^{\emptyset} [\emptyset \mid D']$ by Lemma B.40, hence we can construct the following type derivation:

$$\dfrac{\dfrac{D \rightsquigarrow^{\emptyset} [\Delta''' \mid D''] \qquad \dfrac{\Gamma;\Delta_1 \vdash D' : U}{\Gamma, a \updownarrow T';\Delta_1 \vdash D' : U} \qquad \dfrac{(1)}{\Gamma, a \updownarrow T';\Delta_2,\Delta''' \vdash D'' : V}}{\Gamma, a \updownarrow T';\Delta_A,\Delta',\Delta'' \vdash D' \Rsh D : V}\ \text{Exp Fork}}{\Gamma;\Delta_A \vdash (\nu a)(E' \Rsh E) : V}\ \text{Exp Res}$$

where $(1)$ is constructed as follows:

$$\dfrac{\dfrac{(2)}{\Gamma, a \updownarrow T';\Delta_{21},\Delta''' \vdash D'' : U} \qquad \dfrac{\Gamma;\Delta_{22} \vdash U <: V}{\Gamma, a \updownarrow T';\Delta_{22} \vdash U <: V}}{\Gamma, a \updownarrow T';\Delta_2,\Delta''' \vdash D'' : V}\ \text{Exp Subsum}$$

and $(2)$ is derived from $\Gamma, a \updownarrow T';\Delta_{21} \vdash D : U$ and $D \rightsquigarrow^{\emptyset} [\Delta''' \mid D'']$ using Lemma B.44. We conclude $\Gamma;\Delta \vdash (\nu a)(E' \Rsh E) : T$ by (EXP SUBSUM).

*Case* (HEAT RES FORK 2): the case is analogous to (HEAT RES FORK 1).

*Case* (HEAT RES LET): assume $\mathsf{let}\ x = (\nu a)E\ \mathsf{in}\ E' \Rrightarrow (\nu a)(\mathsf{let}\ x = E\ \mathsf{in}\ E')$ with $a \notin fn(E')$. Assume further $\Gamma;\Delta \vdash \mathsf{let}\ x = (\nu a)E\ \mathsf{in}\ E' : T$. The judgement must follow by an instance of (EXP LET) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma;\Delta_A \vdash \mathsf{let}\ x = (\nu a)E\ \mathsf{in}\ E' : V$ and $\Gamma;\Delta_B \vdash V <: T$ with:

— $\Gamma;\Delta \hookrightarrow \Gamma;\Delta_A,\Delta_B$
— $(\nu a)E \rightsquigarrow^{\emptyset} [\Delta' \mid (\nu a)D]$ with $E \rightsquigarrow^{a} [\Delta' \mid D]$

— $\Gamma; \Delta_A, \Delta' \hookrightarrow \Gamma; \Delta_1, \Delta_2$
— $\Gamma; \Delta_1 \vdash (\nu a)D : U$
— $(\Gamma; \Delta_2) \bullet x : U \vdash E' : V$

Now we note that $\Gamma; \Delta_1 \vdash (\nu a)D : U$ must follow by an instance of (EXP RES) after an instance of (EXP SUBSUM). Since $E \rightsquigarrow^a [\Delta' \mid D]$ implies $D \rightsquigarrow^a [\emptyset \mid D]$ by Lemma B.40, we note that we simply have $\Gamma, a \updownarrow T'; \Delta_{11} \vdash D : U'$ and $\Gamma; \Delta_{12} \vdash U' <: U$ with $\Gamma; \Delta_1 \hookrightarrow \Gamma; \Delta_{11}, \Delta_{12}$. Notice also that $E \rightsquigarrow^a [\Delta' \mid D]$ implies $\mathsf{let}\ x = E\ \mathsf{in}\ E' \rightsquigarrow^a [\Delta' \mid \mathsf{let}\ x = D\ \mathsf{in}\ E']$ by (EXTR LET). We can then construct the following type derivation:

$$\textsc{Exp Res}\ \dfrac{\textsc{Exp Let}\ \dfrac{D \rightsquigarrow^{\emptyset} [\Delta'' \mid D'] \qquad \dfrac{(1)}{\Gamma, a \updownarrow T'; \Delta_1, \Delta'' \vdash D' : U} \qquad \dfrac{(\Gamma; \Delta_2) \bullet x : U \vdash E' : V}{(\Gamma, a \updownarrow T'; \Delta_2) \bullet x : U \vdash E' : V}}{\Gamma, a \updownarrow T'; \Delta_A, \Delta' \vdash \mathsf{let}\ x = D\ \mathsf{in}\ E' : V}}{\Gamma; \Delta_A \vdash (\nu a)(\mathsf{let}\ x = E\ \mathsf{in}\ E') : V}$$

where $(1)$ is constructed as follows:

$$\textsc{Exp Subsum}\ \dfrac{\dfrac{(2)}{\Gamma, a \updownarrow T'; \Delta_{11}, \Delta'' \vdash D' : U'} \qquad \dfrac{\Gamma; \Delta_{12} \vdash U' <: U}{\Gamma, a \updownarrow T'; \Delta_{12} \vdash U' <: U}}{\Gamma, a \updownarrow T'; \Delta_1, \Delta'' \vdash D' : U}$$

and $(2)$ is derived from $\Gamma, a \updownarrow T'; \Delta_{11} \vdash D : U'$ and $D \rightsquigarrow^{\emptyset} [\Delta'' \mid D']$ using Lemma B.44. We conclude $\Gamma; \Delta \vdash (\nu a)(\mathsf{let}\ x = E\ \mathsf{in}\ E') : T$ by (EXP SUBSUM).

*Case* (HEAT FORK COMM): assume $(E \Rsh E') \Rsh E'' \Rrightarrow (E' \Rsh E) \Rsh E''$ with $\Gamma; \Delta \vdash (E \Rsh E') \Rsh E'' : T$. The judgement must follow by an instance of (EXP FORK) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash (E \Rsh E') \Rsh E'' : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:

— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $E \Rsh E' \rightsquigarrow [\Delta_1, \Delta_2 \mid D_1 \Rsh D_2]$ with $E \rightsquigarrow [\Delta_1 \mid D_1]$ and $E' \rightsquigarrow [\Delta_2 \mid D_2]$
— $E'' \rightsquigarrow [\Delta_3 \mid D_3]$
— $\Gamma; \Delta_A, \Delta_1, \Delta_2, \Delta_3 \hookrightarrow \Gamma; \Delta'_1, \Delta'_2$
— $\Gamma; \Delta'_1 \vdash D_1 \Rsh D_2 : U$
— $\Gamma; \Delta'_2 \vdash D_3 : V$

Now we notice that $\Gamma; \Delta'_1 \vdash D_1 \Rsh D_2 : U$ must have been derived by an instance of (EXP FORK) after an instance of (EXP SUBSUM). Since $D_1 \rightsquigarrow [\emptyset \mid D_1]$ and $D_2 \rightsquigarrow [\emptyset \mid D_2]$ by Lemma B.40, it must be the case that $\Gamma; \Delta'_{11} \vdash D_1 \Rsh D_2 : U_2$ and $\Gamma; \Delta'_{12} \vdash U_2 <: U$ with:

— $\Gamma; \Delta'_1 \hookrightarrow \Gamma; \Delta'_{11}, \Delta'_{12}$
— $\Gamma; \Delta'_{11} \hookrightarrow \Gamma; \Delta'_A, \Delta'_B$
— $\Gamma; \Delta'_A \vdash D_1 : U_1$
— $\Gamma; \Delta'_B \vdash D_2 : U_2$

We have $E' \Rsh E \rightsquigarrow [\Delta_1, \Delta_2 \mid D_2 \Rsh D_1]$ by applying (EXTR FORK) to $E \rightsquigarrow [\Delta_1 \mid D_1]$ and $E' \rightsquigarrow [\Delta_2 \mid D_2]$, hence we can construct the following type derivation:

$$\textsc{Exp Fork}\ \dfrac{\textsc{Exp Fork}\ \dfrac{\Gamma; \Delta'_B \vdash D_2 : U_2 \qquad \Gamma; \Delta'_A \vdash D_1 : U_1}{\Gamma; \Delta'_B, \Delta'_A \vdash D_2 \Rsh D_1 : U_1} \qquad \Gamma; \Delta'_2 \vdash D_3 : V}{\Gamma; \Delta_A \vdash (E' \Rsh E) \Rsh E'' : V}$$

since $\Gamma; \Delta_A, \Delta_1, \Delta_2, \Delta_3 \hookrightarrow \Gamma; (\Delta'_B, \Delta'_A), \Delta'_2$ can be derived by Lemma B.8. Finally, we conclude $\Gamma; \Delta \vdash (E' \Rsh E) \Rsh E'' : T$ by (EXP SUBSUM).

*Case* (HEAT FORK ASSOC): assume $(E \Rsh E') \Rsh E'' \Rrightarrow E \Rsh (E' \Rsh E'')$ with $\Gamma; \Delta \vdash (E \Rsh E') \Rsh E'' : T$. The judgement must follow by an instance of (EXP FORK) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash (E \Rsh E') \Rsh E'' : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:

— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $E \Rsh E' \rightsquigarrow [\Delta_1, \Delta_2 \mid D_1 \Rsh D_2]$ with $E \rightsquigarrow [\Delta_1 \mid D_1]$ and $E' \rightsquigarrow [\Delta_2 \mid D_2]$
— $E'' \rightsquigarrow [\Delta_3 \mid D_3]$
— $\Gamma; \Delta_A, \Delta_1, \Delta_2, \Delta_3 \hookrightarrow \Gamma; \Delta'_1, \Delta'_2$
— $\Gamma; \Delta'_1 \vdash D_1 \Rsh D_2 : U$
— $\Gamma; \Delta'_2 \vdash D_3 : V$

Now we notice that $\Gamma; \Delta'_1 \vdash D_1 \Rsh D_2 : U$ must have been derived by an instance of (EXP FORK) after an instance of (EXP SUBSUM). Since $D_1 \rightsquigarrow [\emptyset \mid D_1]$ and $D_2 \rightsquigarrow [\emptyset \mid D_2]$ by Lemma B.40, it must be the case that $\Gamma; \Delta'_{11} \vdash D_1 \Rsh D_2 : U_2$ and $\Gamma; \Delta'_{12} \vdash U_2 <: U$ with:
— $\Gamma; \Delta'_1 \hookrightarrow \Gamma; \Delta'_{11}, \Delta'_{12}$
— $\Gamma; \Delta'_{11} \hookrightarrow \Gamma; \Delta'_A, \Delta'_B$
— $\Gamma; \Delta'_A \vdash D_1 : U_1$
— $\Gamma; \Delta'_B \vdash D_2 : U_2$

We have $E' \Rsh E'' \rightsquigarrow [\Delta_2, \Delta_3 \mid D_2 \Rsh D_3]$ by applying (EXTR FORK) to $E' \rightsquigarrow [\Delta_2 \mid D_2]$ and $E'' \rightsquigarrow [\Delta_3 \mid D_3]$. Moreover, we know that $E'' \rightsquigarrow [\Delta_3 \mid D_3]$ implies $D_3 \rightsquigarrow [\emptyset \mid D_3]$ by Lemma B.40, hence we can construct the following type derivation:

$$\textsc{Exp Fork}\ \dfrac{\Gamma; \Delta'_A \vdash D_1 : U_1 \qquad \dfrac{\Gamma; \Delta'_B \vdash D_2 : U_2 \qquad \Gamma; \Delta'_2 \vdash D_3 : V}{\Gamma; \Delta'_B, \Delta'_2 \vdash D_2 \Rsh D_3 : V}\ \textsc{Exp Fork}}{\Gamma; \Delta_A \vdash E \Rsh (E' \Rsh E'') : V}$$

since $\Gamma; \Delta_A, \Delta_1, \Delta_2, \Delta_3 \hookrightarrow \Gamma; \Delta'_A, (\Delta'_B, \Delta'_2)$ can be derived by Lemma B.8. Finally, we conclude $\Gamma; \Delta \vdash E \Rsh (E' \Rsh E'') : T$ by (EXP SUBSUM).

*Case* (HEAT FORK LET): assume $\mathsf{let}\ x = (E \Rsh E')\ \mathsf{in}\ E'' \Rrightarrow E \Rsh (\mathsf{let}\ x = E'\ \mathsf{in}\ E'')$ with $\Gamma; \Delta \vdash \mathsf{let}\ x = (E \Rsh E')\ \mathsf{in}\ E'' : T$. The judgement must follow by an instance of (EXP LET) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash \mathsf{let}\ x = (E \Rsh E')\ \mathsf{in}\ E'' : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $E \Rsh E' \rightsquigarrow [\Delta_1, \Delta_2 \mid D_1 \Rsh D_2]$ with $E \rightsquigarrow [\Delta_1 \mid D_1]$ and $E' \rightsquigarrow [\Delta_2 \mid D_2]$
— $\Gamma; \Delta_A, \Delta_1, \Delta_2 \hookrightarrow \Gamma; \Delta'_1, \Delta'_2$
— $\Gamma; \Delta'_1 \vdash D_1 \Rsh D_2 : U$
— $(\Gamma; \Delta'_2) \bullet x : U \vdash E'' : V$

Now we notice that $\Gamma; \Delta'_1 \vdash D_1 \Rsh D_2 : U$ must have been derived by an instance of (EXP FORK) after an instance of (EXP SUBSUM). Since $D_1 \rightsquigarrow [\emptyset \mid D_1]$ and $D_2 \rightsquigarrow [\emptyset \mid D_2]$ by Lemma B.40, it must be the case that $\Gamma; \Delta'_{11} \vdash D_1 \Rsh D_2 : U_2$ and $\Gamma; \Delta'_{12} \vdash U_2 <: U$ with:
— $\Gamma; \Delta'_1 \hookrightarrow \Gamma; \Delta'_{11}, \Delta'_{12}$
— $\Gamma; \Delta'_{11} \hookrightarrow \Gamma; \Delta'_A, \Delta'_B$
— $\Gamma; \Delta'_A \vdash D_1 : U_1$
— $\Gamma; \Delta'_B \vdash D_2 : U_2$

We have $\mathsf{let}\ x = E'\ \mathsf{in}\ E'' \rightsquigarrow [\Delta_2 \mid \mathsf{let}\ x = D_2\ \mathsf{in}\ E'']$ by (EXTR LET), hence we can construct the following type derivation:

$$\textsc{Exp Fork}\ \dfrac{\Gamma; \Delta'_A \vdash D_1 : U_1 \qquad \dfrac{\textsc{Exp Subsum}\ \dfrac{\Gamma; \Delta'_B \vdash D_2 : U_2 \qquad \Gamma; \Delta'_{12} \vdash U_2 <: U}{\Gamma; \Delta'_B, \Delta'_{12} \vdash D_2 : U} \qquad (\Gamma; \Delta'_2) \bullet x : U \vdash E'' : V}{\Gamma; \Delta'_B, \Delta'_{12}, \Delta'_2 \vdash \mathsf{let}\ x = D_2\ \mathsf{in}\ E'' : V}\ \textsc{Exp Let}}{\Gamma; \Delta_A \vdash E \Rsh (\mathsf{let}\ x = E'\ \mathsf{in}\ E'') : V}$$

since $\Gamma; \Delta_A, \Delta_1, \Delta_2 \hookrightarrow \Gamma; \Delta'_A, (\Delta'_B, \Delta'_{12}, \Delta'_2)$ can be derived by Lemma B.8. We conclude $\Gamma; \Delta \vdash E \Rsh (\mathsf{let}\ x = E'\ \mathsf{in}\ E'') : T$ by (EXP SUBSUM).

Assume now $E \Rsh (\mathsf{let}\ x = E'\ \mathsf{in}\ E'') \Rrightarrow \mathsf{let}\ x = (E \Rsh E')\ \mathsf{in}\ E''$ with $\Gamma; \Delta \vdash E \Rsh (\mathsf{let}\ x = E'\ \mathsf{in}\ E'') : T$. The judgement must follow by an instance of (EXP FORK) after

an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash E \upharpoonright (\mathsf{let}\ x = E'\ \mathsf{in}\ E'') : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $E \rightsquigarrow [\Delta_1 \mid D_1]$
— $\mathsf{let}\ x = E'\ \mathsf{in}\ E'' \rightsquigarrow [\Delta_2 \mid \mathsf{let}\ x = D_2\ \mathsf{in}\ E'']$ with $E' \rightsquigarrow [\Delta_2 \mid D_2]$
— $\Gamma; \Delta_A, \Delta_1, \Delta_2 \hookrightarrow \Gamma; \Delta'_1, \Delta'_2$
— $\Gamma; \Delta'_1 \vdash D_1 : U_1$
— $\Gamma; \Delta'_2 \vdash \mathsf{let}\ x = D_2\ \mathsf{in}\ E'' : V$
Now we notice that $\Gamma; \Delta'_2 \vdash \mathsf{let}\ x = D_2\ \mathsf{in}\ E'' : V$ must have been derived by an instance of (EXP LET) after an instance of (EXP SUBSUM). Since $E' \rightsquigarrow [\Delta_2 \mid D_2]$ implies $D_2 \rightsquigarrow [\emptyset \mid D_2]$ by Lemma B.40, it must be the case that $\Gamma; \Delta'_{21} \vdash \mathsf{let}\ x = D_2\ \mathsf{in}\ E'' : U_3$ and $\Gamma; \Delta'_{22} \vdash U_3 <: V$ with:
— $\Gamma; \Delta'_2 \hookrightarrow \Gamma; \Delta'_{21}, \Delta'_{22}$
— $\Gamma; \Delta'_{21} \hookrightarrow \Gamma; \Delta'_A, \Delta'_B$
— $\Gamma; \Delta'_A \vdash D_2 : U_2$
— $(\Gamma; \Delta'_B) \bullet x : U_2 \vdash E'' : U_3$
We have $E \upharpoonright E' \rightsquigarrow [\Delta_1, \Delta_2 \mid D_1 \upharpoonright D_2]$ by (EXTR FORK). Moreover, we know that $E \rightsquigarrow [\Delta_1 \mid D_1]$ implies $D_1 \rightsquigarrow [\emptyset \mid D_1]$ by Lemma B.40, hence we can construct the following type derivation:

$$\dfrac{\dfrac{\Gamma; \Delta'_1 \vdash D_1 : U_1 \qquad \Gamma; \Delta'_A \vdash D_2 : U_2}{\Gamma; \Delta'_1, \Delta'_A \vdash D_1 \upharpoonright D_2 : U_2}\,\textsc{Exp Fork} \qquad \dfrac{(1)}{(\Gamma; \Delta'_B, \Delta'_{22}) \bullet x : U_2 \vdash E'' : V}}{\Gamma; \Delta_A \vdash \mathsf{let}\ x = (E \upharpoonright E')\ \mathsf{in}\ E'' : V}\,\textsc{Exp Let}$$

where $\Gamma; \Delta_A, \Delta_1, \Delta_2 \hookrightarrow \Gamma; (\Delta'_1, \Delta'_A), (\Delta'_B, \Delta'_{22})$ can be derived by Lemma B.8, and the derivation $(1)$ is constructed as follows:

$$\dfrac{(\Gamma; \Delta'_B) \bullet x : U_2 \vdash E'' : U_3 \qquad \dfrac{\Gamma; \Delta'_{22} \vdash U_3 <: V}{\Gamma, x : \psi(U_2); \Delta'_{22} \vdash U_3 <: V}}{(\Gamma; \Delta'_B, \Delta'_{22}) \bullet x : U_2 \vdash E'' : V}\,\textsc{Exp Subsum}$$

We conclude $\Gamma; \Delta \vdash \mathsf{let}\ x = (E \upharpoonright E')\ \mathsf{in}\ E'' : T$ by (EXP SUBSUM).

□

The next simple lemma states that tautologies can be safely removed from any typing environment. This is used in some cases of the Subject Reduction proof, to deal with the logical formulas we explicitly introduce in the typing environment to make type-checking more precise (cf. EXP SPLIT).

LEMMA B.47 (REMOVING TAUTOLOGIES). *If* $\Gamma; \Delta, F \vdash E : T$ *and* $\emptyset \vdash F$*, then* $\Gamma; \Delta \vdash E : T$.

PROOF. We know that $\Gamma; \Delta, F \vdash E : T$ implies $\Gamma; \Delta, F \vdash \diamond$ by Lemma B.5. Moreover, the latter implies $\Gamma; \Delta \vdash \diamond$ again by Lemma B.5. Since $\Delta, F \vdash \Delta, F$ by Lemma B.2, we can derive $\Gamma; \Delta \hookrightarrow \Gamma; \Delta, F$ as follows:

$$\dfrac{\Gamma; \Delta \vdash \diamond \qquad (*)\ \dfrac{\emptyset \vdash F \qquad \Delta, F \vdash \Delta, F}{\Delta \vdash \Delta, F} \qquad \Gamma; \Delta, F \vdash \diamond}{\Gamma; \Delta \hookrightarrow \Gamma; \Delta, F}\,\textsc{Rewrite}$$

Here (*) follows by using a standard Cut elimination argument. Since $\Gamma; \Delta, F \vdash E : T$, the conclusion $\Gamma; \Delta \vdash E : T$ follows by Lemma B.9. □

We can finally prove the Subject Reduction theorem. Its statement is remarkably simple: this is mainly due to our type system design, which discharges to the underlying affine logical framework all the complicated issues related to resource consumption. Thus, we do not need to explicitly track in the semantics which resources are consumed upon reduction, unlike to many other substructural type systems.

THEOREM B.48 (SUBJECT REDUCTION). *Let* $fv(E) = \emptyset$*. If* $\Gamma; \Delta \vdash E : T$ *and* $E \rightarrow E'$*, then* $\Gamma; \Delta \vdash E' : T$.

PROOF. By induction on the derivation of $E \rightarrow E'$. In the proof we implicitly appeal to Lemma B.5 and Lemma B.8 several times:

*Case* (RED FUN): assume $(\lambda x. E)\ N \rightarrow E\{N/x\}$ and $\Gamma; \Delta \vdash (\lambda x. E)\ N : T$. The typing judgement must follow by an instance of (EXP APPL) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash (\lambda x. E)\ N : U'\{N/x\}$ and $\Gamma; \Delta_B \vdash U'\{N/x\} <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $\Gamma; \Delta_A \hookrightarrow \Gamma; \Delta_1, \Delta_2$
— $\Gamma; \Delta_1 \vdash \lambda x. E : x : T' \rightarrow U'$
— $\Gamma; \Delta_2 \vdash N : T'$
By Lemma B.32 we know that $\Gamma; \Delta_1 \vdash \lambda x. E : x : T' \rightarrow U'$ implies $(\Gamma; \Delta_1) \bullet x : T' \vdash E : U'$. Now notice that $x \notin dom(\Gamma)$ by Lemma B.5, hence $x \notin fv(\Delta_1)$ by Lemma B.10. By applying Lemma B.28, we then get $\Gamma; \Delta_1, \Delta_2 \vdash E\{N/x\} : U'\{N/x\}$. Since $\Gamma; \Delta \hookrightarrow \Gamma; (\Delta_1, \Delta_2), \Delta_B$, the conclusion $\Gamma; \Delta \vdash E\{N/x\} : T$ follows by an application of (EXP SUBSUM).

*Case* (RED SPLIT): assume $\mathsf{let}\ (x, y) = (M, N)\ \mathsf{in}\ E \rightarrow E\{M/x\}\{N/y\}$ and $\Gamma; \Delta \vdash \mathsf{let}\ (x, y) = (M, N)\ \mathsf{in}\ E : T$. The typing judgement must follow by an instance of (EXP SPLIT) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash \mathsf{let}\ (x, y) = (M, N)\ \mathsf{in}\ E : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $\Gamma; \Delta_A \hookrightarrow \Gamma; \Delta_1, \Delta_2$
— $\Gamma; \Delta_1 \vdash (M, N) : x : T' * U'$
— $(\Gamma; \Delta_2) \bullet x : T' \bullet y : U' \bullet !((x, y) = (M, N)) \vdash E : V$
— $\{x, y\} \cap fv(V) = \emptyset$
By Lemma B.33 we know that $\Gamma; \Delta_1 \vdash (M, N) : x : T' * U'$ implies:
— $\Gamma; \Delta_1 \hookrightarrow \Gamma; \Delta_{11}, \Delta_{12}$
— $\Gamma; \Delta_{11} \vdash M : T'$
— $\Gamma; \Delta_{12} \vdash N : U'\{M/x\}$
Now notice that $x \notin dom(\Gamma)$ by Lemma B.5, hence $x \notin fv(\Delta_2)$ by Lemma B.10. By applying Lemma B.28 twice and noting that $\{x, y\} \cap fv(V) = \emptyset$, we then get $\Gamma; \Delta_{11}, \Delta_{12}, \Delta_2, !((M, N) = (M, N)) \vdash E : V$. Since $\emptyset \vdash !((M, N) = (M, N))$, the latter judgement implies $\Gamma; \Delta_{11}, \Delta_{12}, \Delta_2 \vdash E : V$ by Lemma B.47. Since $\Gamma; \Delta \hookrightarrow \Gamma; (\Delta_{11}, \Delta_{12}, \Delta_2), \Delta_B$, the conclusion $\Gamma; \Delta \vdash E : T$ follows by (EXP SUBSUM).

*Case* (RED MATCH): assume $\mathsf{match}\ h\ N\ \mathsf{with}\ h\ x\ \mathsf{then}\ E\ \mathsf{else}\ E' \rightarrow E\{N/x\}$ and $\Gamma; \Delta \vdash \mathsf{match}\ h\ N\ \mathsf{with}\ h\ x\ \mathsf{then}\ E\ \mathsf{else}\ E' : T$. The typing judgement must follow by an instance of (EXP MATCH) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash \mathsf{match}\ h\ N\ \mathsf{with}\ h\ x\ \mathsf{then}\ E\ \mathsf{else}\ E' : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $\Gamma; \Delta_A \hookrightarrow \Gamma; \Delta_1, \Delta_2$
— $\Gamma; \Delta_1 \vdash h\ N : T'$
— $(\Gamma; \Delta_2) \bullet x : U' \bullet !(h\ x = h\ N) \vdash E : V$
— $\Gamma; \Delta_2 \vdash E' : V$
— $(h, T', U') \in \{(\mathsf{inl}, T_1 + T_2, T_1), (\mathsf{inr}, T_1 + T_2, T_2), (\mathsf{fold}, \mu\alpha. T_1, T_1\{\mu\alpha. T_1/\alpha\})\}$
According to the form of $h$, we invoke either Lemma B.34 or Lemma B.35. and we get $\Gamma; \Delta_1 \vdash N : U'$. Now we notice that $\Gamma; \Delta_2 \vdash E' : V$ implies $fnfv(V) \subseteq dom(\Gamma)$

by Lemma B.5, hence the fact that $x \notin dom(\Gamma)$ implies $x \notin fv(V)$. Moreover, $x \notin dom(\Gamma)$ implies $x \notin fv(\Delta_2)$ by Lemma B.10. By applying Lemma B.28 we then get $\Gamma; \Delta_1, \Delta_2, !(h\ N = h\ N) \vdash E\{N/x\} : V$. Since $\emptyset \vdash !(h\ N = h\ N)$, the latter judgement implies $\Gamma; \Delta_1, \Delta_2 \vdash E\{N/x\} : V$ by Lemma B.47. Since $\Gamma; \Delta \hookrightarrow \Gamma; (\Delta_1, \Delta_2), \Delta_B$, the conclusion $\Gamma; \Delta \vdash E\{N/x\} : T$ follows by (EXP SUBSUM).
Assume now $\mathsf{match}\ M\ \mathsf{with}\ h\ x\ \mathsf{then}\ E\ \mathsf{else}\ E' \to E'$ with $M \neq h\ N$ for all $N$. The type derivation has the same structure as before, but for the obvious changes. Since $\Gamma; \Delta_2 \vdash E' : V$ and $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_2, \Delta_B$, the conclusion $\Gamma; \Delta \vdash E' : T$ follows by (EXP SUBSUM).

*Case* (RED EQ): assume we have $M = M \to \mathsf{true}$ and $\Gamma; \Delta \vdash M = M : T$. The typing judgement must follow by an instance of (EXP EQ) after an instance of (EXP SUBSUM), hence it must be the case that:

$$\Gamma; \Delta_A \vdash M = M : \{x : \mathsf{bool} \mid !(x = \mathsf{true} \multimap M = M)\}$$

and:

$$\Gamma; \Delta_B \vdash \{x : \mathsf{bool} \mid !(x = \mathsf{true} \multimap M = M)\} <: T.$$

with $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$. Recall now that $\mathsf{true} \triangleq \mathsf{inl}(\,)$ and $\mathsf{bool} \triangleq \mathsf{unit} + \mathsf{unit}$, so it is easy to show that we have $\Gamma; \Delta_A \vdash \mathsf{true} : \mathsf{bool}$. Now we note that:

$$\Gamma; \emptyset \vdash !(\mathsf{true} = \mathsf{true} \multimap M = M),$$

thus we get $\Gamma; \Delta_A \vdash \mathsf{true} : \{x : \mathsf{bool} \mid !(x = \mathsf{true} \multimap M = M)\}$ by (VAL REFINE) and the conclusion $\Gamma; \Delta \vdash \mathsf{true} : T$ follows by an application of (EXP SUBSUM).
Assume, instead, that $M = N \to \mathsf{false}$ with $M \neq N$ and $\Gamma; \Delta \vdash M = N : T$. The typing judgement must follow by an instance of (EXP EQ) after an instance of (EXP SUBSUM), hence it must be the case that:

$$\Gamma; \Delta_A \vdash M = N : \{x : \mathsf{bool} \mid !(x = \mathsf{true} \multimap M = N)\}$$

and:

$$\Gamma; \Delta_B \vdash \{x : \mathsf{bool} \mid !(x = \mathsf{true} \multimap M = N)\} <: T.$$

with $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$. Now we note that:

$$\Gamma; \emptyset \vdash !(\mathsf{false} = \mathsf{true} \multimap M = N),$$

thus we get $\Gamma; \Delta_A \vdash \mathsf{false} : \{x : \mathsf{bool} \mid !(x = \mathsf{true} \multimap M = N)\}$ by (VAL REFINE) and the conclusion $\Gamma; \Delta \vdash \mathsf{false} : T$ follows by an application of (EXP SUBSUM).

*Case* (RED COMM): assume $a!M \upharpoonright a? \to M$ and $\Gamma; \Delta \vdash a!M \upharpoonright a? : T$. The typing judgement must follow by an instance of (EXP FORK) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash a!M \upharpoonright a? : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $a!M \rightsquigarrow [\emptyset \mid a!M]$
— $a? \rightsquigarrow [\emptyset \mid a?]$
— $\Gamma; \Delta_A \hookrightarrow \Gamma; \Delta_1, \Delta_2$
— $\Gamma; \Delta_1 \vdash a!M : U$
— $\Gamma; \Delta_2 \vdash a? : V$

We notice that $\Gamma; \Delta_1 \vdash a!M : U$ must follow by an instance of (EXP SEND) after an instance of (EXP SUBSUM), hence:
— $\Gamma; \Delta_1 \hookrightarrow \Gamma; \Delta_{11}, \Delta_{12}$
— $\Gamma; \Delta_{11} \vdash a!M : \mathsf{unit}$
— $\Gamma; \Delta_{12} \vdash \mathsf{unit} <: U$
— $(a \updownarrow T') \in \Gamma$
— $\Gamma; \Delta_{11} \vdash M : T'$

We also notice that $\Gamma; \Delta_2 \vdash a? : V$ must follow by an instance of (EXP RECV) after an instance of (EXP SUBSUM), hence:
— $\Gamma; \Delta_2 \hookrightarrow \Gamma; \Delta_{21}, \Delta_{22}$
— $\Gamma; \Delta_{21} \vdash a? : T'$, since $(a \updownarrow T') \in \Gamma$
— $\Gamma; \Delta_{22} \vdash T' <: V$
Thus we get $\Gamma; \Delta_{11}, \Delta_{22} \vdash M : V$ by (EXP SUBSUM). Since $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_{11}, \Delta_{22}, \Delta_B$, the conclusion $\Gamma; \Delta \vdash M : T$ follows by an application of (EXP SUBSUM).

*Case* (RED LET VAL): assume $\mathsf{let}\ x = M\ \mathsf{in}\ E \to E\{M/x\}$ and $\Gamma; \Delta \vdash \mathsf{let}\ x = M\ \mathsf{in}\ E : T$. The typing judgement must follow by an instance of (EXP LET) after an instance of (EXP SUBSUM). Notice that $M \rightsquigarrow [\emptyset \mid M]$, hence it must be the case that $\Gamma; \Delta_A \vdash \mathsf{let}\ x = M\ \mathsf{in}\ E : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $\Gamma; \Delta_A \hookrightarrow \Gamma; \Delta_1, \Delta_2$
— $\Gamma; \Delta_1 \vdash M : U$
— $(\Gamma; \Delta_2) \bullet x : U \vdash E : V$
— $x \notin fv(V)$
Now notice that $x \notin dom(\Gamma)$ by Lemma B.5, hence $x \notin fv(\Delta_2)$ by Lemma B.10. By applying Lemma B.28 and noting that $x \notin fv(V)$, we then get $\Gamma; \Delta_1, \Delta_2 \vdash E\{M/x\} : V$. Since $\Gamma; \Delta \hookrightarrow \Gamma; (\Delta_1, \Delta_2), \Delta_B$, the conclusion $\Gamma; \Delta \vdash E\{M/x\} : T$ follows by an application of (EXP SUBSUM).

*Case* (RED LET): assume $\mathsf{let}\ x = E\ \mathsf{in}\ E'' \to \mathsf{let}\ x = E'\ \mathsf{in}\ E''$ with $E \to E'$ and $\Gamma; \Delta \vdash \mathsf{let}\ x = E\ \mathsf{in}\ E'' : T$. The typing judgement must follow by an instance of (EXP LET) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash \mathsf{let}\ x = E\ \mathsf{in}\ E'' : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $E \rightsquigarrow [\Delta' \mid D]$
— $\Gamma; \Delta_A, \Delta' \hookrightarrow \Gamma; \Delta_1, \Delta_2$
— $\Gamma; \Delta_1 \vdash D : U$
— $(\Gamma; \Delta_2) \bullet x : U \vdash E'' : V$
By Lemma B.42 we know that $E \to E'$ and $E \rightsquigarrow [\Delta' \mid D]$ imply that there exist $D', \Delta'', D'', D^*$ such that $D \to D'$ and $E' \rightsquigarrow [\Delta', \Delta'' \mid D'']$ with $D' \rightsquigarrow [\Delta'' \mid D^*]$ and $D^* \Rrightarrow D''$. Since Lemma B.42 is depth-preserving, we can apply the inductive hypothesis and get $\Gamma; \Delta_1 \vdash D' : U$. Given that $D' \rightsquigarrow [\Delta'' \mid D^*]$ and $\Gamma; \Delta_1 \vdash D' : U$, we get $\Gamma; \Delta_1, \Delta'' \vdash D^* : U$ by Lemma B.44. Since $D^* \Rrightarrow D''$ and $\Gamma; \Delta_1, \Delta'' \vdash D^* : U$, we get $\Gamma; \Delta_1, \Delta'' \vdash D'' : U$ by Lemma B.46. Hence, we have:
— $E' \rightsquigarrow [\Delta', \Delta'' \mid D'']$
— $\Gamma; \Delta_A, \Delta', \Delta'' \hookrightarrow \Gamma; (\Delta_1, \Delta''), \Delta_2$
— $\Gamma; \Delta_1, \Delta'' \vdash D'' : U$
— $(\Gamma; \Delta_2) \bullet x : U \vdash E'' : V$
We can then apply rule (EXP LET) to get $\Gamma; \Delta_A \vdash \mathsf{let}\ x = E'\ \mathsf{in}\ E'' : V$. The conclusion $\Gamma; \Delta \vdash \mathsf{let}\ x = E'\ \mathsf{in}\ E'' : T$ follows by (EXP SUBSUM).

*Case* (RED RES): assume $(\nu a)E \to (\nu a)E'$ with $E \to E'$ and $\Gamma; \Delta \vdash (\nu a)E : T$. The typing judgement must follow by an instance of (EXP RES) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash (\nu a)E : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $E \rightsquigarrow^a [\Delta' \mid D]$
— $\Gamma, a \updownarrow U; \Delta_A, \Delta' \vdash D : V$
By Lemma B.42 we know that $E \to E'$ and $E \rightsquigarrow^a [\Delta' \mid D]$ imply that there exist $D', \Delta'', D'', D^*$ such that $D \to D'$ and $E' \rightsquigarrow^a [\Delta', \Delta'' \mid D'']$ with $D' \rightsquigarrow^a [\Delta'' \mid D^*]$ and $D^* \Rrightarrow D''$. Since Lemma B.42 is depth-preserving, we can apply the inductive hypothesis and get $\Gamma, a \updownarrow U; \Delta_A, \Delta' \vdash D' : V$. Given that $D' \rightsquigarrow^a [\Delta'' \mid D^*]$ and

$\Gamma, a \updownarrow U; \Delta_A, \Delta' \vdash D' : V$, we get $\Gamma, a \updownarrow U; \Delta_A, \Delta', \Delta'' \vdash D^* : V$ by Lemma B.44. By Lemma B.46 we get $\Gamma, a \updownarrow U; \Delta_A, \Delta', \Delta'' \vdash D'' : V$. Hence, we have:
— $E' \rightsquigarrow^a [\Delta', \Delta'' \mid D'']$
— $\Gamma, a \updownarrow U; \Delta_A, \Delta', \Delta'' \vdash D'' : V$
We can then apply rule (EXP RES) to get $\Gamma; \Delta_A \vdash (\nu a)E' : V$. The conclusion $\Gamma; \Delta \vdash (\nu a)E' : T$ follows by (EXP SUBSUM).

*Case* (RED FORK 1): assume $E \upharpoonright E'' \rightarrow E' \upharpoonright E''$ with $E \rightarrow E'$ and $\Gamma; \Delta \vdash E \upharpoonright E'' : T$. The typing judgement must follow by an instance of (EXP FORK) after an instance of (EXP SUBSUM), hence it must be the case that $\Gamma; \Delta_A \vdash E \upharpoonright E'' : V$ and $\Gamma; \Delta_B \vdash V <: T$ with:
— $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_A, \Delta_B$
— $E \rightsquigarrow [\Delta' \mid D_1]$
— $E'' \rightsquigarrow [\Delta'' \mid D_2]$
— $\Gamma; \Delta_A, \Delta', \Delta'' \hookrightarrow \Gamma; \Delta_1, \Delta_2$
— $\Gamma; \Delta_1 \vdash D_1 : U$
— $\Gamma; \Delta_2 \vdash D_2 : V$
By Lemma B.42 we know that $E \rightarrow E'$ and $E \rightsquigarrow [\Delta' \mid D_1]$ imply that there exist $D_1', \Delta^*, D'', D^*$ such that $D_1 \rightarrow D_1'$ and $E' \rightsquigarrow [\Delta', \Delta^* \mid D'']$ with $D_1' \rightsquigarrow [\Delta^* \mid D^*]$ and $D^* \Rrightarrow D''$. Since Lemma B.42 is depth-preserving, we can apply the inductive hypothesis and get $\Gamma; \Delta_1 \vdash D_1' : U$. Given that $D_1' \rightsquigarrow [\Delta^* \mid D^*]$ and $\Gamma; \Delta_1 \vdash D_1' : U$, we get $\Gamma; \Delta_1, \Delta^* \vdash D^* : U$ by Lemma B.44. By Lemma B.46 we get $\Gamma; \Delta_1, \Delta^* \vdash D'' : U$. Hence, we have:
— $E' \rightsquigarrow [\Delta', \Delta^* \mid D'']$
— $E'' \rightsquigarrow [\Delta'' \mid D_2]$
— $\Gamma; \Delta_A, \Delta', \Delta^*, \Delta'' \hookrightarrow \Gamma; (\Delta_1, \Delta^*), \Delta_2$
— $\Gamma; \Delta_1, \Delta^* \vdash D'' : U$
— $\Gamma; \Delta_2 \vdash D_2 : V$
We can then apply rule (EXP FORK) to get $\Gamma; \Delta_A \vdash E' \upharpoonright E'' : V$. The conclusion $\Gamma; \Delta \vdash E' \upharpoonright E'' : T$ follows by (EXP SUBSUM).

*Case* (RED FORK 2): analogous to the previous case.

*Case* (RED HEAT): assume $E \rightarrow E'$ with $E \Rrightarrow D$, $D \rightarrow D'$ and $D' \rightarrow E'$. Assume further that $\Gamma; \Delta \vdash E : T$. By Lemma B.46 we have $\Gamma; \Delta \vdash D : T$. By inductive hypothesis $\Gamma; \Delta \vdash D' : T$, hence $\Gamma; \Delta \vdash E' : T$ again by Lemma B.46.

□

### B.8. Proof of (robust) safety

We first show that well-typed structures are statically safe.

LEMMA B.49 (STATIC SAFETY). *If* $\varepsilon; \emptyset \vdash \mathbf{S} : T$, *then* $\mathbf{S}$ *is statically safe.*

PROOF. Consider an arbitrary structure:

$$(\nu a_1) \ldots (\nu a_r)((E_1 \upharpoonright E_2) \upharpoonright E_3) \upharpoonright E_4,$$

where:

— $E_1 = \Pi_{i \in [1,m]} \mathsf{assume}\ F_i$,
— $E_2 = \Pi_{j \in [1,n]} \mathsf{assert}\ F_j'$,
— $E_3 = \Pi_{k \in [1,o]} c_k!M_k$, and
— $E_4 = \Pi_{\ell \in [1,p]} \mathcal{L}_\ell[e_\ell]$.

We need to show that $F_1, \ldots, F_m \vdash F_1' \otimes \ldots \otimes F_n'$.

We know that $\varepsilon; \emptyset \vdash \mathbf{S} : T$. This must have been derived by $r$ applications of (EXP RES) followed by three applications of (EXP FORK), possibly interleaving with multiple applications of (EXP SUBSUM). Note that each application of (EXP RES) and (EXP FORK) will make use of extraction, but by Lemma B.39 we can simplify an arbitrary chain of extraction steps with decreasing index sets $\{a_1, \ldots, a_r\}, \ldots, \{a_r\}, \emptyset$ into a single extraction step with index set $\emptyset$. Note that, by definition, extraction does not affect $E_2$, $E_3$, and $E_4$, since they do not contain assumptions, but extracts all the assumed formulas $F_i \neq \mathbf{1}$ from $E_1$. Also note that repeatedly extracting with the same index set $\emptyset$ does not yield any new result, as can be seen using Lemma B.40. By transitivity of subtyping and rewriting, using the previous facts, without loss of generality we have:

— $((E_1 \upharpoonright E_2) \upharpoonright E_3) \upharpoonright E_4 \rightsquigarrow^{\emptyset} [\Delta_1 \mid ((D_1 \upharpoonright E_2) \upharpoonright E_3) \upharpoonright E_4]$,
— where $E_1 \rightsquigarrow^{\emptyset} [\Delta_1 \mid D_1]$ with $\Delta_1 = \{F_i \mid F_i \neq \mathbf{1}\}$ and $D_1 = \Pi_{i \in [1,m]} \mathsf{assume}\ \mathbf{1}$.
— $\Gamma; \Delta_1 \hookrightarrow \Gamma; (\Delta_{A_1}, \Delta_{A_2}, \Delta_{A_3}, \Delta_{A_4}), \Delta_B$ with $\Gamma = a_1 \updownarrow T_1, \ldots, a_r \updownarrow T_r$
— $\Gamma; \Delta_{A_1} \vdash D_1 : U_1$ and $\Gamma; \Delta_{A_i} \vdash E_i : U_i$ for all $i \in \{2, 3, 4\}$
— $\Gamma; \Delta_B \vdash U_4 <: T$.

Hence, we know that $\Gamma; \Delta_{A_2} \vdash E_2 : U_2$, where $E_2$ is the parallel composition of the top-level assertions of $\mathbf{S}$. Such a typing derivation must contain $n-1$ applications of (EXP-FORK) and $n$ applications of (EXP ASSERT), possibly interleaved with multiple applications of (EXP SUBSUM). Again without loss of generality we have:

— $\Gamma; \Delta_{A_2} \hookrightarrow \Gamma; (\Delta_{C_1}, \ldots, \Delta_{C_n}), \Delta_D$
— for all $j \in \{1, \ldots, n\}$: $\Gamma; \Delta_{C_j} \vdash \mathsf{assert}\ F'_j : V_j$
— for all $j \in \{1, \ldots, n\}$: $\Gamma; \Delta_{C_j} \hookrightarrow \Gamma; \Delta'_{C_j}, \Delta''_{C_j}$ for some $\Delta'_{C_j}, \Delta''_{C_j}$ such that:
  — $\Gamma; \Delta'_{C_j} \vdash F'_j$, and
  — $\Gamma; \Delta''_{C_j} \vdash \mathsf{unit} <: V_j$
— $\Gamma; \Delta_D \vdash V_n <: U_2$.

By applying ($\otimes$-RIGHT) and rule (DERIVE), it follows that:

$$\Gamma; \Delta'_{C_1}, \ldots, \Delta'_{C_n} \vdash F'_1 \otimes \ldots \otimes F'_n.$$

Using Lemma B.8 we get $\Gamma; \Delta_1 \hookrightarrow \Gamma; \Delta'_{C_1}, \ldots, \Delta'_{C_n}$. By Lemma B.9 it follows that $\Gamma; \Delta_1 \vdash F'_1 \otimes \ldots \otimes F'_n$. Since $\Delta_1 = \{F_i \mid F_i \neq \mathbf{1}\}$, we get $\Gamma; F_1, \ldots, F_m \vdash F'_1 \otimes \ldots \otimes F'_n$ by Lemma B.7. By inverting rule (DERIVE) this implies:

$$F_1, \ldots, F_m \vdash F'_1 \otimes \ldots \otimes F'_n.$$

□

The safety theorem below states that any well-typed expression is safe. Its proof is simple and relies on the previous results.

RESTATEMENT 3 (OF THEOREM 6.1). *If $\varepsilon; \emptyset \vdash E : T$, then $E$ is safe.*

PROOF. In order to prove that $E$ is safe it suffices to show that, for all expressions $E'$ and structures $\mathbf{S}$ such that $E \rightarrow^* E'$ and $E' \Rrightarrow \mathbf{S}$, it holds that $\mathbf{S}$ is statically safe.

By Theorem B.48, $\varepsilon; \emptyset \vdash E : T$ implies $\varepsilon; \emptyset \vdash E' : T$. By Lemma B.46, $E' \Rrightarrow \mathbf{S}$ implies $\varepsilon; \emptyset \vdash \mathbf{S} : T$. We can conclude that $\mathbf{S}$ is statically safe by Lemma B.49. □

The next lemma is important to show that any opponent is trivially well-typed: it identifies $\mathsf{Un}$ with a number of structural types built around $\mathsf{Un}$ itself.

LEMMA B.50 (UNIVERSAL TYPE). *If $\Gamma; \emptyset \vdash \diamond$, then $\Gamma; \emptyset \vdash T <:> \mathsf{Un}$ for all $T \in \{\mathsf{unit}, x : \mathsf{Un} \rightarrow \mathsf{Un}, x : \mathsf{Un} * \mathsf{Un}, \mathsf{Un} + \mathsf{Un}, \mu\alpha.\, \mathsf{Un}\}$.*

PROOF. By inspection of the syntax-driven kinding rules it follows immediately that $\Gamma;\emptyset \vdash T :: k$ for all $T \in \{\mathsf{unit}, x : \mathsf{Un} \to \mathsf{Un}, x : \mathsf{Un} * \mathsf{Un}, \mathsf{Un} + \mathsf{Un}, \mu\alpha.\,\mathsf{Un}\}$ and $k \in \{\mathsf{pub}, \mathsf{tnt}\}$. We can then conclude by applying Lemma B.20. $\square$

We can now show that any opponent is well-typed. The statement is slightly more general than expected, since we appeal to inductive reasoning in the proof.

LEMMA B.51 (OPPONENT TYPABILITY). *Let $\Gamma;\emptyset \vdash \diamond$. Let $O$ be an expression that does not contain any assumption or assertion such that $(a \updownarrow \mathsf{Un}) \in \Gamma$ for each $a \in fn(O)$ and $(x : \mathsf{Un}) \in \Gamma$ for each $x \in fv(O)$, then $\Gamma;\emptyset \vdash O : \mathsf{Un}$.*

PROOF. By induction on the structure of $O$. In each case we apply the value/expression typing rule corresponding to the structure of $O$ (applying the induction hypothesis to the premises of the typing rule whenever needed). This allows us to derive that $\Gamma;\Delta \vdash O : T$ for some $T \in \{\mathsf{unit}, x : \mathsf{Un} \to \mathsf{Un}, x : \mathsf{Un} * \mathsf{Un}, \mathsf{Un} + \mathsf{Un}, \mu\alpha.\,\mathsf{Un}\}$ by using the following strategies:

— We first note that $O \rightsquigarrow^{\widetilde{a}} [\emptyset \mid O]$ for any $\widetilde{a}$, since by definition $O$ does not contain any assumption.
— In the case of typing a constructor $h \in \{\mathsf{inl}, \mathsf{inr}\}$, we choose the "free" type to be $\mathsf{Un}$.
— If $O$ is of the form $M = N$, we additionally apply (EXP SUBSUM) with subtyping rule (SUB REFINE).
— If $O$ is a split or a match operation, we appeal to Lemma B.7.
— We can easily switch between $T \in \{\mathsf{unit}, x : \mathsf{Un} \to \mathsf{Un}, x : \mathsf{Un} * \mathsf{Un}, \mathsf{Un} + \mathsf{Un}, \mu\alpha.\,\mathsf{Un}\}$ and $\mathsf{Un}$ by Lemma B.50, using (EXP SUBSUM) whenever needed.

We conclude by an application of (EXP SUBSUM), using Lemma B.50. $\square$

Finally, we can prove our main result of interest: if an expression $E$ is assigned type $\mathsf{Un}$ by our type system, then it is robustly safe. The proof is an easy consequence of Theorem 6.1 and Lemma B.51.

RESTATEMENT 4 (OF THEOREM 6.2). *If $\varepsilon;\emptyset \vdash E : \mathsf{Un}$, then $E$ is robustly safe.*

PROOF. Consider an arbitrary opponent $O$, we need to show that the application $O\ E$ is safe. Recall that:

$$O\ E \triangleq \mathsf{let}\ f = O\ \mathsf{in}\ \mathsf{let}\ x = E\ \mathsf{in}\ f\ x.$$

Let $\Gamma = a_1 \updownarrow \mathsf{Un}, \ldots, a_n \updownarrow \mathsf{Un}$ with $fn(O) = \{a_1, \ldots, a_n\}$. Since the opponent $O$ is closed by definition, by Lemma B.51 we know that $\Gamma;\emptyset \vdash O : \mathsf{Un}$. We can apply (EXP SUBSUM) and Lemma B.50 to derive:

$$\Gamma;\emptyset \vdash O : \mathsf{Un} \to \mathsf{Un}. \tag{1}$$

We can apply Lemma B.7 to $\varepsilon;\emptyset \vdash E : \mathsf{Un}$ and get $\Gamma;\emptyset \vdash E : \mathsf{Un}$. Assume now $E \rightsquigarrow [\Delta \mid D]$, by Lemma B.44 we have $\Gamma;\Delta \vdash D : \mathsf{Un}$. By Lemma B.7 we then get:

$$\Gamma, f : \mathsf{Un} \to \mathsf{Un}; \Delta \vdash D : \mathsf{Un}. \tag{2}$$

Since $O \rightsquigarrow^{\emptyset} [\emptyset \mid O]$, we can construct the following type derivation:

$$\frac{(1)\ \dfrac{\cdots}{\Gamma;\emptyset \vdash O : \mathsf{Un} \to \mathsf{Un}} \qquad \dfrac{(2)\ \dfrac{\cdots}{\Gamma, f : \mathsf{Un} \to \mathsf{Un}; \Delta \vdash D : \mathsf{Un}} \qquad \dfrac{\cdots}{\Gamma, f : \mathsf{Un} \to \mathsf{Un}, x : \mathsf{Un}; \emptyset \vdash f\ x : \mathsf{Un}}\ \textsc{Exp Appl}}{\Gamma, f : \mathsf{Un} \to \mathsf{Un}; \emptyset \vdash \mathsf{let}\ x = E\ \mathsf{in}\ f\ x : \mathsf{Un}}\ \textsc{Exp Let}}{\Gamma;\emptyset \vdash \mathsf{let}\ f = O\ \mathsf{in}\ \mathsf{let}\ x = E\ \mathsf{in}\ f\ x : \mathsf{Un}}\ \textsc{Exp Let}$$

Since $O\ E \rightsquigarrow^{\widetilde{b}} [\emptyset \mid O\ E]$ for all $\widetilde{b}$, we can get $\varepsilon;\emptyset \vdash (\nu a_1) \ldots (\nu a_n)(O\ E) : \mathsf{Un}$ by applying $n$ times rule (EXP RES) to the conclusion of the derivation above. By Theorem 6.1, we then know that $(\nu a_1)\ldots(\nu a_n)(O\ E)$ is safe. Since restrictions do not affect safety, we can conclude. $\square$

## C. SOUNDNESS AND COMPLETENESS OF ALGORITHMIC TYPING

In this section we prove the soundness (Theorem 10.1) and completeness (Theorem 10.2) of the algorithmic variant of our type system.

### C.1. Logical properties

We begin by showing some important properties of the logic that play a pivotal role in the bottom-up construction of the unique proof obligation in the algorithmic type system and the corresponding proofs of soundness and completeness.

We use the following convenient notation to denote all logical entailment rules that modify the set of premises.

*Definition* C.1 (*Left Rules* $\vdash^{\mathsf{L}}$). We say $\Delta \vdash^{\mathsf{L}} F$ if the last applied logical entailment rule is a rule of the form ($R$-LEFT) or (CONTR) or (WEAK).

LEMMA C.2 (IMPLICATION).

(*1*) *For all* $\Delta, F, F'$ *we have that* $\Delta \vdash F \multimap F'$ *iff* $\Delta, F \vdash F'$.
(*2*) *For all* $\Gamma, \Delta, F, F'$ *we have that* $\Gamma; \Delta \vdash F \multimap F'$ *iff* $\Gamma; \Delta, F \vdash F'$.

PROOF.

(1) We show both directions separately:
— $\Delta \vdash F \multimap F' \Rightarrow \Delta, F \vdash F'$:

We proceed by induction on the derivation of $\Delta \vdash F \multimap F'$. By inspection of the rules, we know that either $\Delta \vdash F \multimap F'$ by an application of ($\multimap$-RIGHT) or (IDENT) or (FALSE) or $\Delta \vdash^{\mathsf{L}} F \multimap F'$.

*Case* ($\multimap$-RIGHT): We know that $\Delta, F \vdash F'$ by the premise of the rule. We can immediately conclude.

*Case* (IDENT): In this case $\Delta = F \multimap F'$. Using ($\multimap$-LEFT) and (IDENT) we can immediately derive that

$$\frac{F \vdash F \qquad F' \vdash F'}{F \multimap F', F \vdash F'}.$$

*Case* (FALSE): In this case we know $\Delta = \mathbf{0}$. We can apply (FALSE) to derive that $\Delta \vdash F'$ and conclude by (WEAK).

*Case* $\Delta \vdash^{\mathsf{L}} F \multimap F'$ for some rule $R$:

By part (C.1) we know

$$\frac{\Delta' \vdash F \multimap F'}{\Delta \vdash^{\mathsf{L}} F \multimap F'}\ R$$

for some environment $\Delta'$. We apply the induction hypothesis and derive that $\Delta', F \vdash F'$.

We apply $R$ to derive that $\Delta, F \vdash F'$, and conclude.

— $\Gamma; \Delta, F \vdash F' \Rightarrow \Gamma; \Delta \vdash F \multimap F'$:

In this case we can immediately conclude by an application of ($\multimap$-RIGHT).

(2) Follows immediately by the definition of (DERIVE) and (FORM ENV ENTRY), property (1) and Lemma B.5.

$\square$

LEMMA C.3 (UNIVERSAL QUANTIFICATION). *It holds that:*

(*1*) *For all* $x, \Delta, F$ *such that* $x \notin fv(\Delta)$*, we have that* $\Delta \vdash F$ *iff* $\Delta \vdash \forall x.F$.
(*2*) *For all* $\Gamma, x, T, \Delta, F$ *such that* $\Gamma, x : \psi(T); \Delta \vdash \diamond$ *and* $x \notin fnfv(\Delta)$*, we have that* $\Gamma, x : \psi(T); \Delta \vdash F$ *iff* $\Gamma; \Delta \vdash \forall x.F$.

PROOF.

(1) We assume that $x \notin fv(\Delta)$. We show both directions separately:
— $\Delta \vdash F \Rightarrow \Delta \vdash \forall x.F$:
Since $x \notin fv(\Delta)$ we can immediately apply ($\forall$-RIGHT) and derive that

$$\Delta \vdash \forall x.F.$$

— $\Delta \vdash \forall x.F \Rightarrow \Delta \vdash F$:
We proceed by induction on the derivation of $\Delta \vdash \forall x.F$. By inspection of the rules, we know that either $\Delta \vdash \forall x.F$ by an application of ($\forall$-RIGHT) or (IDENT) or (FALSE) or $\Delta \vdash^{\mathsf{L}} \forall x.F$.
*Case* ($\forall$-RIGHT): By definition of the rule we know that $x \notin fv(\Delta)$ and $\Delta \vdash F$ and conclude.
*Case* (IDENT:) In this case $\Delta = \forall x.F$. We know that $\forall x.F \vdash F$ by ($\forall$-LEFT) and (IDENT).
*Case* (FALSE): In this case we know $\Delta = \mathbf{0}$. We can apply (FALSE), and derive $\Delta \vdash F$.
*Case* $\Delta \vdash^{\mathsf{L}} \forall x.F$ for some rule $R$:
By Definition C.1 we know that

$$\frac{\Delta' \vdash \forall x.F}{\Delta \vdash^{\mathsf{L}} \forall x.F.}\,R$$

for some $\Delta'$. We apply the induction hypothesis and derive that $\Delta' \vdash F$.
We can apply $R$ to derive that $\Delta \vdash F$.

(2) Follows immediately by the definition of (DERIVE) and (FORM ENV ENTRY) and *fnfv*, property (1) and Lemma B.5.

□

### C.2. Soundness and completeness of the algorithmic judgements

LEMMA C.4 (SOUNDNESS AND COMPLETENESS OF ALGORITHMIC WELL-FORMEDNESS). *For all* $\Gamma$*, the following holds true:*

(*1*) $\Gamma \vdash_{\mathsf{alg}} \diamond$ *iff* $\Gamma; \emptyset \vdash \diamond$
(*2*) *for all* $T$, $\Gamma \vdash_{\mathsf{alg}} T$ *iff* $\Gamma; \emptyset \vdash T$

PROOF. By induction on the derivation of the respective well-formedness statement. □

LEMMA C.5 (SOUNDNESS AND COMPLETENESS OF ALGORITHMIC KINDING). *For all* $\Gamma, T, k$*, the following holds true:*

(*1*) *for all* $F, \Delta$ *such that* $\Gamma \vdash_{\mathsf{alg}} T :: k; F$ *and* $\Gamma; \Delta \vdash F$*, we have that* $\Gamma; \Delta \vdash T :: k$.
(*2*) *for all* $\Delta$ *such that* $\Gamma; \Delta \vdash T :: k$*, there exists* $F$ *such that* $\Gamma \vdash_{\mathsf{alg}} T :: k; F$ *and* $\Gamma; \Delta \vdash F$*;*

PROOF.

(1) We first prove part (1). The proof proceeds by induction on the length of $\Gamma \vdash_{\mathsf{alg}} T :: k; F$. The base cases are (KIND VAR ALG) and (KIND UNIT ALG): they follow by an inspection of the typing rules and by Lemma C.4.
We now discuss the induction step.

*Case* (KIND FUN ALG): $\Gamma \vdash_{\mathsf{alg}} x : T \to U :: k; !F_1 \otimes !F_2$ is derived by $\Gamma \vdash_{\mathsf{alg}} T :: \overline{k}; F_1$ and $\Gamma, x : \psi(T) \vdash_{\mathsf{alg}} U :: k; F_2$. We also know that $\Gamma; \Delta \vdash !F_1 \otimes !F_2$.

*To show:* $\Gamma; \Delta \vdash x : T \to U :: k$. We know that that $\Gamma; \Delta \hookrightarrow \Gamma; !F_1, !F_2$ by (REWRITE) and (DERIVE) and $\Gamma; !F_1 \vdash F_1$, and $\Gamma; !F_2 \vdash F_2$ by (IDENT), (!-LEFT), and (DERIVE).

By induction hypothesis, $\Gamma; !F_1 \vdash T :: \overline{k}$ and $\Gamma, x : \psi(T); !F_2 \vdash U :: k$. The result follows from (KIND FUN).

*Case* (KIND PAIR ALG): The proof is analogous to the one for (KIND FUN ALG).

*Case* (KIND SUM ALG): The proof is analogous to the one for (KIND FUN ALG).

*Case* (KIND REC ALG): $\Gamma \vdash_{\mathsf{alg}} \mu\alpha.T :: k; !F$ is proved by $\Gamma, \alpha :: k \vdash_{\mathsf{alg}} T :: k; F$. We also know that $\Gamma; \Delta \vdash !F$.

*To show:* $\Gamma; \Delta \vdash \mu\alpha.T :: k$. By (REWRITE) and (DERIVE) and Lemma B.5, $\Gamma; \Delta \hookrightarrow \Gamma; !F$ and $\Gamma; !F \vdash F$ by (IDENT), (!-LEFT), and (DERIVE).

By induction hypothesis, $\Gamma, \alpha :: k; !F \vdash T :: k$. The result follows from (KIND REC).

*Case* (KIND REFINE PUBLIC ALG): The proof follows immediately from the induction hypothesis.

*Case* (KIND REFINE TAINTED ALG): $\Gamma \vdash_{\mathsf{alg}} T :: \mathsf{tnt}; (\forall x.\mathit{forms}(x : T)) \otimes F'$, where $T$ is refined is proved by $\Gamma \vdash_{\mathsf{alg}} \psi(T) :: \mathsf{tnt}; F'$, $\Gamma, x : \psi(T) \vdash_{\mathsf{alg}} \diamond$, and $\Gamma \vdash_{\mathsf{alg}} T$. We also know that $\Gamma; \Delta \vdash (\forall x.\mathit{forms}(x : T)) \otimes F'$.

*To show:* $\Gamma; \Delta \vdash T :: \mathsf{tnt}$. By (REWRITE), (DERIVE), and Lemma B.5 we know that that $\Gamma; \Delta \hookrightarrow \Gamma; (\forall x.\mathit{forms}(x : T)), F'$ and $\Gamma; (\forall x.\mathit{forms}(x : T)) \vdash (\forall x.\mathit{forms}(x : T))$ and $\Gamma; F' \vdash F'$ by (IDENT) and (DERIVE).

By induction hypothesis, $\Gamma; F' \vdash_{\mathsf{alg}} \psi(T) :: \mathsf{tnt}$. By ($\forall$-LEFT), (IDENT), and (DERIVE), $\Gamma, x : \psi(T); (\forall x.\mathit{forms}(x : T)) \vdash \mathit{forms}(x : T)$. The result follows from (KIND REFINE TAINTED).

(2) We now prove part (2). The proof proceeds by induction on the length of $\Gamma; \Delta \vdash T :: k$. The base cases are (KIND VAR) and (KIND UNIT): they follow by an inspection of the typing rules, by Lemma C.4, and by observing that $\Gamma; \Delta \vdash \mathbf{1}$ for any $\Delta$ such that $\Gamma; \Delta \vdash \diamond$. We now discuss the induction step.

*Case* (KIND FUN): $\Gamma; \Delta \vdash x : T \to U :: k$ is proved by $\Gamma; !\Delta_1 \vdash T :: \overline{k}$, $\Gamma, x : \psi(T); !\Delta_2 \vdash U :: k$, and $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2$. By induction hypothesis, there exist $F_1, F_2$ such that $\Gamma \vdash_{\mathsf{alg}} T :: \overline{k}; F_1$, $\Gamma, x : \psi(T) \vdash_{\mathsf{alg}} U :: k; F_2$, $\Gamma; !\Delta_1 \vdash F_1$, and $\Gamma; !\Delta_2 \vdash F_2$.

*To show:* $\Gamma \vdash_{\mathsf{alg}} x : T \to U :: k; !F_1 \otimes !F_2$. The result follows from (KIND FUN ALG).

*To show:* $\Gamma; \Delta \vdash !F_1 \otimes !F_2$. We use (DERIVE) as needed. By (!-RIGHT) we can derive that $\Gamma; !\Delta_1 \vdash !F_1$ and $\Gamma; !\Delta_2 \vdash !F_2$. By ($\otimes$-RIGHT), $\Gamma; !\Delta_1, !\Delta_2 \vdash !F_1 \otimes !F_2$. The result follows from Lemma B.9.

*Case* (KIND PAIR): The proof is analogous to the one for (KIND FUN).

*Case* (KIND SUM): The proof is analogous to the one for (KIND FUN).

*Case* (KIND REC): $\Gamma; \Delta \vdash \mu\alpha.T :: k$ is proved by $\Gamma, \alpha :: k; !\Delta' \vdash T :: k$ and $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$. By induction hypothesis, there exists $F$ such that $\Gamma, \alpha :: k \vdash_{\mathsf{alg}} T :: k; F$ and $\Gamma; !\Delta' \vdash F$.

*To show:* $\Gamma \vdash_{\mathsf{alg}} \mu\alpha.T :: k; !F$. The result follows from (KIND REC ALG).

*To show:* $\Gamma; \Delta \vdash !F$. Using (DERIVE) and (!-RIGHT) we can derive that $\Gamma; !\Delta' \vdash !F$. The result follows from Lemma B.9.

*Case* (KIND REFINE PUBLIC): The proof follows immediately from the induction hypothesis.

*Case* (KIND REFINE TAINTED): $\Gamma; \Delta \vdash T :: \mathsf{tnt}$, where $T$ is refined is proved by $\Gamma; \Delta_1 \vdash \psi(T) :: \mathsf{tnt}$, $\Gamma, x : \psi(T); \Delta_2 \vdash \mathit{forms}(x : T)$, and $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$. By induction hypothesis, there exists $F'$ such that $\Gamma \vdash_{\mathsf{alg}} \psi(T) :: \mathsf{tnt}; F'$ and $\Gamma; \Delta_1 \vdash F'$.

**Table XXVI** Intermediate Subtyping $<:_{\mathsf{alg}}$

(SUB REFL *)
$$\frac{\Gamma;\Delta \vdash T \qquad T \in \{\mathsf{unit},\alpha\}}{\Gamma;\Delta \vdash T <:_{\mathsf{alg}} T}$$

(SUB PUB TNT *)
$$\frac{\Gamma;\Delta_1 \vdash T :: \mathsf{pub} \qquad \Gamma;\Delta_2 \vdash U :: \mathsf{tnt} \qquad \Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2 \qquad T \neq_{\top} U}{\Gamma;\Delta \vdash T <:_{\mathsf{alg}} U}$$

(SUB FUN *)
$$\frac{\Gamma;!\Delta_1 \vdash \overline{T'} <:_{\mathsf{alg}} \overline{T} \quad \Gamma, x:\psi(T');!\Delta_2 \vdash \overline{U} <:_{\mathsf{alg}} \overline{U'} \quad \Gamma;\Delta \hookrightarrow \Gamma;!\Delta_1,!\Delta_2}{\Gamma;\Delta \vdash x:\overline{T} \to \overline{U} <:_{\mathsf{alg}} x:\overline{T'} \to \overline{U'}}$$

(SUB PAIR *)
$$\frac{\Gamma;!\Delta_1 \vdash \overline{T} <:_{\mathsf{alg}} \overline{T'} \quad \Gamma, x:\psi(T);!\Delta_2 \vdash \overline{U} <:_{\mathsf{alg}} \overline{U'} \quad \Gamma;\Delta \hookrightarrow \Gamma;!\Delta_1,!\Delta_2}{\Gamma;\Delta \vdash x:\overline{T} * \overline{U} <:_{\mathsf{alg}} x:\overline{T'} * \overline{U'}}$$

(SUB SUM *)
$$\frac{\Gamma;!\Delta_1 \vdash \overline{T} <:_{\mathsf{alg}} \overline{T'} \qquad \Gamma;!\Delta_2 \vdash \overline{U} <:_{\mathsf{alg}} \overline{U'} \qquad \Gamma;\Delta \hookrightarrow \Gamma;!\Delta_1,!\Delta_2}{\Gamma;\Delta \vdash \overline{T} + \overline{U} <:_{\mathsf{alg}} \overline{T'} + \overline{U'}}$$

(SUB POS REC *)
$$\frac{\Gamma,\alpha;!\Delta' \vdash \overline{T} <:_{\mathsf{alg}} \overline{T'} \quad \alpha \text{ occurs only positively in } \overline{T} \text{ and } \overline{T'} \quad T \neq T' \quad \Gamma;\Delta \hookrightarrow \Gamma;!\Delta'}{\Gamma;\Delta \vdash \mu\alpha.\overline{T} <:_{\mathsf{alg}} \mu\alpha.\overline{T'}}$$

(SUB REFL REC *)
$$\frac{\Gamma;\Delta \vdash \mu\alpha.T}{\Gamma;\Delta \vdash \mu\alpha.T <:_{\mathsf{alg}} \mu\alpha.T}$$

(SUB PUB TNT REC *)
$$\frac{\Gamma;\Delta_1 \vdash \mu\alpha.T :: \mathsf{pub} \quad \Gamma;\Delta_2 \vdash \mu\alpha.U :: \mathsf{tnt} \quad s = \mathsf{SPT} \oplus s' = \mathsf{SPT} \quad \Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2}{\Gamma;\Delta \vdash (\mu\alpha.T)_s <:_{\mathsf{alg}} (\mu\alpha.U)_{s'}}$$

(SUB REFINE *)
$$\frac{\Gamma;\Delta_1 \vdash \psi(\overline{T}) <:_{\mathsf{alg}} \psi(\overline{U}) \quad \Gamma, y:\psi(T);\Delta_2, \mathit{forms}(y:T) \vdash \mathit{forms}(y:U) \quad \Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2 \quad \overline{T} \text{ and/or } \overline{U} \text{ refined}}{\Gamma;\Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}}$$

**Notation:** We use $T$ to denote the non-annotated counterpart $\langle\overline{T}\rangle$ of the annotated type $\overline{T}$.

*To show:* $\Gamma \vdash_{\mathsf{alg}} T :: \mathsf{tnt}; (\forall x.\mathit{forms}(x:T)) \otimes F'$. The result follows from (KIND REFINE TAINTED ALG) and Lemma B.5.
*To show:* $\Gamma;\Delta \vdash (\forall x.\mathit{forms}(x:T)) \otimes F'$. By Lemma B.5, $\Gamma, x:\psi(T);\Delta_2 \vdash \diamond$ and, thus, $x \notin \mathit{dom}(\Gamma)$. Since $\Gamma;\Delta \hookrightarrow \Gamma;\Delta_1,\Delta_2$, we also have that $\Gamma;\Delta_1,\Delta_2 \vdash \diamond$ and, thus, $x \notin \mathit{fnfv}(\Delta_2) \subseteq \mathit{dom}(\Gamma)$. By Lemma C.3, $\Gamma;\Delta_2 \vdash \forall x.\mathit{forms}(x:T)$. By ($\otimes$-RIGHT), $\Gamma;\Delta_1,\Delta_2 \vdash (\forall x.\mathit{forms}(x:T)) \otimes F'$. The result follows from Lemma B.9.

□

In order to prove soundness and completeness of subtyping we will proceed in two steps: we first introduce an intermediate algorithmic variant of the standard subtyping relation $\Gamma;\Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}$ for annotated types that extends the standard one by adding the side conditions in (SUB REFL) and (SUB PUB TNT) present in algorithmic subtyping and provides three disjoint rules for subtyping two iso-recursive types, following the insights given in Section 10.5.

We will then show that we can find annotations to prove the standard subtyping and intermediate subtyping equivalent and show the soundness and completeness of algorithmic subtyping with respect to intermediate subtyping $<:_{\mathsf{alg}}$.

The full definition of the intermediate subtyping rules can be found in Table XXVI. The rules make use of the previously introduced annotated types $\overline{T}$ that might contain type annotation SPT. As in algorithmic subtyping, we assume the function $\psi$ to extend to annotated types and we write $T = \langle\overline{T}\rangle$ to denote the type that results from erasing all type annotations from $\overline{T}$. We say $T$ and $\overline{T}$ are equal up to type annotations.

The soundness and completeness proof of intermediate subtyping makes use of the following propositions and lemmas. We write $\overline{T} = \{x_m : \ldots\{x_2 : \{x_1 : \overline{U} \mid F_1\} \mid F_2\} \ldots \mid F_m\}$ to denote nested (annotated) refinement types, for $m = 0$ this notation simply denotes the annotated type $\overline{U}$.

The following proposition states that all types are also annotated types by construction, which we will use implicitly throughout the following proofs.

PROPOSITION C.6 (TYPES AND ANNOTATED TYPES). *Let $T$ be a type. Then $T$ is also an annotated type, such that $\langle T \rangle = T$.*

LEMMA C.7 (REFINEMENT ERASURE OF ANNOTATED TYPES). *For all types $T$ and annotated types $\overline{T}$ such that $T = \langle \overline{T} \rangle$ it holds that $\psi(T) = \psi(\langle \overline{T} \rangle) = \langle \psi(\overline{T}) \rangle$.*

PROOF. Proof by induction on the structure of $\overline{T}$ using the definitions of $\psi$ and the erasure of annotations $\langle \bullet \rangle$. □

LEMMA C.8 (NESTED REFINEMENTS). *For all types $T$ it holds that there exist $m \geq 0$ and $T_{min}$ and $F_1, \ldots, F_m, x_1, \ldots, x_m$ (if $m > 0$) such that $T = \{x_m : \ldots\{x_2 : \{x_1 : T_{min} \mid F_1\} \mid F_2\} \ldots \mid F_m\}$ and $\psi(T) = \psi(T_{min}) = T_{min}$.*

PROOF. Proof by induction on the structure of $T$ using the definition of $\psi$. □

LEMMA C.9 (ANNOTATED REFINEMENT TYPES). *For all types $T = \{x_m : \ldots\{x_2 : \{x_1 : T_{min} \mid F_1\} \mid F_2\} \ldots \mid F_m\}$, where $\psi(T) = \psi(T_{min}) = T_{min}$, and all annotated types $\overline{T_{min}}$ and $\overline{T} = \{x_m : \ldots\{x_2 : \{x_1 : \overline{T_{min}} \mid F_1\} \mid F_2\} \ldots \mid F_m\}$ such that $T_{min} = \langle \overline{T_{min}} \rangle$ it holds that*

— $\langle \overline{T} \rangle = T$ *and*
— $\psi(\overline{T}) = \overline{T_{min}}$.

PROOF. The first statement follows by induction on $m$ using the definition of $\langle \bullet \rangle$. For the second statement we first note that $T_{min} = \psi(T_{min})$. By the definition of $\langle \bullet \rangle$ we know that $T_{min}$ and $\overline{T_{min}}$ are equal up to typing annotations and thus by definition of $\psi$ it must be the case that $\overline{T_{min}} = \psi(\overline{T_{min}})$. We conclude by induction on $m$, using the definition of $\psi$. □

LEMMA C.10 (SOUNDNESS AND COMPLETENESS OF INTERMEDIATE SUBTYPING). *For all $\Gamma, \Delta, T, U$ it holds that:*

(*1*) *If $\Gamma; \Delta \vdash T$ then $\Gamma; \Delta \vdash T <:_{\mathsf{alg}} T$.*
(*2*) *If there exist $\Delta_1, \Delta_2$ such that $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ and $\Gamma; \Delta_1 \vdash T :: \mathsf{pub}$ and $\Gamma; \Delta_2 \vdash U :: \mathsf{tnt}$ then there exist annotated types $\overline{T}$ and $\overline{U}$ such that $T = \langle \overline{T} \rangle$, $U = \langle \overline{U} \rangle$, and $\Gamma; \Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}$.*
(*3*) *If $\Gamma; \Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}$ and $T = \langle \overline{T} \rangle$, $U = \langle \overline{U} \rangle$, then $\Gamma; \Delta \vdash T <: U$.*
(*4*) *If $\Gamma; \Delta \vdash T <: U$ then there exist $\overline{T}, \overline{U}$ such that $T = \langle \overline{T} \rangle$, $U = \langle \overline{U} \rangle$, and $\Gamma; \Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}$.*

PROOF. Throughout the proof we make us of Proposition C.6 that allows us to consider each type $T$ as an annotated type such that $\langle T \rangle = T$.

(1) Proof by induction on the structure of $T$.
If $T \in \{\alpha, \mathsf{unit}\}$ we can immediately conclude by an application of (SUB REFL *).
If T is an iso-recursive type we can immediately conclude by an application of (SUB REFL REC *).
If $T$ is a pair, sum, or function type the proofs follow the same strategy, which we show exemplarily for $T = x : T_1 * T_2$. We know that $\Gamma; \Delta \hookrightarrow \Gamma; \emptyset, \emptyset$ by Lemma B.8.

Using Lemma B.5 we apply the induction hypothesis to $T_1$ to derive that $\Gamma; \emptyset \vdash T_1 <:_{\mathsf{alg}} T_1$ and to $T_2$ to derive that $\Gamma, x : \psi(T_1); \emptyset \vdash T_2 <:_{\mathsf{alg}} T_2$. We conclude by an application of (SUB PAIR *).
If $T = \{x : U \mid F\}$ we can apply the induction hypothesis and Lemma B.5 to derive that $\Gamma; \emptyset \vdash \psi(T) <:_{\mathsf{alg}} \psi(T)$. It is easy to see that $\Gamma, y : \psi(T); \mathit{forms}(y : T) \vdash \mathit{forms}(y : T)$. Since we know that $\Gamma; \Delta \hookrightarrow \Gamma; \emptyset$ by Lemma B.8 we can conclude by an application of (SUB REFINE *).

(2) Proof by induction on the structure of $T$. Note that whenever it is the case that $T \neq_{\top} U$ no annotations are necessary and we can immediately conclude by an application of (SUB PUB TNT *). In the following we thus assume that $T$ and $U$ share the same top-level constructor, or that one of them is refined.

*Case* In the case where $T = U$ and $T \in \{\alpha, \mathsf{unit}\}$ we can immediately conclude by applying (SUB REFL *) to $\overline{T} := T$ and $\overline{U} := U$ using Lemma B.5.

*Case* In the case where $T$ is of the form $\mu\alpha. T_1$ and $U$ is of the form $\mu\alpha. U_1$ for some $T_1, U_1$ we can immediately conclude by annotating $U$ with SPT such that we can apply (SUB PUB TNT REC *) to $\overline{T} := T$ and $\overline{U} := U_{\mathsf{SPT}}$. We note that $\langle U_{\mathsf{SPT}} \rangle = U$ by definition.

*Case* The proofs of the cases where $T, U$ are a couple of pair, sum, or function types all follow the same strategy, which we show exemplarily for the case $T = x : T_1 * T_2$ and $U = x : U_1 * U_2$ (note that the reasoning for sum types will be somewhat simplified).
We know that $\Gamma; \Delta_1 \vdash x : T_1 * T_2 :: \mathsf{pub}$ which by definition of the only applicable kinding rule (KIND PAIR) implies that there must exist $\Delta_{11}, \Delta_{12}$ such that $\Gamma; \Delta_1 \hookrightarrow \Gamma; !\Delta_{11}, !\Delta_{12}$ and $\Gamma; !\Delta_{11} \vdash T_1 :: \mathsf{pub}$ and $\Gamma, x : \psi(T_1); !\Delta_{12} \vdash T_2 :: \mathsf{pub}$.
Using the same reasoning, we know that $\Gamma; \Delta_2 \vdash x : U_1 * U_2 :: \mathsf{tnt}$ by definition of the only applicable kinding rule (KIND PAIR) implies that there must exist $\Delta_{21}, \Delta_{22}$ such that $\Gamma; \Delta_2 \hookrightarrow \Gamma; !\Delta_{21}, !\Delta_{22}$ and $\Gamma; !\Delta_{21} \vdash U_1 :: \mathsf{tnt}$ and $\Gamma, x : \psi(U_1); !\Delta_{22} \vdash U_2 :: \mathsf{tnt}$.
Applying the induction hypothesis to $\Gamma; !\Delta_{11} \vdash T_1 :: \mathsf{pub}$ and $\Gamma; !\Delta_{21} \vdash U_1 :: \mathsf{tnt}$ (implicitly using Lemma B.8) lets us derive that there exist annotated types $\overline{T_1}, \overline{U_1}$ such that

$$\Gamma; !\Delta_{11}, !\Delta_{21} \vdash \overline{T_1} <:_{\mathsf{alg}} \overline{U_1},$$

where $T_1 = \langle \overline{T_1} \rangle$ and $U_1 = \langle \overline{U_1} \rangle$.
By Lemma B.16 we know that $\Gamma, x : \psi(U_1); !\Delta_{22} \vdash U_2 :: \mathsf{tnt}$ implies $\Gamma, x : \psi(T_1); !\Delta_{22} \vdash U_2 :: \mathsf{tnt}$. We apply the induction hypothesis to the latter statement and $\Gamma, x : \psi(T_1); !\Delta_{12} \vdash T_2 :: \mathsf{pub}$, allowing us to derive that there exist annotated types $\overline{T_2}, \overline{U_2}$ such that

$$\Gamma, x : \psi(T_1); !\Delta_{12}, !\Delta_{22} \vdash \overline{T_2} <:_{\mathsf{alg}} \overline{U_2},$$

where $T_2 = \langle \overline{T_2} \rangle$ and $U_2 = \langle \overline{U_2} \rangle$.
By Lemma B.8 we know that

$$\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_{11}, !\Delta_{21}, !\Delta_{12}, !\Delta_{22}$$

and can thus conclude by an application of (SUB PAIR *), for $\overline{T} := x : \overline{T_1} * \overline{T_2}$ and $\overline{U} := x : \overline{U_1} * \overline{U_2}$. By definition of $\langle \bullet \rangle$ we know that $\langle \overline{T} \rangle = x : \langle \overline{T_1} \rangle * \langle \overline{T_2} \rangle = x : T_1 * T_2 = T$ and analogously $\langle \overline{U} \rangle = U$.

*Case* In the case that $T$ or $U$ (or both) are refined we proceed as follows. First we note that by Lemma C.8 there must thus exist $m \geq 0, n \geq 0$ (where $m > 0$ or $n > 0$ or both) and $T_{min}, U_{min}$ and $F_1, \ldots, F_m, x_1, \ldots, x_m$ and $F'_1, \ldots, F'_n, x'_1, \ldots, x'_n$ such that $T = \{x_m : \ldots \{x_2 : \{x_1 : T_{min} \mid F_1\} \mid F_2\} \ldots \mid F_m\}$ and $U = \{x'_n : \ldots \{x'_2 : \{x'_1 : U_{min} \mid F'_1\} \mid F'_2\} \ldots \mid F'_n\}$ and $\psi(T) = \psi(T_{min}) = T_{min}$ and $\psi(U) = \psi(U_{min}) = U_{min}$.

We know that $\Gamma; \Delta_1 \vdash T :: \mathsf{pub}$, which by Lemma B.13 implies that there exist $\Delta_{11}, \Delta_{12}$ such that $\Gamma; \Delta_1 \hookrightarrow \Gamma; !\Delta_{11}, \Delta_{12}$ and $\Gamma; !\Delta_{11} \vdash \psi(T) :: \mathsf{pub}$.
Furthermore, by Lemma B.13 we know that $\Gamma; \Delta_2 \vdash U :: \mathsf{tnt}$ implies that there exist $\Delta_{21}, \Delta_{22}$ such that $\Gamma; \Delta_2 \hookrightarrow \Gamma; !\Delta_{21}, \Delta_{22}$ and $\Gamma; !\Delta_{21} \vdash \psi(U) :: \mathsf{tnt}$ and $\Delta_{22} \vdash \mathit{forms}(y : U)$ for some $y \notin \mathit{fv}(\Gamma)$.
We can apply the induction hypothesis to $\Gamma; !\Delta_{11} \vdash \psi(T) :: \mathsf{pub}$ (which is equivalent to $\Gamma; !\Delta_{11} \vdash T_{min} :: \mathsf{pub}$) and $\Gamma; !\Delta_{21} \vdash \psi(U) :: \mathsf{tnt}$ (which is equivalent to $\Gamma; !\Delta_{21} \vdash U_{min} :: \mathsf{tnt}$) to derive that there exist $\overline{T_{\mathsf{min}}}, \overline{U_{\mathsf{min}}}$ such that

$$\Gamma; !\Delta_{11}, !\Delta_{21} \vdash \overline{T_{\mathsf{min}}} <:_{\mathsf{alg}} \overline{U_{\mathsf{min}}},$$

where $\psi(T) = T_{\mathsf{min}} = \langle \overline{T_{\mathsf{min}}} \rangle$ and $\psi(U) = U_{\mathsf{min}} = \langle \overline{U_{\mathsf{min}}} \rangle$.
We construct the annotated types $\overline{T}$ and $\overline{U}$ as follows: $\overline{T} := \{x_m : \ldots \{x_2 : \{x_1 : \overline{T_{min}} \mid F_1\} \mid F_2\} \ldots \mid F_m\}$ and $\overline{U} := \{x'_n : \ldots \{x'_2 : \{x'_1 : \overline{U_{min}} \mid F'_1\} \mid F'_2\} \ldots \mid F'_n\}$. By Lemma C.9 we know that $T = \langle \overline{T} \rangle$ and $U = \langle \overline{U} \rangle$ and $\psi(\overline{T}) = \psi(\overline{T_{min}}) = \overline{T_{min}}$ and $\psi(\overline{U}) = \psi(\overline{U_{min}}) = \overline{U_{min}}$.
Weakening in the logic allows us to derive that $\Delta_{22} \vdash \mathit{forms}(y : U)$ implies that $\Delta_{22}, \mathit{forms}(y : T) \vdash \mathit{forms}(y : U)$. By applying (DERIVE) and Lemma B.5 we can conclude that

$$\Gamma, y : \psi(T); \Delta_{22}, \mathit{forms}(y : T) \vdash \mathit{forms}(y : U).$$

We use Lemma B.8 to derive that

$$\Gamma; \Delta \hookrightarrow \Gamma; (!\Delta_{11}, !\Delta_{21}), \Delta_{22}$$

and conclude that $\Gamma; \Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}$ by an application of (SUB REFINE *).

(3) Straightforward induction on the derivation of $\Gamma; \Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}$.
In the case where the last applied rule was SUB REFL * we know that $\overline{T} = \overline{U} = T = U$ and $T \in \{\alpha, \mathsf{unit}\}$ and $\Gamma; \Delta \vdash T$, which allows us to conclude by an application of SUB REFL.
In the case where the last applied rule was (SUB PUB TNT *) we know that $T = \overline{T}$ and $U = \overline{U}$ and $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$ for some $\Delta_1, \Delta_2$ such that $\Gamma; \Delta_1 \vdash T :: \mathsf{pub}$ and $\Gamma; \Delta_2 \vdash U :: \mathsf{tnt}$. We can immediately conclude by an application of (SUB PUB TNT).
In the case where the last applied rule (SUB REFL REC *) we match the necessary premise to conclude with an application of (SUB REFL), in the case of (SUB PUB TNT REC *) we can conclude with an application of (SUB PUB TNT).
In all other cases $R$ * we apply the induction hypothesis to the premises of $R$ * and conclude by an application of $R$. Note that for (SUB REFINE *) we make use of Lemma C.7, which lets us deduce that $\psi(T) = \langle \psi(\overline{T}) \rangle$ and $\psi(U) = \langle \psi(\overline{U}) \rangle$.

(4) Proof by induction on the derivation of $\Gamma; \Delta \vdash T <: U$. We distinguish upon the last applied subtyping rule $R$:

*Case* (SUB REFL): In this case we can immediately conclude by an application of statement (1) of this lemma, choosing $\overline{T} := T$ and $\overline{U} := U$.

*Case* (SUB PUB TNT): In this case we can immediately conclude by an application of statement (2) of this lemma.

*Case* (SUB POS REC) and $T = U$: In this case we can immediately conclude by an application of (SUB REFL REC *), choosing $\overline{T} := T$ and $\overline{U} := U$.

*Case* (SUB POS REC) and $T \neq U$: In this case we know that $T = \mu\alpha. T_1$ and $U = \mu\alpha. U_1$. We apply the induction hypothesis to the subtyping premise of the rule to derive that $\Gamma, \alpha; !\Delta' \vdash \overline{T_1} <:_{\mathsf{alg}} \overline{U_1}$ for some $\overline{T_1}, \overline{U_1}$ such that $T_1 = \langle \overline{T_1} \rangle$ and $U_1 = \langle \overline{U_1} \rangle$ and $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$. We conclude by an application of (SUB POS REC *), choosing $\overline{T} := \mu\alpha. \overline{T_1}$ and $\overline{U} := \mu\alpha. \overline{U_1}$. Note that by definition of $\langle \bullet \rangle$ we know that $\langle \overline{T} \rangle = \mu\alpha. \langle \overline{T_1} \rangle = \mu\alpha. T_1 = T$ and analogously $\langle \overline{U} \rangle = U$.

*Case* $R$ is (SUB PAIR), (SUB FUN), (SUB SUM): These cases follow similarly to the previous case of distinct iso-recursive types by applying the induction hypothesis to the subtyping premises of the rule and concluding by an application of $R$ *.

*Case* (SUB REFINE): In this case we know that $T$ or $U$ (or both) are refined. First we note that by Lemma C.8 there exist $m \geq 0, n \geq 0$ (where $m > 0$ or $n > 0$ or both) and $T_{min}, U_{min}$ and $F_1, \ldots, F_m, x_1, \ldots, x_m$ and $F'_1, \ldots, F'_n, x'_1, \ldots, x'_n$ such that $T = \{x_m : \ldots \{x_2 : \{x_1 : T_{min} \mid F_1\} \mid F_2\} \ldots \mid F_m\}$ and $U = \{x'_n : \ldots \{x'_2 : \{x'_1 : U_{min} \mid F'_1\} \mid F'_2\} \ldots \mid F'_n\}$ and $\psi(T) = \psi(T_{min}) = T_{min}$ and $\psi(U) = \psi(U_{min}) = U_{min}$.
By the definition of (SUB REFINE) we know that there exist $\Delta_1, \Delta_2$ such that $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, \Delta_2$ and $\Gamma; \Delta_1 \vdash \psi(T) <: \psi(U)$ and $\Gamma, x : \psi(T); \Delta_2, \mathit{forms}(y : T) \vdash \mathit{forms}(y : U)$.
We can apply the induction hypothesis to $\Gamma; \Delta_1 \vdash \psi(T) <: \psi(U)$ (which is equivalent to $\Gamma; \Delta_1 \vdash T_{min} <: U_{min}$) to derive that there exist $\overline{T_{\mathsf{min}}}, \overline{U_{\mathsf{min}}}$ such that

$$\Gamma; \Delta_1 \vdash \overline{T_{\mathsf{min}}} <:_{\mathsf{alg}} \overline{U_{\mathsf{min}}},$$

where $\psi(T) = T_{\mathsf{min}} = \langle \overline{T_{\mathsf{min}}} \rangle$ and $\psi(U) = U_{\mathsf{min}} = \langle \overline{U_{\mathsf{min}}} \rangle$.
We construct the annotated types $\overline{T}$ and $\overline{U}$ as follows: $\overline{T} := \{x_m : \ldots \{x_2 : \{x_1 : \overline{T_{min}} \mid F_1\} \mid F_2\} \ldots \mid F_m\}$ and $\overline{U} := \{x'_n : \ldots \{x'_2 : \{x'_1 : \overline{U_{min}} \mid F'_1\} \mid F'_2\} \ldots \mid F'_n\}$. By Lemma C.9 we know that $T = \langle \overline{T} \rangle$ and $U = \langle \overline{U} \rangle$ and $\psi(\overline{T}) = \psi(\overline{T_{min}}) = \overline{T_{min}}$ and $\psi(\overline{U}) = \psi(\overline{U_{min}}) = \overline{U_{min}}$.
We conclude that $\Gamma; \Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}$ by an application of (SUB REFINE *).

□

LEMMA C.11 (SOUNDNESS AND COMPLETENESS OF ALGORITHMIC SUBTYPING).

(*1*) *For all* $\Gamma, \Delta, \overline{T}, \overline{U}$*, the following holds true:*
  (*a*) *For all* $F$ *such that* $\Gamma \vdash_{\mathsf{alg}} \overline{T} <: \overline{U}; F$ *and* $\Gamma; \Delta \vdash F$*, we have that* $\Gamma; \Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}$.
  (*b*) *If* $\Gamma; \Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}$*, then there exists* $F$ *such that* $\Gamma \vdash_{\mathsf{alg}} \overline{T} <: \overline{U}; F$ *and* $\Gamma; \Delta \vdash F$*;*
(*2*) *For all* $\Gamma, \Delta, T, U$*, the following holds true:*
  (*a*) *For all* $\overline{T}, \overline{U}, F$ *such that* $\Gamma \vdash_{\mathsf{alg}} \overline{T} <: \overline{U}; F$ *and* $\Gamma; \Delta \vdash F$ *and* $T = \langle \overline{T} \rangle$*,* $U = \langle \overline{U} \rangle$*, we have that* $\Gamma; \Delta \vdash T <: U$.
  (*b*) *If* $\Gamma; \Delta \vdash T <: U$ *then there exist* $\overline{T}, \overline{U}, F$ *such that* $T = \langle \overline{T} \rangle$*,* $U = \langle \overline{U} \rangle$*, and* $\Gamma \vdash_{\mathsf{alg}} \overline{T} <: \overline{U}; F$ *and* $\Gamma; \Delta \vdash F$.

PROOF.

(1) We first prove part (1a). The proof proceeds by induction on the length of $\Gamma \vdash_{\mathsf{alg}} \overline{T} <: \overline{U}; F$. We use the non-annotated types $T := \langle \overline{T} \rangle$ and $U := \langle \overline{U} \rangle$ whenever applicable.
The base cases (SUB REFL ALG) and (SUB REFL REC ALG) follow by an inspection of the typing rules and by Lemma C.4.
We now discuss the induction step by analyzing the last applied algorithmic subtyping rule.

*Case* (SUB PUB TNT ALG): $\Gamma \vdash_{\mathsf{alg}} T <: U; F_1 \otimes F_2$ is proved by $\Gamma \vdash_{\mathsf{alg}} T :: \mathsf{pub}; F_1$ and $\Gamma \vdash_{\mathsf{alg}} U :: \mathsf{tnt}; F_2$, where $T = \overline{T}$ and $U = \overline{U}$ are not annotated. We also know that $\Gamma; \Delta \vdash F_1 \otimes F_2$ and $T \neq_{\top} U$.
  *To show:* $\Gamma; \Delta \vdash T <:_{\mathsf{alg}} U$. By (REWRITE), (DERIVE), and Lemma B.5 we know that $\Gamma; \Delta \hookrightarrow \Gamma; F_1, F_2$ and $\Gamma; F_1 \vdash F_1$ and $\Gamma; F_2 \vdash F_2$ by (IDENT) and (DERIVE). By Lemma C.5, $\Gamma; F_1 \vdash T :: \mathsf{pub}$ and $\Gamma; F_2 \vdash U :: \mathsf{tnt}$. The result follows from (SUB PUB TNT *).

*Case* (SUB PUB TNT REC ALG): The proof follows the same structure as the one for (SUB PUB TNT ALG) and concludes with an application of (SUB PUB TNT REC *).

*Case* (SUB FUN ALG): $\Gamma \vdash_{\mathsf{alg}} x : \overline{T_1} \to \overline{T_2} <: x : \overline{U_1} \to \overline{U_2}; !F_1 \otimes !F_2$ is derived by $\Gamma \vdash_{\mathsf{alg}} \overline{U_1} <: \overline{T_1}; F_1$ and $\Gamma, x : \psi(U_1) \vdash_{\mathsf{alg}} \overline{T_2} <: \overline{U_2}; F_2$. We also know that $\Gamma; \Delta \vdash !F_1 \otimes !F_2$.

*To show:* $\Gamma; \Delta \vdash x : \overline{T_1} \to \overline{T_2} <:_{\mathsf{alg}} x : \overline{U_1} \to \overline{U_2}$. By (REWRITE), (DERIVE), and Lemma B.5 we know that $\Gamma; \Delta \hookrightarrow \Gamma; !F_1, !F_2$ and $\Gamma; !F_1 \vdash F_1$, and $\Gamma; !F_2 \vdash F_2$ by (IDENT), (!-LEFT), and (DERIVE).

By induction hypothesis, $\Gamma; !F_1 \vdash \overline{U_1} <:_{\mathsf{alg}} \overline{T_1}$ and $\Gamma, x : \psi(U_1); !F_2 \vdash \overline{T_2} <:_{\mathsf{alg}} \overline{U_2}$. The result follows from (SUB FUN *).

*Case* (SUB PAIR ALG): The proof is analogous to the one for (SUB FUN ALG).

*Case* (SUB SUM ALG): The proof is analogous to the one for (SUB FUN ALG).

*Case* (SUB POS REC ALG): $\Gamma \vdash_{\mathsf{alg}} \mu\alpha.\overline{T_1} <: \mu\alpha.\overline{U_1}; !F$ is proved by $\Gamma, \alpha \vdash_{\mathsf{alg}} \overline{T_1} <: \overline{U_1}; F$. We also know that $\Gamma; \Delta \vdash !F$ and that $\alpha$ occurs only positively in $\overline{T_1}$ and $\overline{U_1}$.

*To show:* $\Gamma; \Delta \vdash \mu\alpha.\overline{T_1} <:_{\mathsf{alg}} \mu\alpha.\overline{U_1}$. By (REWRITE), (DERIVE), and Lemma B.5 we know that$\Gamma; \Delta \hookrightarrow \Gamma; !F$ and $\Gamma; !F \vdash F$ by (IDENT), (!-LEFT), and (DERIVE).

By induction hypothesis, $\Gamma, \alpha; !\Delta' \vdash \overline{T_1} <: \overline{U_1}$. The result follows from (SUB POS REC *).

*Case* (SUB REFINE ALG): $\Gamma; \Delta \vdash_{\mathsf{alg}} \overline{T} <: \overline{U}; F \otimes \forall y.(\mathit{forms}(y : T) \multimap \mathit{forms}(y : U))$ is proved by $\Gamma \vdash_{\mathsf{alg}} \psi(\overline{T}) <: \psi(\overline{U}); F$. We also know that $\Gamma; \Delta \vdash F \otimes \forall y.(\mathit{forms}(y : T) \multimap \mathit{forms}(y : U))$ and that at least one of the types $\overline{T}, \overline{U}$ is refined.

*To show:* $\Gamma; \Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}$. By (REWRITE), (DERIVE), and Lemma B.5 we know that $\Gamma; \Delta \hookrightarrow \Gamma; F, \forall y.(\mathit{forms}(y : T) \multimap \mathit{forms}(y : U))$, $\Gamma; F \vdash F$ and $\Gamma; \forall y.(\mathit{forms}(y : T) \multimap \mathit{forms}(y : U)) \vdash \forall y.(\mathit{forms}(y : T) \multimap \mathit{forms}(y : U))$ by (IDENT) and (DERIVE).

By induction hypothesis, $\Gamma; F \vdash \psi(\overline{T}) <:_{\mathsf{alg}} \psi(\overline{U})$.

Without loss of generality, let us assume $y \notin \mathit{dom}(\Gamma)$ and, thus, $y \notin \mathit{fnfv}(\forall y.(\mathit{forms}(y : T) \multimap \mathit{forms}(y : U)))$. (This assumption can be fulfilled by $\alpha$-renaming $y$ if necessary.)

By Lemma B.5, we can easily see that $\Gamma, y : \psi(T); \forall y.(\mathit{forms}(y : T) \multimap \mathit{forms}(y : U)) \vdash \diamond$. By Lemma C.3, $\Gamma, y : \psi(T); \forall y.(\mathit{forms}(y : T) \multimap \mathit{forms}(y : U)) \vdash \mathit{forms}(y : T) \multimap \mathit{forms}(y : U)$. By Lemma C.2, $\Gamma, y : \psi(T); \Delta_2, \mathit{forms}(y : T) \vdash \mathit{forms}(y : U)$.

The result follows from (SUB REFINE *).

We then prove part (1b). The proof proceeds by induction on the length of $\Gamma; \Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}$. The base cases (SUB REFL *) and (SUB REFL REC *), follow by an inspection of the subtyping rules, by Lemma C.4, and by observing that $\Gamma; \Delta \vdash \mathbf{1}$ for any $\Delta$ such that $\Gamma; \Delta \vdash \diamond$. We now discuss the induction step by distinguishing the last applied intermediate subtyping rule.

*Case* (SUB PUB TNT *): $\Gamma; \Delta \vdash T <:_{\mathsf{alg}} U$ is proved by $\Gamma; \Delta_1 \vdash T :: \mathsf{pub}$, $\Gamma; \Delta_2 \vdash U :: \mathsf{tnt}$, $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$, and $T \neq_{\top} U$. Note that here $T = \overline{T}$ and $U = \overline{U}$ are not annotated.

*To show:* $\Gamma \vdash_{\mathsf{alg}} T <: U; F_1 \otimes F_2$. By Lemma C.5, there exist $F_1, F_2$ such that $\Gamma \vdash_{\mathsf{alg}} T :: \mathsf{pub}; F_1$, $\Gamma \vdash_{\mathsf{alg}} U :: \mathsf{tnt}; F_2$, $\Gamma; \Delta_1 \vdash F_1$, and $\Gamma; \Delta_2 \vdash F_2$. The result follows from (SUB PUB TNT ALG).

*To show:* $\Gamma; \Delta \vdash F_1 \otimes F_2$ . By ($\otimes$-RIGHT), $\Gamma; \Delta_1, \Delta_2 \vdash F_1 \otimes F_2$. The result follows from Lemma B.9.

*Case* (SUB PUB TNT REC *): The proof follows the same structure as the one for (SUB PUB TNT *) and concludes with an application of (SUB PUB TNT REC ALG).

*Case* (SUB FUN *): $\Gamma; \Delta \vdash x : \overline{T_1} \to \overline{T_2} <:_{\mathsf{alg}} x : \overline{U_1} \to \overline{U_2}$ is proved by $\Gamma; !\Delta_1 \vdash \overline{U_1} <:_{\mathsf{alg}} \overline{T_1}$, $\Gamma, x : \psi(U_1); !\Delta_2 \vdash \overline{T_2} <:_{\mathsf{alg}} \overline{U_2}$, and $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2$. By induction hypothesis, there exist $F_1, F_2$ such that $\Gamma \vdash_{\mathsf{alg}} \overline{U_1} <: \overline{T_1}; F_1$, $\Gamma, x : \psi(U_1) \vdash_{\mathsf{alg}} \overline{T_2} <: \overline{U_2}; F_2$, $\Gamma; !\Delta_1 \vdash F_1$, and $\Gamma; !\Delta_2 \vdash F_2$.

*To show:* $\Gamma \vdash_{\mathsf{alg}} x : \overline{T_1} \to \overline{T_2} <: x : \overline{U_1} \to \overline{U_2}; !F_1 \otimes !F_2$. The result follows from (SUB FUN ALG).

*To show:* $\Gamma; \Delta \vdash !F_1 \otimes !F_2$. By (!-RIGHT) and (DERIVE) and Lemma B.5 we know that $\Gamma; !\Delta_1 \vdash !F_1$ and $\Gamma; !\Delta_2 \vdash !F_2$. By ($\otimes$-RIGHT) and (DERIVE), $\Gamma; !\Delta_1, !\Delta_2 \vdash !F_1 \otimes !F_2$. The result follows from Lemma B.9.

*Case* (SUB PAIR *): The proof is analogous to the one for (SUB FUN *).

*Case* (SUB SUM *): The proof is analogous to the one for (SUB FUN *).

*Case* (SUB POS REC *): $\Gamma; \Delta \vdash \mu\alpha.\overline{T_1} <:_{\mathsf{alg}} \mu\alpha.\overline{U_1}$ is proved by $\Gamma, \alpha; !\Delta' \vdash \overline{T_1} <:_{\mathsf{alg}} \overline{U_1}$ and $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$. We furthermore know that $\alpha$ occurs only positively in $\overline{T_1}$ and $\overline{U_1}$. By induction hypothesis, there exists $F$ such that $\Gamma, \alpha \vdash_{\mathsf{alg}} \overline{T_1} <: \overline{U_1}; F$ and $\Gamma; !\Delta' \vdash F$.

*To show:* $\Gamma \vdash_{\mathsf{alg}} \mu\alpha.\overline{T_1} <: \overline{U_1}; !F$. The result follows from (SUB POS REC ALG).

*To show:* $\Gamma; \Delta \vdash !F$. By (DERIVE) and (!-RIGHT) we know that $\Gamma; !\Delta' \vdash !F$. We conclude using Lemma B.9.

*Case* (SUB REFINE *): $\Gamma; \Delta \vdash \overline{T} <:_{\mathsf{alg}} \overline{U}$ is proved by $\Gamma; \Delta_1 \vdash \psi(\overline{T}) <:_{\mathsf{alg}} \psi(\overline{U})$, $\Gamma, y : \psi(T); \Delta_2, \mathit{forms}(y : T) \vdash \mathit{forms}(y : U)$, and $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$. By induction hypothesis, there exists $F$ such that $\Gamma \vdash_{\mathsf{alg}} \psi(\overline{T}) <: \psi(\overline{U}); F$ and $\Gamma; \Delta_1 \vdash F$. We also know that at least one of the types $\overline{T}, \overline{U}$ is refined.

*To show:* $\Gamma; \Delta \vdash_{\mathsf{alg}} \overline{T} <: \overline{U}; (\forall y.\mathit{forms}(y : T) \multimap \mathit{forms}(y : U)) \otimes F$. The result follows by (SUB REFINE ALG).

*To show:* $\Gamma; \Delta \vdash (\forall y.\mathit{forms}(y : T) \multimap \mathit{forms}(y : U)) \otimes F$. We know that $(\Gamma, y : \psi(T); \Delta_2, \mathit{forms}(y : T)) \vdash \mathit{forms}(y : U)$. By ($\multimap$-RIGHT), $(\Gamma, y : \psi(T); \Delta_2) \vdash \mathit{forms}(y : T) \multimap \mathit{forms}(y : U)$.

By Lemma B.5, $\Gamma, x : \psi(T); \Delta_2 \vdash \diamond$ and, thus, $x \notin \mathit{dom}(\Gamma)$. Since $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$, we also have that $\Gamma; \Delta_1, \Delta_2 \vdash \diamond$ and, thus, $x \notin \mathit{fnfv}(\Delta_2) \subseteq \mathit{dom}(\Gamma)$.

By Lemma C.3, $(\Gamma; \Delta_2) \vdash \forall y.\mathit{forms}(y : T) \multimap \mathit{forms}(y : U)$. By ($\otimes$-RIGHT), $\Gamma; \Delta_1, \Delta_2 \vdash F \otimes \forall y.(\mathit{forms}(y : T) \multimap \mathit{forms}(y : U))$. The result follows from Lemma B.9.

(2) Part (2a) follows immediately from statement (1a) and the soundness of intermediate subtyping, shown in Lemma C.10, statement (3).
Part (2b) follows immediately from statement (1b) and the completeness of intermediate subtyping, shown in Lemma C.10, statement (4).

□

The following lemma is used in the proof of soundness and completeness of algorithmic typing and states that for algorithmic subtyping type annotations need only occur in *either* the sub- or supertype.

LEMMA C.12 (ONE-SIDED TYPE ANNOTATIONS). *For all* $\Gamma, \overline{T}, \overline{U}, T, UF$ *such that* $\Gamma \vdash_{\mathsf{alg}} \overline{T} <: \overline{U}; F$ *such that* $T = \langle \overline{T} \rangle$ *and* $U = \langle \overline{U} \rangle$ *it holds that:*

(1) *there exist* $\overline{U}'$ *such that* $\langle \overline{U} \rangle = \langle \overline{U}' \rangle = U$ *and* $\Gamma \vdash_{\mathsf{alg}} T <: \overline{U}'; F$*;*
(2) *there exist* $\overline{T}'$ *such that* $\langle \overline{T} \rangle = \langle \overline{T}' \rangle = T$ *and* $\Gamma \vdash_{\mathsf{alg}} \overline{T}' <: U; F$*.*

PROOF. The proof proceeds by simultaneous induction on the length of the derivation of $\Gamma \vdash_{\mathsf{alg}} \overline{T} <: \overline{U}; F$. For the base cases (SUB REFL ALG), (SUB REFL REC ALG), and (SUB PUB TNT ALG) we know that $\overline{T}, \overline{U}$ are not annotated and thus $\langle \overline{T} \rangle = \overline{T} = T$ and $\langle \overline{T} \rangle = \overline{T} = T$. We immediately conclude by selecting $\overline{T}' := T$ and $\overline{U}' := U$.

We now show the inductive cases:

*Case* (SUB FUN ALG) : Notice that the subtyping rule is contravariant in the input.

(1) Statement (1) follows by applying the induction hypothesis (2) to the first subtyping premise and the induction hypothesis (1) to the second subtyping premise. We conclude by an application of (SUB FUN ALG).
(2) Statement (2) follows by applying the induction hypothesis (1) to the first subtyping premise and the induction hypothesis (2) to the second subtyping premise. We conclude by an application of (SUB FUN ALG).

*Case* (SUB PAIR ALG):
(1) Statement (1) follows by applying the induction hypothesis (1) to the first subtyping premise and the induction hypothesis (1) to the second subtyping premise. We conclude by an application of (SUB PAIR ALG).
(2) Statement (2) follows by applying the induction hypothesis (2) to the first subtyping premise and the induction hypothesis (2) to the second subtyping premise. We conclude by an application of (SUB PAIR ALG).

*Case* (SUB SUM ALG):
(1) Statement (1) follows by applying the induction hypothesis (1) to the subtyping premise. We conclude by an application of (SUB SUM ALG).
(2) Statement (2) follows by applying the induction hypothesis (2) to the subtyping premise. We conclude by an application of (SUB SUM ALG).

*Case* (SUB POS REC ALG):
(1) Statement (1) follows by applying the induction hypothesis (1) to the subtyping premise. We conclude by an application of (SUB POS REC ALG).
(2) Statement (2) follows by applying the induction hypothesis (2) to the subtyping premise. We conclude by an application of (SUB POS REC ALG).

*Case* (SUB PUB TNT REC ALG): In this case we know that $\overline{T} = (\mu\alpha.\, T')_s$ and $\overline{U} = (\mu\alpha.\, U')_{s'}$, where $s = \mathsf{SPT} \oplus s' = \mathsf{SPT}$. Furthermore, we know that $T = \langle\overline{T}\rangle = \mu\alpha.\, T'$ and $U = \langle\overline{U}\rangle = \mu\alpha.\, U'$ by definition of $\langle\bullet\rangle$.
(1) We immediately conclude by selecting $\overline{U}' = (\mu\alpha.\, U')_{\mathsf{SPT}}$ and applying (SUB PUB TNT REC ALG).
(2) We immediately conclude by selecting $\overline{T}' = (\mu\alpha.\, T')_{\mathsf{SPT}}$ and applying (SUB PUB TNT REC ALG).

*Case* (SUB REFINE ALG): We observe that refinements types are never annotated on a top-level, only the core type stored therein may be.
(1) Statement (1) follows by applying the induction hypothesis (1) to the subtyping premise and using Lemma C.8 and Lemma C.9. We conclude by an application of (SUB REFINE ALG).
(2) Statement (2) follows by applying the induction hypothesis (2) to the subtyping premise and using Lemma C.8 and Lemma C.9. We conclude by an application of (SUB REFINE ALG).

□

The following proposition is used in the proof of soundness and completeness of algorithmic typing and states that extraction is unaffected by typing annotations.

PROPOSITION C.13 (ANNOTATED EXTRACTION). *For all $\widetilde{a}, \Delta$ it holds that:*

(1) *for all $\overline{E}, \overline{D}$ such that $\overline{E} \leadsto^{\widetilde{a}} [\Delta \mid \overline{D}]$ it must be the case that $\langle\overline{E}\rangle \leadsto^{\widetilde{a}} [\Delta \mid \langle\overline{D}\rangle]$;*
(2) *for all $E, D, \overline{D}$ such that $D = \langle\overline{D}\rangle$ and $E \leadsto^{\widetilde{a}} [\Delta \mid D]$ it must be the case that there exists an annotated expression $\overline{E}$ such that $E = \langle\overline{E}\rangle$ and $\overline{E} \leadsto^{\widetilde{a}} [\Delta \mid \overline{D}]$.*

LEMMA C.14 (TYPING TRUTH ASSUMPTION). *For all $\Gamma, \Delta, T$ such that $\Gamma; \Delta \vdash \mathsf{assume}\ \mathbf{1} : T$ it holds that $\Gamma; \emptyset \vdash \mathsf{assume}\ \mathbf{1} : T$ and $\Gamma; \Delta \vdash \mathsf{unit} <: T$.*

PROOF. By induction on the length of the derivation $\Gamma; \Delta \vdash \mathsf{assume}\ \mathbf{1} : T$. We first note that $\Gamma; \emptyset \vdash \mathsf{assume}\ \mathbf{1} : \mathsf{unit}$ by (EXP TRUE). By Lemma B.5 we know that $\Gamma; \Delta \vdash T$. In the base case (EXP TRUE) we know that $T = \mathsf{unit}$ and conclude that $\Gamma; \Delta \vdash \mathsf{unit} <: \mathsf{unit}$ by (SUB REFL).

In the inductive case we know that the last applied rule must have been (EXP SUBSUM) and thus there exist $T', \Delta_1, \Delta_2$ such that $\Gamma; \Delta_1 \vdash \mathsf{assume}\ \mathbf{1} : T'$ and $\Gamma; \Delta_2 \vdash T' <: T$, where $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$. By induction hypothesis we know that $\Gamma; \Delta_1 \vdash \mathsf{unit} <: T'$ and we can thus conclude by an application of Lemma B.22 and Lemma B.9. □

RESTATEMENT 5 (OF THEOREMS 10.1 AND 10.2). *For all $\Gamma, \Delta, T$, the following holds true:*

(1) *for all $\overline{E}, F$ such that $\Gamma \vdash_{\mathsf{alg}} \overline{E} : T; F$ and $\Gamma; \Delta \vdash F$, we have that $\Gamma; \Delta \vdash \langle \overline{E} \rangle : T$;*
(2) *for all $E$ such that $\Gamma; \Delta \vdash E : T$, there exist $\overline{E}, F$ such that $\langle \overline{E} \rangle = E$, $\Gamma \vdash_{\mathsf{alg}} \overline{E} : T; F$, and $\Gamma; \Delta \vdash F$.*

PROOF.

(1) The proof proceeds by induction on the length of $\Gamma \vdash_{\mathsf{alg}} \overline{E} : T; F$. The base cases are (VAL VAR ALG), (VAL UNIT ALG), (EXP TRUE ALG), (EXP RECV ALG), and (EXP ASSERT ALG): in all of these cases $\overline{E} = \langle \overline{E} \rangle$ (meaning $\overline{E}$ does not contain annotations) and they follow by an inspection of the typing rules and by Lemma C.4. We show the induction cases in the following. Most cases follow a very similar structure so we show detailed examples for standard proof strategies and omit the details for analogous cases.

*Case* (VAL FUN ALG): $\Gamma \vdash_{\mathsf{alg}} \lambda x : T_1. \overline{D} : x : T_1 \to T_2; !\forall x.(\mathit{forms}(x : T_1) \multimap F')$ is proved by $\Gamma, x : \psi(T_1) \vdash_{\mathsf{alg}} \overline{D} : T_2; F'$. We also know that $\Gamma; \Delta \vdash !\forall x.(\mathit{forms}(x : T_1) \multimap F')$.

*To show:* $\Gamma; \Delta \vdash \langle \lambda x : T. \overline{D} \rangle : x : T_1 \to T_2$. We first note that $\langle \lambda x : T. \overline{D} \rangle$ is equal to $\lambda x. \langle \overline{D} \rangle$. By (REWRITE), (DERIVE), and Lemma B.5 we know that $\Gamma; \Delta \hookrightarrow \Gamma; !\forall x.(\mathit{forms}(x : T_1) \multimap F')$ and $\Gamma; !\forall x.(\mathit{forms}(x : T_1) \multimap F') \vdash \forall x.(\mathit{forms}(x : T_1) \multimap F')$ by (IDENT), (!-LEFT), and (DERIVE).
Without loss of generality, let us assume $x \notin \mathit{dom}(\Gamma)$ and, thus, $x \notin \mathit{fnfv}(!\forall x.(\mathit{forms}(x : T_1) \multimap F'))$. (This assumption can be fulfilled by $\alpha$-renaming $x$ if necessary.)
By Lemma B.5, we can easily see that $\Gamma, x : \psi(T_1); !\forall x.(\mathit{forms}(x : T_1) \multimap F') \vdash \diamond$.
By Lemma C.3, $\Gamma, x : \psi(T_1); !\forall x.(\mathit{forms}(x : T_1) \multimap F') \vdash \mathit{forms}(x : T_1) \multimap F'$. By Lemma C.2, $\Gamma, x : \psi(T_1); !\forall x.(\mathit{forms}(x : T_1) \multimap F'), \mathit{forms}(x : T_1) \vdash F'$.
By induction hypothesis, $\Gamma, x : \psi(T_1); !\forall x.(\mathit{forms}(x : T_1) \multimap F'), \mathit{forms}(x : T_1) \vdash \langle \overline{D} \rangle : T_2$.
The result follows by an application of (VAL FUN).

*Case* (VAL PAIR ALG): $\Gamma \vdash_{\mathsf{alg}} (\overline{M}, \overline{N}) : x : T_1 * T_2; !F_1 \otimes !F_2$ is proved by $\Gamma \vdash_{\mathsf{alg}} \overline{M} : T_1; F_1$ and $\Gamma \vdash_{\mathsf{alg}} \overline{N} : T_2\{M/x\}; F_2$, where $M := \langle \overline{M} \rangle$. We also know that $\Gamma; \Delta \vdash !F_1 \otimes !F_2$.

*To show:* $\Gamma; \Delta \vdash \langle (\overline{M}, \overline{N}) \rangle : x : T_1 * T_2$. We first note that $\langle (\overline{M}, \overline{N}) \rangle$ is equal to $(\langle \overline{M} \rangle, \langle \overline{N} \rangle)$. By (REWRITE), (DERIVE), and Lemma B.5 we know that $\Gamma; \Delta \hookrightarrow \Gamma; !F_1, !F_2$ and $\Gamma; !F_1 \vdash F_1$ and $\Gamma; !F_2 \vdash F_2$ by (IDENT), (!-LEFT), and (DERIVE).
By applying the induction hypothesis twice we know that $\Gamma; !F_1 \vdash \langle \overline{M} \rangle : T_1$ and $\Gamma; !F_2 \vdash \langle \overline{N} \rangle : T_2\{\langle \overline{M} \rangle / x\}$.
The result follows by an application of (VAL PAIR).

*Case* (VAL INL ALG): $\Gamma \vdash_{\mathsf{alg}} (\mathsf{inl}\ \overline{M})_{\_+T_2} : T_1 + T_2; !F'$ is proved by $\Gamma \vdash_{\mathsf{alg}} \overline{M} : T_1; F_1$ and $\Gamma \vdash_{\mathsf{alg}} T_2$. We also know that $\Gamma; \Delta \vdash !F'$.

*To show:* $\Gamma;\Delta \vdash \langle(\mathsf{inl}\ \overline{M})_{\_+T_2}\rangle : T_1 + T_2$. We first note that $\langle(\mathsf{inl}\ \overline{M})_{\_+T_2}\rangle$ is equal to $\mathsf{inl}\ \langle\overline{M}\rangle$. By (REWRITE), (DERIVE), and Lemma B.5 we know that $\Gamma;\Delta \hookrightarrow \Gamma;!F'$ and $\Gamma;!F' \vdash F'$ by (IDENT), (!-LEFT), and (DERIVE).
By applying the induction hypothesis we know that $\Gamma;!F' \vdash \langle\overline{M}\rangle : T_1$.
By Lemma C.4 we know that $\Gamma;\emptyset \vdash T_2$ and thus by Lemma B.7 $\Gamma;!F' \vdash T_2$.
The result follows by an application of (VAL INL).

*Case* (VAL INR ALG): The proof is analogous to the one for (VAL INL ALG).

*Case* (VAL FOLD ALG): $\Gamma \vdash_{\mathsf{alg}} \mathsf{fold}\ \overline{M} : \mu\alpha.T';!F'$ is proved by $\Gamma \vdash_{\mathsf{alg}} \overline{M} : T'\{\mu a.T'/\alpha\};F'$. We also know that $\Gamma;\Delta \vdash !F'$.

*To show:* $\Gamma;\Delta \vdash \langle\mathsf{fold}\ \overline{M}\rangle : \mu\alpha.T'$. We first note that $\langle\mathsf{fold}\ \overline{M}\rangle$ is equal to $\mathsf{fold}\ \langle\overline{M}\rangle$. By (REWRITE), (DERIVE), and Lemma B.5 we know that $\Gamma;\Delta \hookrightarrow \Gamma;!F'$ and $\Gamma;!F' \vdash F'$ by (IDENT), (!-LEFT), and (DERIVE).
By applying the induction hypothesis we know that $\Gamma;!F' \vdash \langle\overline{M}\rangle : T'\{\mu a.T'/\alpha\}$.
The result follows by an application of (VAL FOLD).

*Case* (VAL REF ALG): $\Gamma \vdash_{\mathsf{alg}} \overline{M}_{\{x:\_\ \mid\ F\}} : \{x : T' \mid F\};F' \otimes F\{\langle\overline{M}\rangle/x\}$ is proved by $\Gamma \vdash_{\mathsf{alg}} \overline{M} : T';F'$ and $\mathit{fnfv}(F) \subseteq \mathit{dom}(\Gamma) \cup \{x\}$. We also know that $\Gamma;\Delta \vdash F' \otimes F\{\langle\overline{M}\rangle/x\}$.

*To show:* $\Gamma;\Delta \vdash \langle\overline{M}_{\{x:\_\ \mid\ F\}}\rangle : \{x : T' \mid F\}$. We first note that $\langle\overline{M}_{\{x:\_\ \mid\ F\}}\rangle$ is equal to $\langle\overline{M}\rangle$. By (REWRITE), (DERIVE), and Lemma B.5 we know that $\Gamma;\Delta \hookrightarrow \Gamma;F', F\{\langle\overline{M}\rangle/x\}$ and $\Gamma;F' \vdash F'$ and $\Gamma;F\{\langle\overline{M}\rangle/x\} \vdash F\{\langle\overline{M}\rangle/x\}$ by (IDENT). By induction hypothesis, $\Gamma;F' \vdash \langle\overline{M}\rangle : T$. The result follows from (VAL REFINE).

*Case* (EXP APPL ALG): The proof follows straightforwardly from (REWRITE), (DERIVE), Lemma B.5, (IDENT) by applying the induction hypothesis twice.

*Case* (EXP LET ALG): $\Gamma \vdash_{\mathsf{alg}} \mathsf{let}\ x = \overline{E_1}\ \mathsf{in}\ \overline{E_2} : T;\Delta' \multimap (F_1 \otimes (\forall x.\mathit{forms}(x : U) \multimap F_2))$ is proved by $\overline{E_1} \rightsquigarrow^{\emptyset} [\Delta' \mid \overline{E_1'}]$, $\Gamma \vdash_{\mathsf{alg}} \overline{E_1'} : U;F_1$, $\Gamma, x : \psi(U) \vdash_{\mathsf{alg}} \overline{E_2} : T;F_2$, $x \notin \mathit{fv}(T)$. We also know that $\Gamma;\Delta \vdash \Delta' \multimap (F_1 \otimes (\forall x.\mathit{forms}(x : U) \multimap F_2))$.

*To show:* $\Gamma;\Delta \vdash \langle\mathsf{let}\ x = \overline{E_1}\ \mathsf{in}\ \overline{E_2}\rangle : T$. We first note that $\langle\mathsf{let}\ x = \overline{E_1}\ \mathsf{in}\ \overline{E_2}\rangle$ is equal to $\mathsf{let}\ x = \langle\overline{E_1}\rangle\ \mathsf{in}\ \langle\overline{E_2}\rangle$. By Lemma C.2 and Lemma B.2, $\Gamma;\Delta,\Delta' \vdash F_1 \otimes (\forall x.\mathit{forms}(x : U) \multimap F_2)$.
By (REWRITE), (DERIVE), and Lemma B.5 it holds that $\Gamma;\Delta,\Delta' \hookrightarrow \Gamma;F_1,(\forall x.\mathit{forms}(x : U) \multimap F_2)$ and $\Gamma;F_1 \vdash F_1$ and $\Gamma;\forall x.\mathit{forms}(x : U) \multimap F_2 \vdash \forall x.\mathit{forms}(x : U) \multimap F_2$ by (IDENT) and (DERIVE).
Without loss of generality, let us assume $x \notin \mathit{dom}(\Gamma)$ and, thus, $x \notin \mathit{fnfv}(\forall x.\mathit{forms}(x : U) \multimap F_2)$. (This assumption can be fulfilled by $\alpha$-renaming $x$ if necessary.)
By Lemma B.5, we can easily see that $\Gamma, x : \psi(U);\forall x.\mathit{forms}(x : U) \multimap F_2 \vdash \diamond$. By Lemma C.3, $\Gamma, x : \psi(U);\forall x.\mathit{forms}(x : U) \multimap F_2 \vdash \mathit{forms}(x : U) \multimap F_2$. By Lemma C.2, $\Gamma, x : \psi(U);\forall x.\mathit{forms}(x : U) \multimap F_2, \mathit{forms}(x : U) \vdash F_2$.
We note that by statement (1) of Proposition C.13 it holds that $\langle\overline{E_1}\rangle \rightsquigarrow^{\emptyset} [\Delta' \mid \langle\overline{E_1'}\rangle]$.
By induction hypothesis, $\Gamma;F_1 \vdash \langle\overline{E_1'}\rangle : T$ and $\Gamma, x : \psi(U);\forall x.\mathit{forms}(x : U) \multimap F_2, \mathit{forms}(x : U) \vdash \langle\overline{E_2}\rangle : U$.
The result follows from (EXP LET).

*Case* (EXP SPLIT ALG): The proof follows a similar strategy as the one for (EXP LET ALG).

*Case* (EXP MATCH ALG): The proof follows a similar strategy as the one for (EXP LET ALG).

*Case* (EXP EQ ALG): The proof follows straightforwardly from (REWRITE), (DERIVE), (IDENT), Lemma B.5, the induction hypothesis, ($\otimes$-RIGHT), and Lemma B.9.

*Case* (EXP ASSUME ALG): $\Gamma \vdash_{\mathsf{alg}} (\mathsf{assume}\ F_1)_T : T; F_1 \multimap F_2$ is proved by $\Gamma \vdash_{\mathsf{alg}} (\mathsf{assume}\ \mathbf{1})_{\_<:T} : T; F_2$ and $\mathit{fnfv}(F) \subseteq \mathit{dom}(\Gamma)$, where $F_1 \neq \mathbf{1}$. We also know that $\Gamma; \Delta \vdash F_1 \multimap F_2$.

*To show:* $\Gamma; \Delta \vdash \langle(\mathsf{assume}\ F_1)_T\rangle : T$. We first note that $\langle(\mathsf{assume}\ F_1)_T\rangle$ is equal to $\mathsf{assume}\ F_1$.
By Lemma C.2 we know that $\Gamma; \Delta, F_1 \vdash F_2$.
By applying the induction hypothesis we know that $\Gamma; \Delta, F_1 \vdash \mathsf{assume}\ \mathbf{1} : T$.
The result follows by an application of (EXP ASSUME) .

*Case* (EXP RES ALG): The proof follows a similar and slightly simplified strategy as the one for (EXP LET ALG).

*Case* (EXP SEND ALG): The proof follows straightforwardly from the induction hypothesis using the fact that $\langle a!\overline{M}\rangle$ is equal to $a!\langle\overline{M}\rangle$.

*Case* (EXP FORK ALG): The proof follows a similar strategy as the one for (EXP LET ALG).

(2) The proof proceeds by induction on the length of $\Gamma; \Delta \vdash E : T$. The base cases are (VAL VAR), (VAL UNIT), (EXP TRUE), (EXP RECV), and (EXP ASSERT): in these cases we choose $\overline{E} := E$ and $F := \mathbf{1}$. The statement follows by an inspection of the typing rules and by Lemma C.4.
We show the induction cases in the following. Most cases follow a very similar structure so we show detailed examples for standard proof strategies and omit the details for analogous cases.
For all cases the proof is split into two parts: we first show that there exists an annotated term $\overline{E}$ and a formula $F$ such that $\Gamma \vdash_{\mathsf{alg}} \overline{E} : T; F$ and $\langle\overline{E}\rangle = E$. We then prove that $\Gamma; \Delta \vdash F$.

*Case* (VAL FUN): $\Gamma; \Delta \vdash \lambda x.\, D : x : T_1 \to T_2$ is proved by $\Gamma, x : \psi(T_1); !\Delta', \mathit{forms}(x : T_1) \vdash D : T_2$ and $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$.
By induction hypothesis, there exist $\overline{D}, F'$ such that $\langle\overline{D}\rangle = D$, $\Gamma, x : \psi(T_1) \vdash \overline{D} : T_2; F'$ and $\Gamma, x : \psi(T_1); !\Delta', \mathit{forms}(x : T_1) \vdash F'$.

*To show:* $\Gamma \vdash_{\mathsf{alg}} \lambda x : T_1.\, \overline{D} : x : T_1 \to T_2; !\forall x.(\mathit{forms}(x : T_1) \multimap F')$. By Lemma B.5, $\mathit{fnfv}(T_1) \subseteq \mathit{dom}(\Gamma) \cup \{x\}$. The result follows from (VAL FUN ALG). We note that $\langle\lambda x : T_1.\, \overline{D}\rangle = \lambda x.\, D$.
*To show:* $\Gamma; \Delta \vdash !\forall x.(\mathit{forms}(x : T_1) \multimap F')$. By ($\multimap$-RIGHT), $\Gamma, x : \psi(T_1); !\Delta' \vdash \mathit{forms}(x : T_1) \multimap F'$.
By Lemma B.5, $\Gamma, x : \psi(T_1) \vdash \diamond$ and $x \notin \mathit{fnfv}(!\Delta') \subseteq \mathit{dom}(\Gamma)$. By Lemma C.3, $\Gamma; !\Delta' \vdash \forall x.(\mathit{forms}(x : T_1) \multimap F')$. By (!-RIGHT), $\Gamma; !\Delta' \vdash !\forall x.(\mathit{forms}(x : T_1) \multimap F')$.
The result follows from Lemma B.9.

*Case* (VAL PAIR): $\Gamma; \Delta \vdash (M, N) : x : T_1 * T_2$ is proved by $\Gamma; !\Delta_1 \vdash M : T_1$ $\Gamma; !\Delta_2 \vdash N : T_2\{M/x\}$ and $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta_1, !\Delta_2$.
By applying the induction hypothesis twice we know that there exist $\overline{M}, \overline{N}, F_1, F_2$ such that $\langle\overline{M}\rangle = M$ and $\langle\overline{N}\rangle = N$ and $\Gamma \vdash \overline{M} : T_1; F_1$ and $\Gamma \vdash \overline{N} : T_2\{M/x\}; F_2$ and $\Gamma; !\Delta_1 \vdash F_1$ and $\Gamma; !\Delta_2 \vdash F_2$.

*To show:* $\Gamma \vdash_{\mathsf{alg}} (\overline{M}, \overline{N}) : x : T_1 * T_2; !F_1 \otimes !F_2$. The result follows immediately from (VAL PAIR ALG). We note that $\langle(\overline{M}, \overline{N})\rangle = (M, N)$.
*To show:* $\Gamma; \Delta \vdash !F_1 \otimes !F_2$. We apply (!-RIGHT) to derive that $\Gamma; !\Delta_1 \vdash !F_1$ and $\Gamma; !\Delta_2 \vdash !F_2$. By ($\otimes$-RIGHT), $\Gamma; !\Delta_1, !\Delta_2 \vdash !F_1 \otimes !F_2$.
The result follows from Lemma B.9.

*Case* (VAL INL): $\Gamma; \Delta \vdash \mathsf{inl}\ M : T_1 + T_2$ is proved by $\Gamma; !\Delta' \vdash M : T_1$ and $\Gamma; !\Delta' \vdash T_2$ and $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$.
By applying the induction hypothesis we know that there exist $\overline{M}, F'$ such that $\langle\overline{M}\rangle = M$ and $\Gamma \vdash \overline{M} : T_1; F'$ and $\Gamma; !\Delta' \vdash F'$.

*To show:* $\Gamma \vdash_{\mathsf{alg}} (\mathsf{inl}\ \overline{M})_{\_+T_2} : T_1 + T_2; !F'$. We know that $\Gamma; \emptyset \vdash_{\mathsf{alg}} T_2$ by Lemma B.5 and thus $\Gamma \vdash_{\mathsf{alg}} T_2$ by Lemma C.4. The result follows immediately from (VAL INL ALG). We note that $\langle (\mathsf{inl}\ \overline{M})_{\_+T_2} \rangle = \mathsf{inl}\ M$.

*To show:* $\Gamma; \Delta \vdash !F'$. By (!-RIGHT) we know that $\Gamma; !\Delta' \vdash !F'$. The result follows from Lemma B.9.

*Case* (VAL INR): The proof is analogous to the case of (VAL INL).

*Case* (VAL FOLD): $\Gamma; \Delta \vdash \mathsf{fold}\ M : \mu\alpha.T'$ is proved by $\Gamma; !\Delta' \vdash M : T'\{\mu\alpha.T'/\alpha\}$ and $\Gamma; \Delta \hookrightarrow \Gamma; !\Delta'$.

By applying the induction hypothesis we know that there exist $\overline{M}, F'$ such that $\langle \overline{M} \rangle = M$ and $\Gamma \vdash \overline{M} : T'\{\mu\alpha.T'/\alpha\}; F'$ and $\Gamma; !\Delta' \vdash F'$.

*To show:* $\Gamma \vdash_{\mathsf{alg}} \mathsf{fold}\ \overline{M} : \mu\alpha.T'; !F'$. We know that $\Gamma; \emptyset \vdash_{\mathsf{alg}} T_2$ by Lemma B.5 and thus $\Gamma \vdash_{\mathsf{alg}} T_2$ by Lemma C.4. The result follows immediately from (VAL INL ALG). We note that $\langle \mathsf{inl}\ \overline{M} \rangle = \mathsf{inl}\ M$.

*To show:* $\Gamma; \Delta \vdash !F'$. By (!-RIGHT) we know that $\Gamma; !\Delta' \vdash !F'$. The result follows from Lemma B.9.

*Case* (VAL REFINE): $\Gamma; \Delta \vdash M : \{x : T' \mid F'\}$ is proved by $\Gamma; \Delta_1 \vdash M : T'$, $\Gamma; \Delta_2 \vdash F'\{M/x\}$, and $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$.

By induction hypothesis, there exist $\overline{M}, F''$ such that $\langle \overline{M} \rangle = M$, $\Gamma \vdash_{\mathsf{alg}} \overline{M} : T'$, and $\Gamma; \Delta_1 \vdash F''$.

*To show:* $\Gamma \vdash_{\mathsf{alg}} \overline{M}_{\{x:\_ \mid F'\}} : \{x : T \mid F'\}; F'' \otimes F'\{M/x\}$. By Lemma B.5, $\mathit{fnfv}(F') \subseteq \mathit{dom}(\Gamma) \cup \{x\}$. The result follows from (VAL REF ALG). We note that $\langle \overline{M}_{\{x:\_ \mid F'\}} \rangle = M$.

*To show:* $\Gamma; \Delta \vdash F'' \otimes F'\{M/x\}$. The result follows from ($\otimes$-RIGHT) and Lemma B.9.

*Case* (EXP SUBSUM): $\Gamma; \Delta \vdash E : T$ is proved by $\Gamma; \Delta_1 \vdash E : T'$ and $\Gamma; \Delta_2 \vdash T' <: T$ and $\Gamma; \Delta \hookrightarrow \Gamma; \Delta_1, \Delta_2$.

By applying the induction hypothesis we know that there exist $\overline{E}, F'$ such that $\langle \overline{E} \rangle = E$ and $\Gamma \vdash \overline{E} : T'; F'$ and $\Gamma; \Delta_1 \vdash F'$.

Furthermore, by Lemma C.11 we know that there exist $\overline{T'}, \overline{T}, F''$ such that that $T' = \langle \overline{T'} \rangle$, $T = \langle \overline{T} \rangle$, and $\Gamma \vdash_{\mathsf{alg}} \overline{T'} <: \overline{T}; F''$ and $\Gamma; \Delta_2 \vdash F''$. By Lemma C.12 it follows that there exists $\overline{T}^*$ such that that $T = \langle \overline{T}^* \rangle$ and $\Gamma \vdash_{\mathsf{alg}} T' <: \overline{T}^*; F''$

*To show:* $\Gamma \vdash_{\mathsf{alg}} \overline{E}_{\_<:\overline{T}^*} : T; F_1 \otimes F_2$. The result follows immediately from (EXP SUBSUM ALG). We note that $\langle \overline{E} \rangle = E$ as stated above.

*To show:* $\Gamma; \Delta \vdash !F'$. By (!-RIGHT) we know that $\Gamma; !\Delta' \vdash !F'$. The result follows from Lemma B.9.

*Case* (EXP APPL): The proof follows straightforwardly from the induction hypothesis using ($\otimes$-RIGHT) and Lemma B.9.

*Case* (EXP LET): $\Gamma; \Delta \vdash \mathsf{let}\ x = E_1\ \mathsf{in}\ E_2 : T$ is proved by $E_1 \rightsquigarrow^{\emptyset} [\Delta' \mid E_1']$, $\Gamma; \Delta_1 \vdash E_1' : U$, $\Gamma, x : \psi(U); \Delta_2, \mathit{forms}(x : U) \vdash E_2 : T$, $x \notin \mathit{fv}(T)$, and $\Gamma; \Delta, \Delta' \hookrightarrow \Gamma; \Delta_1, \Delta_2$.

By induction hypothesis, there exists $\overline{E_1'}, F_1$ such that $\langle \overline{E_1'} \rangle = E_1'$, $\Gamma \vdash_{\mathsf{alg}} \overline{E_1'} : U; F_1$, and $\Gamma; \Delta_1 \vdash F_1$. By induction hypothesis, there exists $\overline{E_2}, F_2$ such that $\langle \overline{E_2} \rangle = E_2$, $\Gamma, x : \psi(U) \vdash \overline{E_2} : T; F_2$, and $\Gamma, x : \psi(U); \Delta_2, \mathit{forms}(x : U) \vdash F_2$.

We note that by statement (2) of Proposition C.13 it holds that there exists $\overline{E_1}$ such that $\overline{E_1} \rightsquigarrow^{\emptyset} [\Delta' \mid \overline{E_1'}]$.

*To show:* $\Gamma \vdash_{\mathsf{alg}} \mathsf{let}\ x = \overline{E_1}\ \mathsf{in}\ \overline{E_2} : T; \Delta' \multimap (F_1 \otimes \forall x.(\mathit{forms}(x : U) \multimap F_2))$. By Lemma B.5, $\mathit{fnfv}(\Delta_1) \subseteq \mathit{dom}(\Gamma)$. The result follows from (EXP LET ALG). We note that $\langle \mathsf{let}\ x = \overline{E_1}\ \mathsf{in}\ \overline{E_2} \rangle$ is equal to $\mathsf{let}\ x = E_1\ \mathsf{in}\ E_2$.

*To show:* $\Gamma; \Delta \vdash \Delta' \multimap (F_1 \otimes \forall x.(\mathit{forms}(x : U) \multimap F_2))$. By ($\multimap$-RIGHT), $\Gamma, x : \psi(U); \Delta_2 \vdash \mathit{forms}(x : U) \multimap F_2$. By Lemma B.5, $x \notin \mathit{fnfv}(\Delta_2) \subseteq \mathit{dom}(\Gamma)$.

By Lemma C.3, $\Gamma, x : \psi(U); \Delta_2 \vdash \forall x.(\mathit{forms}(x : U) \multimap F_2)$. By ($\otimes$-RIGHT), $\Gamma; \Delta_1, \Delta_2 \vdash F_1 \otimes \forall x.(\mathit{forms}(x : U) \multimap F_2)$. By Lemma B.9, $\Gamma; \Delta, \Delta' \vdash F_1 \otimes \forall x.(\mathit{forms}(x : U) \multimap F_2)$. By ($\multimap$-RIGHT), $\Gamma; \Delta \vdash \Delta' \multimap (F_1 \otimes \forall x.(\mathit{forms}(x : U) \multimap F_2))$.

*Case* (EXP SPLIT): The proof follows a similar strategy as the one for (EXP LET).

*Case* (EXP MATCH): The proof follows a similar strategy as the one for (EXP LET).

*Case* (EXP EQ): The proof follows straightforwardly from applying the induction hypothesis twice and using ($\otimes$-RIGHT) and Lemma B.9.

*Case* (EXP ASSUME): $\Gamma; \Delta \vdash \mathsf{assume}\ F' : T$ is proved by $\Gamma; \Delta, F' \vdash \mathsf{assume}\ \mathbf{1} : T$, where $F' \neq \mathbf{1}$.

We first note that by Lemma C.14 it holds that $\Gamma; \emptyset \vdash \mathsf{assume}\ \mathbf{1} : \mathsf{unit}$ and $\Gamma; \Delta, F' \vdash \mathsf{unit} <: T$.

By combining Lemma C.11 and Lemma C.12 we know that there exist $\overline{T}, F''$ such that that $T = \langle \overline{T} \rangle$, and $\Gamma \vdash_{\mathsf{alg}} \mathsf{unit} <: \overline{T}; F''$ and $\Gamma; \Delta, F' \vdash F''$. By inspection of the algorithmic subtyping rules it follows that $\overline{T}$ must not contain any annotations ($T = \overline{T}$) and thus $\Gamma \vdash_{\mathsf{alg}} \mathsf{unit} <: T; F''$.

By applying the induction hypothesis (see proof of base case (EXP TRUE)) to $\Gamma; \emptyset \vdash \mathsf{assume}\ \mathbf{1} : \mathsf{unit}$ it follows that $\Gamma \vdash_{\mathsf{alg}} \mathsf{assume}\ \mathbf{1} : \mathsf{unit}; \mathbf{1}$ and $\Gamma; \emptyset \vdash \mathbf{1}$.

*To show:* $\Gamma \vdash_{\mathsf{alg}} (\mathsf{assume}\ F')_T : T; F' \multimap (\mathbf{1} \otimes F'')$. We first apply (EXP SUBSUM ALG) to derive that $\Gamma \vdash_{\mathsf{alg}} (\mathsf{assume}\ \mathbf{1})_{\_<:T} : T; \mathbf{1} \otimes F''$.

We know that $\Gamma; \emptyset \vdash_{\mathsf{alg}} T_2$ by Lemma B.5 and thus $\Gamma \vdash_{\mathsf{alg}} T_2$ by Lemma C.4. The result follows from (EXP ASSUME ALG). We note that $\langle (\mathsf{assume}\ F')_T \rangle = \mathsf{assume}\ F'$.

*To show:* $\Gamma; \Delta \vdash F' \multimap (\mathbf{1} \otimes F'')$. As stated above we know that $\Gamma; \emptyset \vdash \mathbf{1}$ and $\Gamma; \Delta, F' \vdash F''$. By ($\otimes$-RIGHT) it holds that $\Gamma; \Delta, F' \vdash \mathbf{1} \otimes F''$

*Case* (EXP RES): The proof follows a similar strategy as the one for (EXP LET).

*Case* (EXP SEND): The proof follows straightforwardly from the induction hypothesis.

*Case* (EXP FORK): The proof follows a similar strategy as the one for (EXP LET).

□