Generic type inference for the ψ-calculi

Eva F. Graversen

Mathias O. Bjerregaard

Department of Computer Science, Aalborg University, Denmark

Summary

In this report we define a type inference algorithm for both a simple and linear type system for the ψ-calculi. ψ-calculi were originally introduced by Bengtson et al. and the type systems originally introduced by Hüttel.

The two type inference algorithms developed in this report are very similar, and follows the same approach: we convert the typing rules for processes into constraint generation rules. We utilise an already existing and decidable fragment of first-order logic as the constraint language, as this ensures that we can always either find a solution to the constraints, or determine that there is no solution. The conversion from typing rules to constraint generation rules is relatively straight forward: instead of using a type environment that assigns types to names, we use an environment assigning type variables to names, and instead of checking the side conditions in the typing rules, we encode them as constraints instead. This way, in order to determine if a process is well-typed, it is enough to find a solution to the constraints, i.e. an assignment of types to type variables.

As the ψ-calculi is just a framework for defining process calculi, it must be instantiated in order to be used. This requires one to define what terms, assertions, and conditions are allowed in the particular ψ-calculus one considers. Consequently, the type systems defined by Hüttel depends on this instantiation, and the typing rules for terms, assertions, and conditions are thus in general undefined; there are however some requirements the rules must adhere to. This implies, as one might expect, that the type inference rules for terms, assertions, and conditions are in general undefined in our type inference algorithm, and must be defined together with the instantiation of the type system. The rules can in most cases be constructed by a similar conversion as the one we conducts for the typing rules for processes. Defining the constraint generation rules for terms, assertions, and conditions are however not enough. The logic we have chosen to utilise in this report, requires one to define the axioms by which the constraints can be manipulated. In other words, relations have no inherent meaning, and we must define the meaning of the relations through a fixed set of constraints. We refer to these constraints as axioms. An example of such an axiom could be

$$\forall a : \forall b : \forall c : ((\mathsf{Eq}(a, b) \wedge \mathsf{Eq}(b, c)) \Rightarrow \mathsf{Eq}(a, c))$$

which ensures the relation Eq becomes transitive. The axioms are required to adhere to some simple requirements, in order to ensure that the type inference is correct. The definition of these axioms is entirely dependent of which relations one chooses to utilise for ones type system, and therefore they must be defined from scratch each time. One could argue that we could have predefined for instance an equivalence relation to represent equality with the required axioms, as this would always be the same. However such a relation is trivial and it would provide no notable benefits to have such a relation predefined.

We can now summarise the steps requires to instantiate the type inference algorithm:

(1) Instantiate a suitable ψ-calculus
(2) Instantiate the type system as required
(3) Derive the constraint generation rules for terms, assertions, and conditions
(4) Define the required relations and axioms

After the above steps have been completed, one have a type inference algorithm that is proved to be correct.

Contents

CHAPTER 1

Introduction

Since the introduction of the CCS [17] and π-calculus [18] by Milner et al., process calculi have been a fundamental tool to describe and reason about parallel and distributed systems. Since then, many versions of the π-calculus have been developed. For instance the spi calculus [24] was developed by Abadi and Gordon with the purpose of describing cryptography protocols and the distributed π-calculus [23], $D\pi$, was developed by Hennessy and Riely in order to reason about dynamically evolving networks, where processes can migrate between locations. In [7] Gardner and Wischik introduce a π-calculus that captures the fusion of names i.e. when a process like $a!x.P \mid a?y.Q$ communicate we conduct the substitution $[x/y]Q$, which is the same as stating that henceforth, for $P \mid Q$, we have $x = y$. These equalities thus becomes part of the process syntax, but have no semantics by themselves. In addition many papers introduce their own version of such process calculi, tailored specifically for the problem setting relevant to the paper. While this approach of developing a new process calculus for every new problem setting does provide some benefits, it also introduces an unnecessary overhead: the syntax, semantics, semantic equivalence, and axiomatization of equivalence must be defined every time, despite being similar to the myriad of other derivations of e.g. the π-calculus. In 2009 Bengtson et al. formulated a new group of process calculi called the ψ-calculi [2]. The ψ-calculi is a framework characterising many of the common aspects of the many process calculi previously defined, where every ψ-calculus shares the same basic syntax and semantics for processes, but the terms, assertions, and conditions used can be unique for each instantiation of the framework. Unlike the π-calculus, ψ-calculi are allowed to use more complex structures then just names for e.g. channels, and the assertions and conditions allows one to make complex reasoning, e.g. in a case-structure similar to the programming languages of the C-family. This allows one to define a ψ-calculus as required by the setting in which the calculus is to be utilised, without having to provide the syntax and semantics of the full calculus, but instead only the definition of terms, assertions, and conditions. In addition one get the notion of semantic equivalence, axiomatisation of equivalence, and other properties proved for the ψ-calculi for free.

While process calculi provide a desirable way to define parallel and distributed systems, they provide no means of reasoning about properties of such systems. For this purpose type systems have been commonly used. One of the first type systems for process calculi was developed by Milner [18] for the π-calculus; the idea here was to ensure only names of the correct type could be send along each channel, thus preventing one from sending e.g. numbers on channels meant for strings. Since then a myriad of type systems have been developed: In [6] Fournet, Gordon, and Maffeis define a type system, based on the one described by Gordon and Jeffrey in [10], for the spi calculus which ensures that all expectations are justifiable i.e. if a processes at some point expect X to hold, then we know that we have the sufficient evidence for this; and in [15] Kobayashi et al. introduce the concept of linearity to the π-calculus: channels may be linear meaning they can and must only be used once, or channels can be unlimited and used arbitrarily (including never). This allows us to, for instance, ensure that some specific communication takes place, and that it only happens once.

Unfortunately, just as for process calculi, for each of these type systems one must also prove a common set of theorems, like subject reduction and some form of safety. Moreover each of these type systems was developed for a particular incarnation of some process

calculus, and if we would like to combine them for some common process calculus, we would have to redefine and re-prove each theorem again. In [16] König defined a generic type system for analysing input/output capabilities in the π-calculus; and in [14] Igarashi and Kobayashi define a generic type system for a variation of the π-calculus that uses an abstraction of the process as its environment and can be used for, among other things, deadlock detection and race detection. Utilising the ψ-calculi as defined by Bengtson et al., Hüttel defined two generic type systems for all ψ-calculi in [12, 13], the first concerning simple type systems a la [18], the second for resource dependent type systems a la [15]. Just like the ψ-calculi framework, these type systems are partially undefined, e.g. the typing rules for terms depends on the definition of terms in the particular ψ-calculus, and must thus be defined together. However, as long as these definitions satisfy certain well-defined properties subject reduction is guaranteed.

While type systems are beneficial for their ability to ensure that certain properties holds, they do require the user to explicitly define the type of each term in order to determine if a process is well-typed. This may be doable for small example systems, but quickly becomes impractical for larger parallel or distributed systems. The automation of defining the types for each term is called type inference or type reconstruction, and this is a well-studied area. For instance most programming languages with types provides some semblance of type inference, as no programmer wants to define and write the type of each term in their program.

In general one can split type inference algorithms into three categories: One is to create an original algorithm from scratch, for instance in [22] Palsberg and Schwartzbach utilise a trace graph to generate constraints, instead of the more common approach of using syntax-directed rules; and Flanagan et al. conduct their inference in [5] using a minimal assignment they iteratively increase until either a solution or contradiction is found. Another approach is to generate constraints in a known logic, and using a solver for said logic to find a solution. This can be seen in [9, 11], where Hüttel et al. conduct type inference for the type system defined in [6] by creating constraints in the alternating-fixed-point-logic. The final approach is to translate the problem into another type system, conduct inference there, and translate the solution back, as seen in [21], where Padovani conducts type inference on session types by translating them into linear types.

However, just as for the correctness of the type systems, each of these different type inference approaches is required to provide proofs of correctness. Since Bengtson et al. and Hüttel successfully defined frameworks capable of encompassing many different previously defined process calculi and type systems, the question thus becomes whether or not one can define a general framework for type inference building on the same ideas. If successful, one would have a complete process calculi framework, i.e. a framework for processes, a framework for type systems, and a framework for type inference, and thus removing some of the otherwise tedious, but required work, involved when defining a processes calculus, type system, and type inference algorithm. Moreover, exchanging ideas and/or incorporating ideas from other type systems becomes easier, as they will all share common basic definitions and typing rules. To achieve this one has to consider what type inference approach would be applicable. König gives a very general type inference algorithm for her generic type system in [16], but does not provide any details about how to accomplish each step; and in [14] Igarashi and Kobayashi provide a set of constraint generation rules for their type system, that require additional rules to be defined for each instantiation, though they do not provide a strategy for solving these constraints. For a general type system, following their example and creating constraints, but making them easy to solve by doing so in a known logic seems like the most sensible way to proceed. It is difficult to create an new unique algorithm with large parts missing, and likely even harder to find a way to instantiate the gaps in such an algorithm. And to come up with another non-generic type system, that any instantiation could be translated into with relative ease, would likely be impossible. On the other hand, we can generate constraints in a know

logic by rules similar to the typing rules, as long as one manages to find a way to represent the types and their relations in the constraint language.

Contribution and structure. In this report we define a type inference approach for the type systems defined in [12, 13] for the ψ-calculi. To our knowledge this is the first proposed algorithm for type inference for these type systems, and specifically:

- We propose a framework for type inference for both simple and linear type systems in the ψ-calculi
- We prove that the proposed framework is sound
- We provide example instantiations of the type inference framework
- We provide examples of how to use the instantiated framework for type inference
- We highlight some of the problematic aspects of more advanced type systems with regards to the proposed framework

The report is structured as follows: in Chapter 2 we introduce the ψ-calculi and ALFP; in Chapters 3 and 4 we describe type inference for two generic type systems in the following way: in Section 3.1 we introduce the type system defined in [12]; in Section 3.2 we define the constraint generation and type inference for this type system; in Sections 3.3 and 3.4 we introduce two example encodings of previous type systems in the ψ-calculi and conduct type inference for those; in Section 4.1 we introduce the type system of [13]; in Section 4.2 we describe its constraint generation and inference; in Section 4.3 we introduce an example encoding of a previous type system in the ψ-calculi and conduct type inference for this; in Section 4.4 we look at a failed attempt at instantiating the linearly typed ψ-calculi to ensure termination in value-passing processes and make our own attempt at correcting it and generating constraints for it; and finally we present our conclusion in Chapter 5.

CHAPTER 2

Preliminaries

Before presenting our work, we will use this chapter to describe some important concepts, namely ψ-calculi in Section 2.1 and alternation-free least fixpoint logic in Section 2.2.

2.1. ψ-calculi

ψ-calculi, as presented in [12] generalise variants of the π-calculus. A ψ-calculus has a set of processes, ranged over by $P, Q, \ldots$, containing occurrences of terms, ranged over by $M, N, \ldots$. Both processes and terms can contain names. The syntax of processes can be seen in Table 2.1.1.

The syntax mostly resembles the π-calculus, with a few key differences: Rather than a channel name, M in an output or input prefix can be an arbitrary term. Furthermore, in the input construct $\underline{M}(\lambda\vec{x})N.P$, $(\lambda\vec{x})N$ is a pattern, where the variable names $\vec{x}$ can occur free in N and P. Any N' received on channel M must match the pattern $(\lambda\vec{x})N$, meaning that it must be possible to get N' by instantiating the variable names $\vec{x}$ in N with terms.

We also introduce the concept of a nominal set, meaning a set whose members can be affected by names being bound or swapped. When x is a member of a nominal set and a is a name, we use $a\#x$ to denote that a is fresh for x. We refer to a nominal set with internal structure as a nominal datatype. We use term substitution—$X[\vec{x} := \vec{Y}]$ means that the terms of $\vec{Y}$ substitute the names of $\vec{x}$ in X—in the nominal datatypes of ψ-calculi, shown in Table 2.1.2. We also have a number of operations on the nominal datatypes, seen in Table 2.1.3, which must be instantiated for each ψ-calculus.

We let $\mathsf{names}(P)$ describe the names of a process P and $\mathsf{fn}(P)$ describe the free names of P.

We also introduce some requirements and definition related to the assertions. We introduce the concept of equality for assertions denoted $\simeq$, and in addition introduce some requirement for composition of assertions as defined in Definition 2.1.1.

$P ::=$	$\underline{M}(\lambda\vec{x})N.P$	input
	$\overline{M}\,N.P$	output
	$P_1 \mid P_2$	parallel composition
	$(\nu n : T)P$	restriction of name n
	$*P$	replication
	case $\sigma_1 : P_1, \ldots, \sigma_k : P_k$	conditional
	$(\!\vert\Psi\vert\!)$	assertion process

TABLE 2.1.1. The syntax of the ψ-calculi

Nominal data type		Variables
T	terms	M, N
C	conditions	σ
A	assertions	Ψ

TABLE 2.1.2. Nominal data types

$\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$	composition of assertions
$\leftrightarrow : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$	channel equivalence
$\mathbf{1} \in \mathbf{A}$	unit assertion
$\vDash \subseteq \mathbf{A} \times \mathbf{C}$	entailment

TABLE 2.1.3. Operations on nominal data types

DEFINITION 2.1.1 (Assertions equivalence ($\simeq$)). For any assertion Ψ we have $\Psi \simeq \Psi$. For composition of assertions, the composition must adhere to the following constraints:

$$\Psi_1 \otimes \Psi_2 \simeq \Psi_2 \otimes \Psi_1 \qquad \Psi_1 \otimes (\Psi_2 \otimes \Psi_3) \simeq (\Psi_1 \otimes \Psi_2) \otimes \Psi_3$$
$$\Psi \otimes \mathbf{1} \simeq \Psi \qquad \Psi \simeq \Psi' \Rightarrow \Psi \otimes \Psi_1 \simeq \Psi' \otimes \Psi_1$$

In addition we say that an assertions Ψ is idempotent if $\Psi \otimes \Psi \simeq \Psi$

Since we in general allow assertions to be combined, it is only natural to also introduce an ordering of assertions. We define the ordering of assertions as seen in Definition 2.1.2. The definition is rather straightforward, and should not be surprising.

DEFINITION 2.1.2 (Ordering of assertions). We write $\Psi_1 \leq \Psi_2$ if there exists a Ψ such that $\Psi_1 \otimes \Psi \simeq \Psi_2$

Since we consider composition of assertions, we ultimately also consider decomposition of assertions. These two notions are however not enough, and we additionally define the assertion exclusion operator $\div$ in Definition 2.1.3. This operator is representing the removal of any and all sub-assertions Ψ_i in an composite assertion Ψ that involves a given set of names. If the assertions is not composite or if all sub-assertions involve the given set of names, the unit assertion $\mathbf{1}$ is the result.

DEFINITION 2.1.3 (Assertion exclusion). For every assertion Ψ and name set of names L we assume the existence of a largest sub-assertion $\Psi \div L \leq \Psi$ not containing any occurrences of names in L

The assertion information of ψ-calculi processes can be extracted as its frame $\mathcal{F}(P) = \langle E_P, \Psi_P \rangle$, where E_P records the types of the names local to P and Ψ_P records the assertions of P. These are called qualified frames. Definition 2.1.4 shows how to find the frame of a process. We use $\mathcal{F}(P) \otimes \mathcal{F}(Q)$ to mean $\langle E_P, E_Q, \Psi_P \otimes \Psi_Q \rangle$ when $\mathrm{dom}(E_P) \# \mathrm{dom}(E_Q)$, $\mathrm{dom}(E_P) \# \Psi_Q$, and $\mathrm{dom}(E_Q) \# \Psi_P$, and use $(\nu n : T)\mathcal{F}(P)$ to mean $\langle n : T, E_P, \Psi_P \rangle$.

DEFINITION 2.1.4 (Frame of a process).

$$\mathcal{F}(P \mid Q) = \mathcal{F}(P) \otimes \mathcal{F}(Q) \qquad \mathcal{F}((\nu n : T)P) = (\nu n : T)\mathcal{F}(P)$$
$$\mathcal{F}(\langle\!|\Psi|\!\rangle) = \langle \epsilon, \Psi \rangle \qquad \mathcal{F}(P) = 1 \text{ otherwise}$$

We use labelled semantics to describe ψ-calculi. Our labelled transitions are of the form $\Psi \lhd P \xrightarrow{\alpha} P'$, where the label α is defined by the following rules:

$$\alpha ::= \tau \mid \underline{M}N \mid \overline{M}N \mid (\nu \vec{b} : \vec{T})\overline{M}N.$$

Lastly we define the function bn, defined as $\mathrm{bn}((\nu \vec{b} : \vec{T})\overline{M}N) = \vec{b}$ and for any other α $\mathrm{bn}(\alpha) = \emptyset$ and present the the labelled semantic rules in Table 2.1.4.

[In]	$\dfrac{\Psi \vDash M \dot{\leftrightarrow} K}{\Psi \lhd \underline{M}(\lambda \vec{x})N.P \xrightarrow{\underline{K}N[\vec{x}:=\vec{L}]} P[\vec{x} := \vec{L}]}$	[Out]	$\dfrac{\Psi \vDash M \dot{\leftrightarrow} K}{\Psi \lhd \overline{M}\, N.P \xrightarrow{\overline{K}N} P}$
[Com]	$\dfrac{\begin{array}{c}\Psi_Q \otimes \Psi \lhd P \xrightarrow{\overline{M}(\nu \vec{a}:\vec{T})N} P' \\ \Psi_p \otimes \Psi \lhd Q \xrightarrow{\underline{K}N} Q' \\ \Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \dot{\leftrightarrow} K\end{array}}{\Psi \lhd P \mid Q \xrightarrow{\tau} (\nu \vec{a} : \vec{T})(P' \mid Q')}\ \vec{a} \# Q$	[Case]	$\dfrac{\Psi \lhd P_i \xrightarrow{\alpha} P' \qquad \Psi \vDash \phi_i}{\mathbf{case}\ \tilde{\sigma} : \tilde{P} \xrightarrow{\alpha} P'}$
[Par]	$\dfrac{\Psi_Q \otimes \Psi \lhd P \xrightarrow{\alpha} P'}{\Psi \lhd P \mid Q}\ \mathrm{bn}(\alpha) \# Q$	[Scope]	$\dfrac{\Psi \lhd P \xrightarrow{\alpha} P' \qquad b \# a, \Psi}{\Psi \lhd (\nu b : T)P \xrightarrow{\alpha} (\nu b : T)P'}$
[Open]	$\dfrac{\Psi \lhd P \xrightarrow{\overline{M}(\nu \vec{a}:\vec{T})N} P'}{\Psi \lhd (\nu b : T_1)P \xrightarrow{\overline{M}(\nu \vec{a}:\vec{T} \cup \{b:T_1\})N} P'}$ $b \# a, \Psi, M$ and $b \in \mathsf{names}(N) \cup \mathsf{names}(\vec{T}) \cup \mathsf{names}(T_1)$	[Rep]	$\dfrac{\Psi \lhd P \mid *P \xrightarrow{\alpha} P'}{\Psi \lhd *P \xrightarrow{\alpha} P'}$

TABLE 2.1.4. Labelled sematics of the ψ-calculi

2.2. Alternation-free least fixpoint logic

We make use of the *alternation-free least fixpoint logic* (ALFP) as defined in [20] to formulate the constraints used to conduct type inference. We therefore briefly present the syntax of this logic. We assume a fixed countable set of variables $\mathbb{X}$ ranged over by $x, \ldots$ and a finite ranked alphabet of predicate symbols $\mathbb{R}$ ranged over by $R, \ldots$. The set of clauses, C, is given by Table 2.2.1.

We utilise the ALFP for our constraint solving, as it constitutes a decidable fragment of first-order logic with already existing solvers [19]. This allows us to focus solely on the definition and generation of constraint adhering to the rules of ALFP, as the solution to such constraints can be found using existing tools.

In the remainder of this paper, we will utilise the Succinct Solver as developed in [20, 19] for solving our ALFP constraints. In Appendices A.1 to A.3 we provide the Succinct Solver's solutions to the examples presented in Sections 3.4.3, 4.3.1 and 4.4.2. We do not present any examples of the Succinct Solver encoding of the generated constraints, as this

$P ::=$		Pre-condition
	$R(x_1, \ldots, x_k)$	predicate
	$\neg R(x_1, \ldots, x_k)$	negation of predicate
	$P_1 \wedge P_2$	conjunction
	$P_1 \vee P_2$	disjunction
	$\exists x :\ P$	existential quantification
	$\forall x :\ P$	universal quantification
$C ::=$		Clause
	$R(x_1, \ldots, x_k)$	predicate
	$\mathbf{1}$	true
	$C_1 \wedge C_2$	conjunction
	$P \Rightarrow C$	implication
	$\forall x :\ C$	universal quantification

TABLE 2.2.1. The syntax of the ALFP

encoding is straight forward and follows many of the known patterns from programming languages. For instance we encode $P \wedge Q$ as "`P & Q`", $P \Rightarrow Q$ as "`P => Q`", $\forall x : P$ as "`A x.(P)`", and so forth. The translation from the constraint presented in the examples to the constraints given to the Succinct Solver is thus trivial.

REMARK. According to the Succinct Solver manual [25] and [19] the Succinct Solver V2.0 is not compatible with pre-conditions of the form $\forall x : P$. This implies that constraints like $\forall x : (\forall y : P(y, x)) \Rightarrow Q(x)$ are not solvable by the Succinct Solver V2.0.

CHAPTER 3

Simply typed ψ-calculi

In this chapter we introduce the first of our two generic type systems, simply typed ψ-calculi, in Section 3.1; present our type inference for said type system in Section 3.2; and present two examples of instantiations of the type system and inference in Sections 3.3 and 3.4.

3.1. A type system for a simply typed ψ-calculi

We introduce the generic type system for all ψ-calculi developed in [12]. This is a simply typed type system, resembling the earlier type systems developed for e.g. the π-calculus. A notable difference between this type system and most others is the generality of the type system. The type system only partially defines the required typing rules, and utilises undefined side conditions for some rules. While this might seem problematic it is the consequence of the generality of the ψ-calculi. The type system must thus be instantiated together with the instantiation of the particular ψ-calculus in question. The instantiation of the type system requires adding typing rules for terms, assertions and conditions; defining the binary relation $\leftharpoonup$ for types; and defining the required side conditions in the compositionality rules. For instance if one wanted to create a ψ-calculus to model the π-calculus, see [2], and chose to instantiate this type system to model a simply typed π-calculus, e.g. the type system used in [8], one would be required to define the typing rules for terms, which in this case would simply be names, the $\leftharpoonup$ relation, and the side conditions for compositionality. The side conditions would be mostly trivial as composition is not utilised and the typing rules for assertions would not be required, as the simple typed π-calculus does not utilise such constructs.

We now proceed to present the type system. Firstly we consider the typing contexts, or environments, used by this type system. The environments are defined in Definition 3.1.1, and consists of type annotations or bindings of names of the form $x : T$ and assertions of the form Ψ.

Definition 3.1.1 (Environments and well-formedness). Let environments, Γ, be defined as

$$\Gamma ::= \Gamma, x : T \mid \Gamma, \Psi \mid \emptyset.$$

We say an environment Γ is well-formed, denoted $\Gamma \vdash \diamond$ if it is a partial function from names to types such that if $\Gamma = \Gamma_1, \Psi, \Gamma_2$ and x is a name in Ψ then $x \in \mathsf{dom}(\Gamma_1)$.

In addition to the definition of environments we define extension of environments, both in terms of bindings and assertions as seen in Definitions 3.1.2 and 3.1.3. We combine the two notions of extension as seen in Definition 3.1.4.

Definition 3.1.2 (Binding extension ($<_T$)). Let Γ_1 and Γ_2 be environments such that $\Gamma_1 \vdash \diamond, \Gamma_2 \vdash \diamond, \Gamma_1 = \Gamma_{10}, \ldots, \Gamma_{1(k+1)}$, and $\Gamma_2 = \Gamma_{10}, u_1 : T_1, \Gamma_{11}, u_2 : T_2, \ldots, u_k : T_k, \Gamma_{1(k+1)}$ for some $u_i : T_i$. We then say Γ_2 extends Γ_1 with additional bindings, denoted $\Gamma_1 <_T \Gamma_2$

Definition 3.1.3 (Assertion extension ($<_A^0$)). Let Γ_1 and Γ_2 be environments such that $\Gamma_2 = \Gamma_1, \Psi$. We say Γ_2 extends Γ_1 with an additional assertion, denoted $\Gamma_1 <_A^0 \Gamma_2$. In addition we let $<_A$ be the transitive closure of $<_A^0$

Definition 3.1.4 (Environment extension ($<$)). We let the relation $<$ denote the least preorder containing $<_T \cup <_A$

[Comp-Ast]	$\dfrac{\Gamma \vdash \Psi_1 \qquad \Gamma \vdash \Psi_2}{\Gamma \vdash \Psi_1 \otimes \Psi_2}$
[Comp-Cond]	$\dfrac{\Gamma \vdash \sigma_i \qquad 1 \leq i \leq k \qquad \mathcal{X}}{\Gamma \vdash g(\sigma_1, \ldots, \sigma_k)}$
[Comp-Term]	$\dfrac{\Gamma \vdash M_i : T_i \qquad 1 \leq i \leq k \qquad \mathcal{X}}{\Gamma \vdash f(M_1, \ldots, M_k) : T}$

TABLE 3.1.1. Compositionality typing rules

We now introduce the judgements for the type system. Since we have three types of nominal data in the ψ-calculi, we also have three types of judgements as defined in Definition 3.1.5.

DEFINITION 3.1.5 (Type judgements). Let judgements, $\mathcal{J}$, be defined as

$$\mathcal{J} ::= M : T \mid \sigma \mid \Psi,$$

namely one judgement for each of the three nominal data types in the ψ-calculi. Moreover we say $\mathcal{J}_\Gamma$ is a *qualified judgement* of the form $\Gamma \vdash \mathcal{J}$ if $\mathsf{names}(\mathcal{J}) \subseteq \mathsf{dom}(\Gamma)$

As with environments we introduce the notion of extension for qualified judgements as defined in Definition 3.1.6. Note that the extension of qualified judgements require the actual judgement to be the same, but allows for an extension of the corresponding environment.

DEFINITION 3.1.6 (Judgement extension). Let $\mathcal{J}^1_{\Gamma_1}$ and $\mathcal{J}^2_{\Gamma_2}$ be qualified judgements such that $\Gamma_1 < \Gamma_2$ and $\mathcal{J}^1 = \mathcal{J}^2$. We say $\mathcal{J}^2_{\Gamma_2}$ extends $\mathcal{J}^1_{\Gamma_1}$ denoted $\mathcal{J}^1_{\Gamma_1} < \mathcal{J}^2_{\Gamma_2}$

Before we present the typing rules we present the compositionality rules as seen in Table 3.1.1. These rules must be adhered to by the type system instantiating. In the compositionality rules $\mathcal{X}$ denotes any additional side condition one would like to hold. These side conditions must however satisfy three conditions: (1) the side conditions must be monotone with regards to environment extensions, (2) the side conditions must be topical, and (3) assertion typability must respect composition of environment assertions. We refer to [12] for clarifications.

We finally present the typing rules in Table 3.1.2. It is worth mentioning again, that there are no typing rules for either assertions or terms. These must be defined when instantiating the type system. Although it must hold in any instantiation that $\Gamma \vdash x : T$ if $\Gamma(x) = T$.

$$[\mathsf{In}]\quad \frac{\Gamma, \vec{x} : \vec{T} \vdash P \qquad \Gamma \vdash (\lambda \vec{x}) N : \vec{T} \to U_o \qquad \Gamma \vdash M : U_s \qquad U_s \leftrightarrows U_o}{\Gamma \vdash \underline{M}(\lambda \vec{x}) N.P}$$

$$[\mathsf{Out}]\quad \frac{\Gamma \vdash M : T_s \qquad \Gamma \vdash N : T_o \qquad \Gamma \vdash P \qquad T_s \leftrightarrows T_o}{\Gamma \vdash \overline{M}\, N.P}$$

$$[\mathsf{Par}]\quad \frac{\Gamma, \Gamma_2, \Psi_2 \vdash P_1 \qquad \Gamma, \Gamma_1, \Psi_1 \vdash P_2 \qquad \mathcal{F}(P_1) = \langle \Gamma_1, \Psi_1 \rangle \qquad \mathcal{F}(P_2) = \langle \Gamma_2, \Psi_2 \rangle}{\Gamma \vdash P_1 \mid P_2}$$

$$[\mathsf{Res}]\quad \frac{\Gamma, x : T \vdash P}{\Gamma \vdash (\nu x : T) P} \qquad [\mathsf{Rep}]\quad \frac{\Gamma \vdash P}{\Gamma \vdash *P} \qquad [\mathsf{Ast}]\quad \frac{\Gamma \vdash \Psi}{\Gamma \vdash (\!|\Psi|\!)}$$

$$[\mathsf{Case}]\quad \frac{\Gamma \vdash \sigma_i \qquad \Gamma \vdash P_i \qquad 1 \leq i \leq k}{\Gamma \vdash \mathbf{case}\ \sigma_1 : P_1, \ldots, \sigma_k : P_k} \qquad [\mathsf{Pat}]\quad \frac{\Gamma, \vec{x} : \vec{T} \vdash M : U}{\Gamma \vdash (\lambda \vec{x}) M : \vec{T} \to U}$$

TABLE 3.1.2. Typing rules for the simply typed ψ-calculi

3.2. Constraint generation for simply typed ψ-calculi

In this section we define the constraints generation for the type system presented in Section 3.1. The constraint generation is inspired by [9], and follows many of the ideas proposed in this paper. This section only defines the constraints imposed by the uninstantiated type system, and thus only provides a subset of the total required constraints. In addition the constraints generated in this section all adhere to the definition of clauses in ALFP (see Section 2.2), and the question of satisfiability is thus decidable.

We let $[\![C]\!]$ denote the encoding of term C in some formalism F to term C' in target formalism F' such that the meaning of C and C' is the same. In Tables 3.2.1 to 3.2.3 we will use this notation to express the ALFP clause encoding of some constraints. The primary utility of this convention is to reduce the otherwise unnecessary verbosity of simple constraints, and enable us to generate constraints whose meaning is only later defined.

The constraint generation for processes can be seen in Table 3.2.1. The constraint generation for processes is of the form $\Gamma \vdash P \rightsquigarrow \phi$ where ϕ is a constraint, created as a conjuction of many smaller constraints, that must be satisfied in order for Γ to type the

$$[\mathsf{Par}]\quad \frac{\Gamma, \mathcal{F}(P_2) \vdash P_1 \rightsquigarrow \phi_1 \qquad \Gamma, \mathcal{F}(P_1) \vdash P_2 \rightsquigarrow \phi_2}{\Gamma \vdash P_1 \mid P_2 \rightsquigarrow \ \phi_1 \wedge \phi_2}$$

$$[\mathsf{Rep}]\quad \frac{\Gamma \vdash P \rightsquigarrow \phi}{\Gamma \vdash *P \rightsquigarrow \phi} \qquad [\mathsf{Ass}]\quad \frac{\Gamma \vdash \Psi \rightsquigarrow \phi}{\Gamma \vdash (\!|\Psi|\!) \rightsquigarrow \phi} \qquad [\mathsf{Res}]\quad \frac{\Gamma, x : \tau \vdash P \rightsquigarrow \phi_p}{\Gamma \vdash (\nu x : \tau) P \rightsquigarrow \ \phi_p}$$

$$[\mathsf{Out}]\quad \frac{\Gamma \vdash M \rightsquigarrow \tau_m; \phi_m \qquad \Gamma \vdash N \rightsquigarrow \tau_n; \phi_n \qquad \Gamma \vdash P \rightsquigarrow \phi_p}{\Gamma \vdash \overline{M}\, N.P \rightsquigarrow \phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\tau_m \leftrightarrows \tau_n]\!]}$$

$$[\mathsf{In}]\quad \frac{\Gamma, \vec{x} : \vec{\tau} \vdash P \rightsquigarrow \phi_p \qquad \Gamma \vdash (\lambda \vec{x}) N \rightsquigarrow (\vec{\tau} \to \tau_u); \phi_n \qquad \Gamma \vdash M \rightsquigarrow \tau_m; \phi_m}{\Gamma \vdash \underline{M}(\lambda \vec{x}) N.P \rightsquigarrow \ \phi_p \wedge \phi_n \wedge \phi_m \wedge [\![\tau_m \leftrightarrows \tau_u]\!]}$$

$$[\mathsf{Cas}]\quad \frac{\Gamma \vdash \sigma_i \rightsquigarrow \phi_{si} \qquad \Gamma \vdash P_i \rightsquigarrow \phi_{pi} \qquad 1 \leq i \leq k}{\Gamma \vdash \mathbf{case}\ \sigma_1 : P_1, \ldots, \sigma_k : P_k \rightsquigarrow \bigwedge_{i=1}^{k} \phi_{si} \wedge \phi_{pi}}$$

TABLE 3.2.1. Constraint generation for processes

[Pat]	$\dfrac{\Gamma, \vec{x} : \vec{\tau} \vdash M \rightsquigarrow \tau_m; \phi_m}{\Gamma \vdash (\lambda \vec{x}) M \rightsquigarrow (\vec{\tau} \to \tau_m);\ \phi_m}$
[Var]	Must be defined together with the type system
[Ast]	Must be defined together with the type system
[Lal]	$[\![\tau_1 \leftrightarrow \tau_2]\!]$ Must be defined together with the type system

TABLE 3.2.2. Constraint generation for terms

[Comp-Ast]	$\dfrac{\Gamma \vdash \Psi_1 \rightsquigarrow \phi_1 \qquad \Gamma \vdash \Psi_2 \rightsquigarrow \phi_2}{\Gamma \vdash \Psi_1 \otimes \Psi_2 \rightsquigarrow \phi_1 \wedge \phi_2}$
[Comp-Cond]	$\dfrac{\Gamma \vdash \sigma_i \rightsquigarrow \phi_i \qquad 1 \leq i \leq k \qquad \mathcal{X} \rightsquigarrow \phi_{\mathcal{X}}}{\Gamma \vdash g(\sigma_1, \ldots, \sigma_k) \rightsquigarrow (\bigwedge_{i=1}^{k} \phi_i) \wedge \phi_{\mathcal{X}}}$
[Comp-Term]	$\dfrac{\Gamma \vdash M_i \rightsquigarrow \tau_i; \phi_i \qquad 1 \leq i \leq k \qquad \mathcal{X} \rightsquigarrow \phi_{\mathcal{X}}}{\Gamma \vdash f(M_1, \ldots, M_k) \rightsquigarrow \tau; (\bigwedge_{i=1}^{k} \phi_i) \wedge \phi_{\mathcal{X}}}$

TABLE 3.2.3. Constraint generation for compositionality

process P. We make use of the function $\mathcal{F}$ in the constraint generation, but as the result of this function can be computed statically, it introduces no additional complexity.

The constraint generation for terms, as one might expect, is largely undefined. The constraint generation is of the form $\Gamma \vdash M \rightsquigarrow \tau; \phi$ where τ is the type variable assigned to the term M and ϕ is the the conjunction of constraints, that must be satisfied in order for Γ to type check the expression $M : \tau$.

REMARK. The Γ used in $\Gamma \vdash P \rightsquigarrow \phi$ should be decided for the individual type system, but should always use type variables τ in stead of actual types.

The only rule that is independent of the instantiation of the type system is the [Pat] rule as defined in Table 3.2.2. The table also includes the two remaining term rules that must be defined, namely [Var] and [Ast]. It is worth mentioning that the instantiation of these rules may lead to several rules for both terms and assertions, but we will refer to these collections of rules as [Var] and [Ast] respectively. In addition, since the meaning of the constraint $T_1 \leftrightarrow T_2$ depends on the instantiated type system, the encoding of this constraint depends on the instantiation as well. For this reason the encoding of the constraint $T_1 \leftrightarrow T_2$ is left undefined and must be provided together with the instantiated type system. As for the constraint generation rules, the encoding must adhere to rules for clauses for ALFP as defined in Section 2.2.

The last part of the constraint generation is related to compositionality of terms and assertions as seen in Table 3.2.3. The $\mathcal{X} \rightsquigarrow \phi_{\mathcal{X}}$ part of both [Comp-Cond] and [Comp-Term] refers to any additional side conditions one would require. These are thus instantiable. For instance in [Comp-Term] one would most likely add a side condition relating the type variables τ_i from the premise with the type variable τ in the conclusion.

While the generated constraints serves as a starting point, they do not explain how to manipulate the relations. Despite what we may or may not call our relations, they have no inherent meaning in ALFP, and all of the common logical deductions we normally apply are nonexistent. In other words, simply presenting a conjunction of predicates does not lead to the desired solution. Consider for instance a case where we have $[\![\tau_1 \leftrightarrow \tau_2]\!] \wedge [\![\tau_1 \leftrightarrow \tau_3]\!]$. In many type systems one would be able to make the conclusion that $\tau_2 = \tau_3$, since a channel must send/receive objects of the same type. But as we have already mentioned,

the constraint $[\![\tau_1 \leftarrowtriangle \tau_2]\!]$ does not state this; it merely state something must be true between τ_1 and τ_2. Assuming we deduce that $\tau_2 = \tau_3$, we now have to represent $\tau_2 = \tau_3$. Clearly we could make a predicate $\mathsf{Eq}(\tau_2, \tau_3)$, but again the predicate have no inherent meaning—despite its name—and we have no way of determining if for instance $P(\tau_3)$ should hold if $P(\tau_2)$ holds. The solution to this, is to introduce a set of axioms that explains how to make logical and correct deductions based on the generated constraints. While this might seem tedious it is not without advantage: we are free to determine exactly what deductions can be made, and we are thus free to disallow otherwise common deductions in specific cases if it should not be allowed for this case. These axioms would have to adhere to the conditions introduced by ALFP in order to be solvable. Using this approach we can express the above deductions as axioms in ALFP giving us

$$\forall\tau_1 : \forall\tau_2 : \forall\tau_3 : ([\![\tau_1 \leftarrowtriangle \tau_2]\!] \wedge [\![\tau_1 \leftarrowtriangle \tau_3]\!]) \Rightarrow \mathsf{Eq}(\tau_2, \tau_3)$$

and

$$\forall\tau_1 : \forall\tau_2 : (P(\tau_1) \wedge \mathsf{Eq}(\tau_1, \tau_2)) \Rightarrow P(\tau_2).$$

In addition to the idea of axioms we introduce a special relation called Fail. This special relations is meant to represent failure: if a solution contains no Fail-relations it is a real solution, but if it contains any Fail-relations the solution is in fact not a solution, but a failure. This relation can be useful in cases where we have $[\![\tau_1 \leftarrowtriangle \tau_2]\!] \wedge [\![\tau_2 \leftarrowtriangle \tau_1]\!]$. Assuming we get these constraints for a simple type system with channels, it would represent the fact that $T_1 = \mathsf{Ch}(T_2)$ and $T_2 = \mathsf{Ch}(T_1)$; in other words it would represent a recursive channel type, which is not allowed. We can represent this failure-criterion with the Fail-relation by introducing the axiom

$$\forall\tau_1 : \forall\tau_2 : ([\![\tau_1 \leftarrowtriangle \tau_2]\!] \wedge [\![\tau_2 \leftarrowtriangle \tau_1]\!]) \Rightarrow \mathsf{Fail}(\tau_1, \tau_2).$$

This way, when computing a solution, we can easily determine whether or not we have recursive channels, and we can also easily determine which channels are the problem; we can simply refer to the environments and find the names bound to τ_1 and τ_2 respectively.

Remark. Whenever we refer to a solution for a constraint or a conjunction of constraints, we assume that the solution is failure-free i.e. that there exists no Fail-relations in the solution.

In addition to the constraint generation rules that must be defined together with the instantiation of the type system, we require the definition of a type assigning function $\triangleright$. The type assigning function $\triangleright$ takes a solution to one's generated constraints and some type variable τ, and then computes the type of τ. We can only give a very general definition of $\triangleright$, as the details of type assignment given a solution to some constraints depends entirely on how the constraint and solution are defined, which in turn depends on the type system it was generated for.

Definition 3.2.1 (Type assigning function $\triangleright$). Let $\triangleright : 2^{\mathbb{C}} \times \mathsf{TVar} \to \mathsf{Type}$ be a function that given a constraint solution, Φ, and a type variable τ, returns a type T corresponding to the type τ have been given by the predicates of Φ.

We will write $\Phi \triangleright \tau$ for the function application on the constraint solution Φ and type variable τ.

We also introduce an auxiliary function $\mathscr{L}$ as seen in Definition 3.2.2. This function computes a solution, if one exists, to an ALFP constraint. An example of such a function could be the Succinct Solver [20].

Definition 3.2.2 (Constraint solution $\mathscr{L}$). Let $\mathscr{L} : \mathbb{C} \rightharpoonup 2^{\mathbb{C}}$ be a function that, given a conjunction of ALFP constraints, ϕ, computes a solution, Φ. The solution Φ is a minimum set of predicates such that if all the predicates in Φ hold then ϕ is satisfied.

We finally introduce the concept of a minimal typing environment. For most type systems a process, term etc. can be typed in an environment containing only the free names

of the process, term or whatever other constructs we might wish to type. There might however be cases where an environment containing only the free names is not enough, and for this reason we introduce the concept of a minimal typing environment. We formally define the concept in Definition 3.2.3.

DEFINITION 3.2.3 (Minimal environment). Let $\mathcal{J}$ be a judgement. We let $\Gamma_{\mathcal{J}}$ denote a minimal environment such that:

(1) $\Gamma_{\mathcal{J}} \vdash \mathcal{J}$
(2) $\forall \Gamma : \Gamma \vdash \mathcal{J} \Rightarrow \Gamma_{\mathcal{J}} \leq \Gamma$

We now utilise the type assigning function $\rhd$, the solution computing function $\mathscr{L}$, and minimal environments to set forth Theorem 3.2.4 and prove it in Appendix B.1.1. Note the five requirements of the constraint generation for terms, conditions, and assertions, and the encoding of the compatibility predicate $\leftrightharpoons$.

Theorem 3.2.4. *Let P be a process. For any instantiation of the type system and constraint generation such that:*

(1) *For any message M, there exists a type environment Γ and type T such that $\Gamma \vdash M : T$, if and only if $\Gamma_M \vdash M \rightsquigarrow \tau; \phi$ such that $\mathscr{L}(\phi)$ is defined, $\mathscr{L}(\phi) \rhd \tau = T$, and $\forall n : \tau_n \in \Gamma_M : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$*
(2) *For any condition σ, there exists a type environment Γ such that $\Gamma \vdash \sigma$, if and only if $\Gamma_\sigma \vdash \sigma \rightsquigarrow \phi$ such that $\mathscr{L}(\phi)$ is defined, and $\forall n : \tau_n \in \Gamma_\sigma : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$*
(3) *For any assertion Ψ, there exists a type environment Γ such that $\Gamma \vdash \Psi$, if and only if $\Gamma_\Psi \vdash \Psi \rightsquigarrow \phi$ such that $\mathscr{L}(\phi)$ is defined, and $\forall n : \tau_n \in \Gamma_\Psi : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$*
(4) *For any type variables τ_1 and τ_2, and any constraint ϕ, if $\mathscr{L}(\phi \wedge [\![\tau_1 \leftrightharpoons \tau_2]\!])$ is defined, then $\mathscr{L}(\phi \wedge [\![\tau_1 \leftrightharpoons \tau_2]\!]) \rhd \tau_1 \leftrightharpoons \mathscr{L}(\phi \wedge [\![\tau_1 \leftrightharpoons \tau_2]\!]) \rhd \tau_2$.*
(5) *For any constraint generated by a process, ϕ, and any type variables, τ_1 and τ_2, if $\mathscr{L}(\phi) \rhd \tau_1 \leftrightharpoons \mathscr{L}(\phi) \rhd \tau_2$ then $\mathscr{L}(\phi \wedge [\![\tau_1 \leftrightharpoons \tau_2]\!])$ exists.*

There exists a type environment Γ such that $\Gamma \vdash P$, if and only if $\Gamma_P \vdash P \rightsquigarrow \phi$ such that $\mathscr{L}(\phi)$ is defined, and $\forall n : \tau_n \in \Gamma_P : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$

Having proven that our type inference works, the following two sections will show some examples of how it might be instantiated for different ψ-calculi and instantiations of the type system.

3.3. Explicit fusion as a ψ-calculus

In this section we present a possible encoding of the π-calculus with explicit fusion—as introduced in [7] under the name π_F-calculus—as a simply typed ψ-calculus. The π_F-calculus is similar to that of the π-calculus in the sense processes can send names to each other, but differs on how to handle these I/O actions. In the classic π-calculus we treat such inputs with a substitution, but in the π_F-calculus we do this with explicit fusion e.g.

$$z!(x).P \mid z?(y).Q \mid R \quad \longrightarrow_{\pi_F} \quad \langle x = y \rangle \mid P \mid Q \mid R.$$

In order to encode the π_F-calculus, we first define the nominal data types and its operations as seen in Table 3.3.1, where we let $\mathcal{N}$ denote the set of names. In addition we define our simple types as seen in Table 3.3.2. We only require a simple base type, unit, and a channel type $\mathsf{Ch}\ T$.

Secondly, for convenience, we present the syntax of the π_F-calculus in Table 3.3.3. We omit the semantics of the π_F-calculus as they are similar to that of π-calculus with the exception of adding fusions instead of substituting names.

We can now present the encoding of the π_F-calculus as a ψ-calculus as presented in Table 3.3.4.

We now only need to define the typing rules for terms, assertions, and conditions and the corresponding constraint generation rules for inference. We define those as seen in Tables 3.3.5 and 3.3.6. The latter of the two tables also include the encoding of $\tau_1 \leftrightharpoons \tau_2$.

$\mathbf{T}$:	$\mathcal{N}$
$\mathbf{C}$:	$\{a = b \mid a, b \in \mathbf{T}\} \cup \{a \leftrightarrows b \mid a, b \in \mathbf{T}\}$
$\mathbf{A}$:	$\{\{a_1 = b_1, \ldots, a_n = b_n\} \mid a_i, b_i \in \mathcal{N}, n \geq 0\}$
$\otimes$:	$\cup$
$\mathbf{1}$:	$\emptyset$
$\vDash$:	$\{(\Psi, a = b) \mid a = b \in \mathsf{EQ}(\Psi)\} \cup \{(\Psi, a \leftrightarrows b) \mid \Psi \vDash a = b\}$

TABLE 3.3.1. Nominal data types and operations

$T ::=$		Type
	unit	unit type
	$\mathsf{Ch}\ T$	channel type

TABLE 3.3.2. Types for the ψ-calculus encoding the π_F-calculus

$P ::=$		Process
	$\mathbf{0}$	inactivity
	$P \mid P$	parallel composition
	$x!x.P$	output
	$x?x.P$	input
	$(\nu x : T)P$	restriction
	$\langle x = x \rangle$	fusion
x		Name

TABLE 3.3.3. The syntax of the π_F-calculus

[Nil]	$[\![\mathbf{0}]\!] = (\!\{a = a\}\!)$	[Par]	$[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$
[Out]	$[\![x!y.P]\!] = \overline{x}\, y.[\![P]\!]$	[In]	$[\![x?y.P]\!] = \underline{x}(\lambda y)y.[\![P]\!]$
[Res]	$[\![(\nu x : T)P]\!] = (\nu x : T)[\![P]\!]$	[Fus]	$[\![\langle x = y \rangle]\!] = (\!\{x = y\}\!)$

TABLE 3.3.4. Encoding of π in ψ

While having to make sure the constraints generated by the rules are enough, we also need to make sure they satisfy the conditions imposed by ALFP. Luckily, in this case we simple generate conjunctions of predicates for all rules, which according to Table 2.2.1 are valid ALFP constraints.

In addition to the constraints generated by the process, we also define some axioms, seen in Table 3.3.8. These are simply used to propagate $\mathsf{OfChanType}$ predicate though type equality constraints and ensure that at least one Fail predicate will be created if and only if no solution exists.

Finally we define the type assigning function $\triangleright$, as seen in Table 3.3.7.

[Var]	$\dfrac{\Gamma(n) = T}{\Gamma \vdash n : T}$
[Ast]	$\dfrac{\Gamma(a_i) = \Gamma(b_i) \qquad 1 \leq i \leq n}{\Gamma \vdash \{a_1 = b_1, \ldots, a_n = b_n\}}$
[Con]	$\dfrac{\Gamma(a) = \Gamma(b)}{\Gamma \vdash a = b}$ and $\dfrac{\Gamma(a) = \Gamma(b)}{\Gamma \vdash a \leftrightarrows b}$

TABLE 3.3.5. Typing rules for terms, assertions, and conditions

[Term]	$\dfrac{\Gamma(n) = \tau}{\Gamma \vdash n : \tau \rightsquigarrow \tau; \mathtt{T}}$
[Ast]	$\dfrac{\Gamma(a_i) = \tau_{ai} \qquad \Gamma(b_i) = \tau_{bi} \qquad 1 \leq i \leq n}{\Gamma \vdash \{a_1 = b_1, \ldots, a_n = b_n\} \rightsquigarrow \bigwedge_{i=1}^{n} \mathsf{EQ}(\tau_{ai}, \tau_{bi})}$
[Con]	$\dfrac{\Gamma(a) = \tau_a \qquad \Gamma(b) = \tau_b}{\Gamma \vdash a = b \rightsquigarrow \mathsf{EQ}(\tau_a, \tau_b)}$ and $\dfrac{\Gamma(a) = \tau_a \qquad \Gamma(b) = \tau_b}{\Gamma \vdash a \leftrightarrows b \rightsquigarrow \mathsf{EQ}(\tau_a, \tau_b)}$
$[\![\tau_1 \leftharpoonup \tau_2]\!]$	$\mathsf{OfChanType}(\tau_1, \tau_2)$

TABLE 3.3.6. Constraint generation rules for terms, assertions, and conditions and the encoding of $T \leftharpoonup U$ in ALFP

$$\frac{\mathsf{OfChanType}(\tau_1, \tau_2) \in \Phi \qquad \Phi \rhd \tau_2 = T}{\Phi \rhd \tau_1 = \mathsf{Ch}\ T}$$

$$\frac{\nexists \mathsf{OfChanType}(\tau, \tau') \in \Phi}{\Phi \rhd \tau = \mathsf{unit}}$$

TABLE 3.3.7. Type assigning function

$\forall a : \mathsf{Eq}(a, a)$ $\forall a : \forall b : \mathsf{Eq}(a, b) \Rightarrow \mathsf{Eq}(b, a)$ $\forall a : \forall b : \forall c : (\mathsf{Eq}(a, b) \wedge \mathsf{Eq}(b, c)) \Rightarrow \mathsf{Eq}(a, c)$
$\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : (\mathsf{OfChantype}(t_1, t_2) \wedge \mathsf{Eq}(t_1, t_3) \wedge \mathsf{OfChanType}(t_3, t_4)) \Rightarrow \mathsf{Eq}(t_2, t_4)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : (\mathsf{OfChantype}(t_1, t_2) \wedge \mathsf{Eq}(t_2, t_4) \wedge \mathsf{OfChanType}(t_3, t_4)) \Rightarrow \mathsf{Eq}(t_1, t_3)$
$\forall t_1 : \forall t_2 : \forall t_3 : (\mathsf{OfChanType}(t_1, t_2) \wedge \mathsf{Eq}(t_1, t_3)) \Rightarrow \mathsf{OfChanType}(t_3, t_2)$ $\forall t_1 : \forall t_2 : \forall t_3 : (\mathsf{OfChanType}(t_1, t_2) \wedge \mathsf{Eq}(t_2, t_3)) \Rightarrow \mathsf{OfChanType}(t_1, t_3)$
$\forall t : (\mathsf{OfChanType}(t, t) \Rightarrow \mathsf{Fail}(t)$

TABLE 3.3.8. Axioms for explicit fusion

3.3.1. Example of simply typed π-calculus with explicit fusion. Let us now consider the following process

$$P = x!a.x?b.\mathbf{0} \mid y?c.y!c.\mathbf{0} \mid \langle x = y \rangle$$

which encoded in our ψ-calculus would be defined as

$$P = \overline{x}\, a.\underline{x}(\lambda b)b.(\!|\{b = b\}|\!) \mid \underline{y}(\lambda c)c.\overline{y}\, c.(\!|\{c = c\}|\!) \mid (\!|\{x = y\}|\!),$$

for which we want to infer the typing environment used to type this process, if there exists such an environment.

In order to do this we construct a minimal typing environment for the process, using the free names of P, giving us:

$$\Gamma = x : \tau_1, y : \tau_2, a : \tau_3.$$

We can now apply our newly defined constraint generation rules to generate the relevant constraints, to which—since we know they conform to ALFP—we can apply our solution function $\mathscr{L}$ to infer the correct instantiation of the variables, if such an instantiation exists.

Using our generation rules we get the following constraints:

$$\begin{array}{ll} \mathsf{OfChanType}(\tau_1, \tau_3) & \mathsf{Eq}(\tau_4, \tau_4) \\ \mathsf{OfChanType}(\tau_1, \tau_4) & \mathsf{Eq}(\tau_1, \tau_2) \\ \mathsf{OfChanType}(\tau_2, \tau_5) & \mathsf{Eq}(\tau_5, \tau_5) \end{array}$$

By applying $\mathscr{L}$ to our constraints and axioms we get

$$\begin{array}{ll} \mathsf{OfChanType}(\tau_1, \tau_3) & \mathsf{Eq}(\tau_1, \tau_1) \\ \mathsf{OfChanType}(\tau_1, \tau_4) & \mathsf{Eq}(\tau_2, \tau_2) \\ \mathsf{OfChanType}(\tau_1, \tau_5) & \mathsf{Eq}(\tau_3, \tau_3) \\ \mathsf{OfChanType}(\tau_2, \tau_3) & \mathsf{Eq}(\tau_4, \tau_4) \\ \mathsf{OfChanType}(\tau_2, \tau_4) & \mathsf{Eq}(\tau_5, \tau_5) \\ \mathsf{OfChanType}(\tau_2, \tau_5) & \mathsf{Eq}(\tau_1, \tau_2) \\ & \mathsf{Eq}(\tau_2, \tau_1) \\ & \mathsf{Eq}(\tau_3, \tau_4) \\ & \mathsf{Eq}(\tau_4, \tau_3) \\ & \mathsf{Eq}(\tau_3, \tau_5) \\ & \mathsf{Eq}(\tau_5, \tau_3) \\ & \mathsf{Eq}(\tau_4, \tau_5) \\ & \mathsf{Eq}(\tau_5, \tau_4) \end{array}$$

We finally apply our type assigning function to each of our type variables in the minimal initial environment.

- $x : \tau_1$**:** We want to compute $\Phi \rhd \tau_1$. We investigate Φ and find $\mathsf{OfChanType}(\tau_1, \tau_3)$, $\mathsf{OfChanType}(\tau_1, \tau_4)$, and $\mathsf{OfChanType}(\tau_1, \tau_5)$. By our axioms we know that $\tau_3 = \tau_4 = \tau_5$—which is also represented as relations in Φ—and can chose any of them and proceed. We chose $\mathsf{OfChanType}(\tau_1, \tau_3)$ and can conclude that $\Psi \rhd \tau_1 = \mathsf{Ch}(\Psi \rhd \tau_3)$. We once again investigate Φ, but find no $\mathsf{OfChanType}$-constraint for τ_3 and conclude $\Phi \rhd \tau_3 = \mathsf{unit}$ implying $\Phi \rhd \tau_1 = \mathsf{Ch\ Unit}$.
- $y : \tau_2$**:** We want to compute $\Phi \rhd \tau_2$. We investigate Φ and find $\mathsf{Eq}(\tau_2, \tau_1)$. We can thus conclude that $\Phi \rhd \tau_2 = \Phi \rhd \tau_1 = \mathsf{Ch\ unit}$. We could also proceed with a very similar deduction as presented above.
- $a : \tau_3$**:** We want to compute $\Phi \rhd \tau_3$. We investigate Φ and find no $\mathsf{OfChanType}$-constraint for τ_3 and conclude $\Phi \rhd \tau_3 = \mathsf{unit}$.

3.4. Correspondence assertions as a ψ-calculus

In this section we define constraint generation for correspondence assertions. We will make use of a combination of the type systems as defined in [9] and [6]. The section is divided into three subsections: In Section 3.4.1 we present the encoding of correspondence assertions in a ψ-calculus and how to solve the generated constraints; in Section 3.4.2 we present the problems encountered while encoding correspondence assertions, and what restrictions we require in order for the encoding to work; and finally in Section 3.4.3 we present an example of type inference in the constructed ψ-calculus.

$\mathbf{T}$:	M as defined in Table 3.4.3
$\mathbf{C}$:	$\{M = N \mid M, N \in \mathbf{T}\}$
	$\{M = \mathsf{enc}(*, k) \mid M \in \mathbf{T}, k \in \mathcal{N}\}$
	$\{M \neq \mathsf{enc}(*, k) \mid M \in \mathbf{T}, k \in \mathcal{N}\}$
	$\{M \text{ as } N \mid M, N \in \mathbf{T}\}$
	$\{a \leftrightsquigarrow b \mid a, b \in \mathcal{N}\}$
$\mathbf{A}$:	$\{\mathsf{begin}\ \ell(M) \mid M \in \mathbf{T}, \ell \notin \mathcal{N}\} \cup \{\mathsf{end}\ \ell(M) \mid M \in \mathbf{T}, \ell \notin \mathcal{N}\}$
	$\{\vec{x} = \mathsf{dec}(M, k) \mid M \in \mathbf{T}, k \in \mathcal{N}, \vec{x} \subseteq \mathcal{N}\}$
$\otimes$:	undefined
$\mathbf{1}$:	$\mathbf{1}$
$\vDash$:	undefined

TABLE 3.4.1. Nominal data types and operations

$T ::=$		Type
	Un	public data
	$\mathsf{Ch}(T)$	channel type
	$\mathsf{Pair}(x : T, T)$	dependent pair
	$\mathsf{Ok}(S)$	ok to assume S

TABLE 3.4.2. Types used for correspondence encoding

3.4.1. Correspondence encoding. As previously, we only need to define the constraint generation for terms, assertions, and conditions, but unlike explicit fusion, the terms we consider in this case are not just names, but messages. The nominal data types for the ψ-calculus we use to encode correspondence assertions are shown in Table 3.4.1; the terms are defined by formation rules as presented in Table 3.4.3; and finally the types as seen in Table 3.4.2. The compatibility predicate is defined as follows: $\mathsf{Ch}(T) \leftharpoonup T$ and $\mathsf{Un} \leftharpoonup \mathsf{Un}$. The opponent type Un is used to denote public data. This data may flow to and from the any opponent process. The purpose of a opponent process can be summarised as:

> Given a process P representing the legitimate participants making up a system, we want to show that no opponent process O can induce P into an unsafe state, where some expectation is unjustified. An opponent is any process within our spi calculus, except it is not allowed to include any expectations itself. (The opponent goal is to confuse the legitimate participants about who is doing what.) [6]

It is worth noticing that the composition of assertions for this ψ-calculus is undefined, but that is intentional: the syntax only allows for one assertion at a time, and disallows combination. It is not possible—with the syntax we have chosen—to combine e.g. $\mathsf{begin}\ \ell(M)$ and $\mathsf{begin}\ \ell(N)$ into for instance $\mathsf{begin}\ \ell(M, N)$ as this will denote something else, namely the assertion of the pair $\ell(M, N)$ and not both $\ell(M)$ and $\ell(N)$ individually.

Regarding the syntactical components we have chosen, we have chosen to use split instead of e.g. $\mathsf{exercise}$ as defined in [9]. We did this only for convenience, and one could just as well have chosen $\mathsf{exercise}$, as each can be encoded using the other.

The two components $\mathsf{begin}\ \ell(M)$ and $\mathsf{end}\ \ell(M)$ represents assertions of the effects $\ell(M)$. We will interchangeably use effects and assertions to denote $\ell(M)$ as they, for the this type system, are synonymous. The $\mathsf{begin}\ \ell(M)$-assertions is used to denote that the process can now safely assume that $\ell(M)$ holds. Correspondingly the $\mathsf{end}\ \ell(M)$-assertions denotes that a process at this point assumes $\ell(M)$ holds. A straight forward notion of safety for a process is thus: if P reduces to something like $\mathsf{end}\ \ell(M) \mid P'$ then $P' = \mathsf{begin}\ \ell(M) \mid P''$.

$P ::=$		Process
	$\overline{M}\langle M\rangle$	output
	$a(x).P$	input
	$\mathbf{0}$	inactivity
	$P \mid P$	parallel composition
	$!P$	replication
	$(\nu a : T)P$	restriction
	$\mathsf{case}\ M\ \mathsf{of} x_M : P\ \mathsf{else}\ Q$	decryption
	$\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q$	choice
	$\mathsf{split}\ M\ \mathsf{as}\ (x, y)\ \mathsf{in}\ P$	split
	$\mathsf{begin}\ \ell(M)$	assert $\ell(M)$
	$\mathsf{end}\ \ell(M)$	expectation of assertion $\ell(M)$
$M ::=$		Message
	x	variable
	$\mathsf{pair}(M, M)$	pair
	$\{M_1\}_{M_2}$	message encrypted with key M_2
	$\mathsf{fst}\ M$	first element of pair
	$\mathsf{snd}\ M$	second element of pair
	ok	ok
a		Name
x		Variable

TABLE 3.4.3. The syntax of the π-calculus for correspondence assertions

This implies that every time the process assumes something holds, we can with certainty conclude that the assertions holds.

The encoding of the syntax presented in Table 3.4.3 as a ψ-calculusis straightforward and presented in Table 3.4.4. Notice that we have to enforce termination of a process after an output, as ψ-calculi in general do not adhere to this principle, and we encode termination as the unit assertion ($\mathbb{1}$). The unit assertion in this setting has no inherent meaning or implications, and thus serves the same purpose as the more classical nil process $\mathbf{0}$. The only non-trivial encoding cases are [If], [Spl] and [Dec], and we will explain them briefly. While [If] and [Spl] might at first glance seem similar, they are inherently quite different. In [If] the check of whether $M = N$ is purely a semantic detail and in terms of type checking irrelevant. M might be of type T and N of type U, or they might be of the same type. In any case we only care whether or not the processes P and Q are well-typed—if they are, the choice will be well-typed—as determining whether or not M and N have the same type is superfluous: even if they have the same type, we cannot determine whether they are equal or not. For instance consider two messages with the type Un. Since all messages can have the Un-type, there is no way the type alone is enough. In [Spl] on the other hand, it is crucial that M and $\mathsf{pair}(x, y)$ share the same type: in split we are clearly stating that M can be written as $\mathsf{pair}(x, y)$ and thus they must share the same type. For this reason we utilise the condition M as N instead of the "empty" condition $M = N$ used in [If]. In addition we make sure that the names x and y are correctly scoped using the $(\nu x : T_x)(\nu y : T_y)$ component. For [Dec] we introduce the assertions $M_1 = \mathsf{enc}(*, M_2)$ and $M_1 \neq \mathsf{enc}(*, M_2)$, which are used to determine whether or not M_1 is encrypted using the key M_2. If that is the case, we introduce the assertion $\vec{x} = \mathsf{dec}(M_1, M_2)$ denoting that the names $\vec{x}$ now represent the decoding of M_1 with key M_2.

We now present the typing rules for first assertions, conditions, and then terms (messages). The typing rules for assertions are found in Table 3.4.5. The only rules of importance are the rules [End] and [Dcr]. For the [End] rules we either require the assertion

[Out]	$[\![\overline{M_1}\langle M_2\rangle]\!] = \overline{M_1}\, M_2.[\![\mathbf{0}]\!]$
[In]	$[\![a(x).P]\!] = a(\lambda x)x.[\![P]\!]$
[Nil]	$[\![\mathbf{0}]\!] = (\!|\mathbf{1}|\!)$
[Par]	$[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$
[Rep]	$[\![!P]\!] = *[\![P]\!]$
[Res]	$[\![(\nu x : T)P]\!] = (\nu x : T)[\![P]\!]$
[Dec]	$[\![\mathsf{case}\ M_1\ \mathsf{of}\ x_{M_2} : P\ \mathsf{else}\ Q]\!] = (\nu x : T)\left(\mathbf{case}\ \begin{array}{l} M_1 = \mathsf{enc}(*, M_2) : ((\!|x = \mathsf{dec}(M_1, M_2)|\!) \mid [\![P]\!]) \\ M_1 \neq \mathsf{enc}(*, M_2) : [\![Q]\!] \end{array}\right)$
[If]	$[\![\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!] = \mathbf{case}\ M = N : [\![P]\!]\ ,\ M \neq N : [\![Q]\!]$
[Spl]	$[\![\mathsf{split}\ M\ \mathsf{as}\ (x, y)\ \mathsf{in}\ P]\!] = (\nu x : T_x)(\nu y : T_y)(\mathbf{case}\ M\ \mathsf{as}\ \mathsf{pair}(x, y) : [\![P]\!])$
[Beg]	$[\![\mathsf{begin}\ \ell(M)]\!] = (\!|\mathsf{begin}\ \ell(M)|\!)$
[End]	$[\![\mathsf{end}\ \ell(M)]\!] = (\!|\mathsf{end}\ \ell(M)|\!)$

TABLE 3.4.4. Encoding of correspondence π in ψ

[One]	$\Gamma \vdash \mathbf{1}$
[Dcr-1]	$\dfrac{\Gamma \vdash x : T \qquad \Gamma \vdash M_2 : \mathsf{Key}(T) \qquad \Gamma \vdash M_1 : \mathsf{Un}}{\Gamma \vdash x = \mathsf{dec}(M_1, M_2)}$
[Dcr-2]	$\dfrac{\Gamma \vdash x : \mathsf{Un} \qquad \Gamma \vdash M_2 : \mathsf{Un} \qquad \Gamma \vdash M_1 : \mathsf{Un}}{\Gamma \vdash x = \mathsf{dec}(M_1, M_2)}$
[Bgn]	$\Gamma \vdash \mathsf{begin}\ \ell(M)$
[End-1]	$\dfrac{\Gamma = \Gamma_1, \ell(M), \Gamma_2 \qquad \mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma_1)}{\Gamma \vdash \mathsf{end}\ \ell(M)}$
[End-2]	$\dfrac{\Gamma = \Gamma_1, x : \mathsf{Ok}(S), \Gamma_2 \qquad \ell(M) \subseteq S}{\Gamma \vdash \mathsf{end}\ \ell(M)}$
	Remaining assertions are assumed to always be well-typed

TABLE 3.4.5. Typing rules for assertions

$\ell(M)$ to exists in the environment Γ, or we require that there exists an $\mathsf{ok} : \mathsf{Ok}(S)$ binding in the environment Γ such that $\ell(M) \subseteq S$; in other words that the process has received the ability to assert $\mathsf{end}\ \ell(M)$.

The typing rules for conditions are equally simple, and can be seen in Table 3.4.6. The only noticeable rule is [Seq]. We use this condition in the encoding of spilt, and since we here explicitly require the two terms to be equal, they must have the same type.

The typing rules for terms can be seen in Table 3.4.7. The rules are identical to ones presented in [9] but with the addition of opponent type Un, as in [6]. This implies we for most terms have two rules: either the term can be typed with its specific or more describing type—e.g. a pair having the type $\mathsf{Pair}(x : T, U(x))$—or it can be typed with the type Un.

[Eq]	$\Gamma \vdash M = N$
[Seq]	$\dfrac{\Gamma \vdash M_1 : T \qquad \Gamma \vdash M_2 : T}{\Gamma \vdash M_1 \text{ as } M_2}$
[Eeq-1]	$\Gamma \vdash M_1 = \mathsf{enc}(*, M_2)$
[Eeq-2]	$\Gamma \vdash M_1 \neq \mathsf{enc}(*, M_2)$

TABLE 3.4.6. Typing rules for conditions

[Nam]	$\dfrac{\Gamma \vdash \diamond \qquad \Gamma = \Gamma_1, a : T, \Gamma_2}{\Gamma \vdash a : T}$
[Enc]	$\dfrac{\Gamma \vdash M : T \qquad \Gamma \vdash N : \mathsf{Key}(T)}{\Gamma \vdash \{M\}_N : \mathsf{Un}}$
[Enc-Un]	$\dfrac{\Gamma \vdash M : \mathsf{Un} \qquad \Gamma \vdash N : \mathsf{Un}}{\Gamma \vdash \{M\}_N : \mathsf{Un}}$
[Pai]	$\dfrac{\Gamma \vdash M : T \qquad \Gamma \vdash M' : T'(M)}{\Gamma \vdash \mathsf{pair}(M, M') : \mathsf{Pair}(x : T, T'(x))}$
[Pai-Un]	$\dfrac{\Gamma \vdash M : \mathsf{Un} \qquad \Gamma \vdash M' : \mathsf{Un}}{\Gamma \vdash \mathsf{pair}(M, M') : \mathsf{Un}}$
[Ok]	$\dfrac{\Gamma \vdash \diamond \qquad \forall \phi \in S : \ \phi \in \Gamma \vee \exists (x : \mathsf{Ok}(R)) \in \Gamma : \ \phi \in R}{\Gamma \vdash \mathsf{ok} : \mathsf{Ok}(S)}$
[Ok-Un]	$\dfrac{\Gamma \vdash \diamond}{\Gamma \vdash \mathsf{ok} : \mathsf{Un}}$
[Fst]	$\dfrac{\Gamma \vdash M : \mathsf{Pair}(x : T, T'(x))}{\Gamma \vdash \mathsf{fst}\ M : T}$
[Fst-Un]	$\dfrac{\Gamma \vdash M : \mathsf{Un}}{\Gamma \vdash \mathsf{fst}\ M : \mathsf{Un}}$
[Snd]	$\dfrac{\Gamma \vdash M : \mathsf{Pair}(x : T, T'(x))}{\Gamma \vdash \mathsf{snd}\ M : T'(\mathsf{fst}\ M)}$
[Snd-Un]	$\dfrac{\Gamma \vdash M : \mathsf{Un}}{\Gamma \vdash \mathsf{snd}\ M : \mathsf{Un}}$

TABLE 3.4.7. Typing rules for terms (messages)

The main property of the type system is, as described in Theorem 3.4.3, that if a process P can be typed when all its free names have the type Un, then P is robustly safe. Safe and robustly safe are defined in Definition 3.4.1 and Definition 3.4.2 respectively.

DEFINITION 3.4.1 (Safe process). A process P is safe if whenever $P \rightarrow^* (\nu \vec{a} : \vec{T})(\mathsf{end}\ \ell(M) \mid P')$ we have $P' \equiv (\mathsf{begin}\ \ell(M) \mid P'')$ for some P''

DEFINITION 3.4.2 (Robustly safe process). A process P is robustly safe if for every opponent O we have that $P \mid O$ is safe

T	$\mathsf{Eq}(a, b)$	
$\mathsf{Eqa}(\mu_1, \mu_2, \mu_3, \mu_4)$	$\mathsf{Eqi}(\mu_1, \mu_2, \mu_3, \mu_4)$	$\mathsf{Eqi}(\mu_1, \mu_2, \mu_3, \mu_4, \mu_5, \mu_6)$
$\mathsf{IsCha}(\tau_1, \tau_2)$	$\mathsf{IsKey}(\tau_1, \tau_2)$	$\mathsf{IsOk}(\tau)$
$\mathsf{IsPair}(\tau_1, \mu, \tau_2, \tau_3)$	$\mathsf{CanOk}(\tau, \xi)$	$\mathsf{IsUn}(\tau)$
$\mathsf{IsCha?}(\tau_1, \tau_2)$	$\mathsf{IsKey?}(\tau_1, \tau_2)$	$\mathsf{IsOk?}(\tau)$
$\mathsf{IsPair?}(\tau_1, \mu, \tau_2, \tau_3)$	$\mathsf{CanOk?}(\tau, \xi)$	
$\mathsf{NamVar}(a)$	$\mathsf{PairVar}(x)$	
$\mathsf{MsgPai}(\mu_1, \mu_2, \mu_3)$	$\mathsf{MsgOk}(\mu)$	$\mathsf{MsgEnc}(\mu_1, \mu_2, \mu_3)$
$\mathsf{MsgFst}(\mu_1, \mu_2)$	$\mathsf{MsgSnd}(\mu_1, \mu_2)$	$\mathsf{FrmlMsg}(\xi, \ell, \mu)$
$\mathsf{IsAbs}(\tau_1, \tau_2, \mu_1, \mu_2, \tau_3)$	$\mathsf{Abs}(\mu_1, \mu_2, \mu_3)$	
$\mathsf{AbsCanOk}(\tau, \mu, \xi)$	$\mathsf{AbsCanOk?}(\tau, \mu, \xi)$	
$\mathsf{AbsMsgPai}(\mu_1, \mu_2, \mu_3, \mu_4)$	$\mathsf{AbsMsgOk}(\mu_1, \mu_2)$	$\mathsf{AbsMsgEnc}(\mu_1, \mu_2, \mu_3, \mu_4)$
$\mathsf{AbsMsgFst}(\mu_1, \mu_2, \mu_3)$	$\mathsf{AbsMsgSnd}(\mu_1, \mu_2, \mu_3)$	$\mathsf{AbsFrmlMsg}(\mu_1, \xi, \ell, \mu_2)$
$\mathsf{Ins}(\mu_1, \mu_2, \mu_3)$		
$\mathsf{InsCanOk}(\tau, \mu_1, \mu_2, \xi)$	$\mathsf{InsCanOk?}(\tau, \mu_1, \mu_2, \xi)$	
$\mathsf{InsMsgPai}(\mu_1, \mu_2, \mu_3, \mu_4, \mu_5)$	$\mathsf{InsMsgOk}(\mu_1, \mu_2, \mu_3)$	$\mathsf{InsMsgEnc}(\mu_1, \mu_2, \mu_3, \mu_4, \mu_5)$
$\mathsf{InsMsgFst}(\mu_1, \mu_2, \mu_3, \mu_4)$	$\mathsf{InsMsgSnd}(\mu_1, \mu_2, \mu_3, \mu_4)$	$\mathsf{InsFrmlMsg}(\mu_1, \mu_2, \xi, \ell, \mu_3)$

TABLE 3.4.8. Constraint language

Theorem 3.4.3. *If* $x_1 : \mathsf{Un}, \ldots, x_k : \mathsf{Un} \vdash P$ *where* $\{x_1, \ldots, x_k\} = \mathit{fn}(P)$ *then* P *is robustly safe.*

3.4.1.1. *Constraint generation for terms, assertions, and conditions.* Before we present the constraint generation rules for the typing rules presented in Tables 3.4.5 to 3.4.7 we introduce the constraint language we will use, and what each of the constraints is supposed to denote. The full constraint language can be seen in Table 3.4.8. As the rules have to be syntax directed, we need to collapse the two rules for each term into a single rule. We do this by following the approach presented in [11] and introduce the concept of maybe-constraints. These constraints denote that something might be true, but not necessarily. For instance, a pair can be typed either with a $\mathsf{Pair}(x : T, U(x))$ type or an Un type. In the constraint generation we thus introduce the constraint $\mathsf{IsPair?}(\tau, \cdots)$ to denote that the type τ might be a Pair-type. If we later on discover that τ also might be channel type, i.e. we get a $\mathsf{IsCha?}(\tau, \cdots)$ constraint, we can conclude that τ must be an Un type, as this is the only type that can correctly type both channels and pairs. If we never get a contradictory constraint for the $\mathsf{IsPair?}(\tau, \cdots)$ constraint we can conclude that τ is indeed a Pair-type.

The correspondence assertions themselves require a bit more work. If we limited the processes to always combine an ok-term with the full term of one of the assertions, e.g. $\mathsf{begin}\ \ell(a) \mid \overline{x}\,\mathsf{pair}(a, \mathsf{ok}).(\!|\mathbf{1}|\!)$, we could handle the correspondence assertions using only a single variable. This, however, is not the case. Consider for instance the process

$$(\mathsf{begin}\ \ell(\mathsf{pair}(a, b)) \mid \overline{x}\,\mathsf{pair}(a, \mathsf{ok}).(\!|\mathbf{1}|\!)) \mid \cdots \mid \underline{x}(\lambda c)c.\mathbf{case}\ c\ \mathsf{as}\ \mathsf{pair}(d, e) : \mathsf{end}\ \ell(\mathsf{pair}(d, b)).$$

It is clear that this process can be well-typed, assuming that the name b is known to all parallel components. The problem thus lies in the dependent pair $\mathsf{pair}(a, \mathsf{ok})$ and the assertion $\mathsf{begin}\ \ell(\mathsf{pair}(a, b))$. We need to make sure the type of the ok-term allows us to substitute a for d in the receiving process. We do this by encoding the message inside the assertion $(\mathsf{pair}(a, b))$, using the MsgX predicates—here X is simply a placeholder—to encode messages, e.g. $\mathsf{MsgFst}(\mu_1, \mu_2)$ predicate means that $\mu_1 = \mathsf{fst}\ \mu_2$, and similarly $\mathsf{FrmlMsg}(\xi, \ell, \mu)$ denotes that $\xi = \ell(\mu)$, and then conducting our substitution on those, by using the Abs predicate to denote that a type is an abstraction of another, and the Ins predicates to denote that a type is an instantiation of another. The encoding of messages

[MN-C]	$\dfrac{x \in \mathcal{N}}{x \twoheadrightarrow x; \mathsf{NamVar}(x); \mathtt{T}}$
[MP-C]	$\dfrac{M \twoheadrightarrow \mu_m; \psi_m; \phi_m \qquad N \twoheadrightarrow \mu_n; \psi_n; \phi_n}{\mathsf{pair}(M,N) \twoheadrightarrow \mu; \mathsf{MsgPai}(\mu,\mu_m,\mu_n); \psi_m \wedge \phi_m \wedge \psi_n \wedge \phi_n}$
[MF-C]	$\dfrac{M \twoheadrightarrow \mu_m; \psi_m; \phi_m}{\mathsf{fst}\ M \twoheadrightarrow \mu; \mathsf{MsgFst}(\mu,\mu_m); \psi_m \wedge \phi_m}$
[MS-C]	$\dfrac{M \twoheadrightarrow \mu_m; \psi_m; \phi_m}{\mathsf{snd}\ M \twoheadrightarrow \mu; \mathsf{MsgSnd}(\mu,\mu_m); \psi_m \wedge \phi_m}$
[MO-C]	$\mathsf{ok} \twoheadrightarrow \mu; \mathsf{MsgOk}(\mu); \mathtt{T}$
[ME-C]	$\dfrac{M \twoheadrightarrow \mu_m; \psi_m; \phi_m \qquad N \twoheadrightarrow \mu_n; \psi_n; \phi_n}{\{M\}_N \twoheadrightarrow \mu; \mathsf{MsgEnc}(\mu,\mu_m,\mu_n); \psi_m \wedge \phi_m \wedge \psi_n \wedge \phi_n}$
[FM-C]	$\dfrac{M \twoheadrightarrow \mu; \psi; \phi}{\ell(M) \twoheadrightarrow \xi; \mathsf{FrmlMsg}(\xi,\ell,\mu); \psi \wedge \phi}$

TABLE 3.4.9. Message and formula structure encoding constraints

is defined in Table 3.4.9, and further discussion of the difficulties of combining Ok- and Pair-types can be found in Section 3.4.2.

We can now finally present the constraint generation rules for terms and assertions as seen in Tables 3.4.10 and 3.4.11.

In order to solve the constraints, we need a set of axioms, that allows us to reason about the predicates constructed during the constraint generation. We thus introduce the set of constraints seen in Tables 3.4.13 to 3.4.15. Many of the axioms are rather simple: for instance if $\mathsf{Eq}(a,b)$ and $\mathsf{IsOk}(a)$ then $\mathsf{IsOk}(b)$ and so forth. Others are slightly more complicated, for instance the constraint

$$\forall t : \forall x : \forall u_1 : \forall u_2 : \forall t_f : \forall t_s : (\mathsf{IsOk?}(t) \wedge \neg\mathsf{IsCha?}(t,u_1) \wedge \neg\mathsf{IsKey?}(t,u_2) \\ \wedge \neg\mathsf{IsPair?}(t,x,t_f,t_s) \wedge \neg\mathsf{IsUn}(t)) \Rightarrow \mathsf{IsOk}(t)$$

allows us to deduce whether or not a $\mathsf{IsOk?}(\tau)$ constraint can be transformed from its maybe-state to certainty-state.

We also need to define the type assigning function $\triangleright$, as seen in Table 3.4.16. In the definition we make use of the functions $\triangleright'$, $\lhd$, $\square$, and $\lozenge$ defined in Tables 3.4.16 to 3.4.18, for constructing abstract types, messages, abstract messages, and instantiated abstract messages, respectively. Note that we use T to denote that any type may be used, and $\mathsf{IsX}(\tau,\vec{\tau})$ to denote $\mathsf{IsOk}(\tau)$, $\mathsf{IsCha}(\tau,\tau_1)$, $\mathsf{IsPair}(\tau,x,\tau_1,\tau_2)$, $\mathsf{IsKey}(\tau,\tau_1)$, or $\mathsf{IsUn}(\tau)$. Also note that we use the same χ as the reconstruction of all x's in pairs. This is because the same pair will use one x when being sent, and another when being split. Obviously these types need to be identical when reconstructed, so we use χ.

We are now almost ready to prove Items 1 to 5 of Theorem 3.2.4 for correspondence assertions, but before we can do so we need Lemmas 3.4.5 to 3.4.8, which we prove in Appendices B.1.2, B.1.3, B.1.6 and B.1.7, and to define the minimal environment.

The minimal environment used for our type inference is defined as seen in Definition 3.4.4, and in order to ensure robust safety one can simply add $\bigwedge_{\tau_n \in \mathsf{ran}(\Gamma)} \mathsf{IsUn}(\tau_n)$ to the conjunction generated by P.

The proofs of Item 4, Item 5, and Items 1 to 3 of Theorem 3.2.4 for correspondence assertions can be found in Appendix B.1.4, Appendix B.1.5, and Appendix B.1.8. It should be noted that we only prove that a type assignment is found if the process is robustly safe,

[Nam-C] $$\frac{\Gamma \vdash \diamond \qquad \Gamma(a) = \tau}{\Gamma \vdash a \rightsquigarrow \tau; \mathsf{NamVar}(a)}$$

[Enc-C] $$\frac{\Gamma \vdash M \rightsquigarrow \tau_m; \phi_m \qquad \Gamma \vdash N \rightsquigarrow \tau_n; \phi_n}{\Gamma \vdash \{M\}_N \rightsquigarrow \tau;\ \ \phi_m \wedge \phi_n \wedge \mathsf{IsUn}(\tau) \wedge \mathsf{IsKey?}(\tau_n, \tau_m)}$$

[Pai-C] $$\frac{\Gamma \vdash M \rightsquigarrow \tau_m; \phi_m \qquad M \twoheadrightarrow \mu; \psi_m; \varphi_m \qquad \Gamma \vdash N \rightsquigarrow \tau_n; \phi_n}{\Gamma \vdash \mathsf{pair}(M,N) \rightsquigarrow \tau;\ \begin{array}{l}\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n \wedge \mathsf{NamVar}(x) \wedge \mathsf{PairVar}(x) \wedge \\ \mathsf{IsPair?}(\tau, x, \tau_m, \tau_n') \wedge \mathsf{IsAbs?}(\tau_n', \tau_n, \mu, x, \tau_m)\end{array}}$$

[Ok-C] $$\frac{\Gamma \vdash \diamond \qquad \Psi_i \twoheadrightarrow \xi_i; \psi_i; \phi_i \qquad \Psi_i \in \Gamma}{\Gamma \vdash \mathsf{ok} \rightsquigarrow \tau;\ \begin{array}{l}\mathsf{IsOk?}(\tau) \wedge \mathsf{IsOkTerm}(\tau) \wedge \bigwedge_i (\mathsf{CanOk?}(\tau, \xi_i) \wedge \psi_i \wedge \phi_i) \wedge \\ \bigwedge_{\tau' \in \mathsf{rng}(\Gamma)} (\forall \xi' : (\mathsf{IsOk}(\tau') \wedge \mathsf{CanOk}(\tau', \xi')) \Rightarrow \mathsf{CanOk?}(\tau, \xi')) \wedge \\ \bigwedge_{\tau' \in \mathsf{rng}(\Gamma)} (\forall \xi' : \forall x : \forall \mu : (\mathsf{IsOk}(\tau') \wedge \mathsf{InsCanOk}(\tau', x, \mu, \xi') \wedge \\ \qquad \mathsf{IsOk}(\tau)) \Rightarrow \mathsf{InsCanOk?}(\tau, x, \mu, \xi'))\end{array}}$$

[Fst-C] $$\frac{\Gamma \vdash M \rightsquigarrow \tau_m; \phi_m}{\Gamma \vdash \mathsf{fst}\ M \rightsquigarrow \tau;\ \begin{array}{l}\phi_m \wedge \mathsf{IsPair?}(\tau_m, x, \tau, \tau_2') \wedge \mathsf{NamVar}(x) \wedge \\ \mathsf{IsAbs?}(\tau_2', \tau_2, \mu, x, \tau) \wedge \mathsf{PairVar}(x)\end{array}}$$

[Snd-C] $$\frac{\Gamma \vdash M \rightsquigarrow \tau_m; \phi_m \qquad \mathsf{fst}\ M \twoheadrightarrow \mu; \psi_1; \phi_1}{\Gamma \vdash \mathsf{snd}\ M \rightsquigarrow \tau;\ \begin{array}{l}\phi_m \wedge \psi_1 \wedge \phi_1 \wedge \mathsf{IsPair?}(\tau_m, x, \tau_1, \tau') \wedge \\ \mathsf{IsAbs?}(\tau', \tau, \mu, x, \tau_1) \wedge \mathsf{NamVar}(x) \wedge \mathsf{PairVar}(x)\end{array}}$$

[Eq-C] $\Gamma \vdash M = N \rightsquigarrow \mathtt{T}$

[Neq-C] $\Gamma \vdash M \neq N \rightsquigarrow \mathtt{T}$

[Seq-C] $$\frac{\Gamma \vdash M \rightsquigarrow \tau_m; \phi_m \qquad \Gamma \vdash N \rightsquigarrow \tau_n; \phi_n}{\Gamma \vdash M\ \mathsf{as}\ N \rightsquigarrow \phi_m \wedge \phi_n \wedge \mathsf{Eq}(\tau_m, \tau_n)}$$

[Eeq-C-1] $\Gamma \vdash M_1 = \mathsf{enc}(*, M_2) \rightsquigarrow \mathtt{T}$

[Eeq-C-2] $\Gamma \vdash M_1 \neq \mathsf{enc}(*, M_2) \rightsquigarrow \mathtt{T}$

TABLE 3.4.10. Constraint generation for correspondence terms and conditions in ψ

as defined in Definition 3.4.2, as a type assignment will not be found if the process can only be typed if there is an Ok-type in the initial environment.

DEFINITION 3.4.4 (Minimal correspondence environment). For any safe process P the minimal environment Γ_P used for type inference is

$$\Gamma_P = \bigcup_{n \in \mathsf{fn}(P)} n : \tau_n$$

where all τ_n are distinct and unique

Lemma 3.4.5 (Message and formula encoding)**.**

(1) *For any message M, if $M \twoheadrightarrow \mu; \psi; \phi$ then $\mathscr{L}(\psi \wedge \phi) \lhd \mu = M$*
(2) *For any assertion $\ell(M)$, if $\ell(M) \twoheadrightarrow \xi; \psi; \phi$ then $\mathscr{L}(\psi \wedge \phi) \lhd \xi = \ell(M)$*

[One-C] $\Gamma \vdash \mathbf{1} \rightsquigarrow \mathtt{T}$

[Dcr-C] $$\frac{\Gamma \vdash x \rightsquigarrow \tau_x; \phi_x \qquad \Gamma \vdash M_1 \rightsquigarrow \tau_{m1}; \phi_{m1} \qquad \Gamma \vdash M_2 \rightsquigarrow \tau_{m2}; \phi_{m2}}{\Gamma \vdash x = \mathsf{dec}(M_1, M_2) \rightsquigarrow \mathsf{IsKey?}(\tau_{m2}, \tau_x) \wedge \mathsf{IsUn}(\tau_{m1}) \wedge \phi_x \wedge \phi_{m1} \wedge \phi_{m2}}$$

[Bgn-C] $\Gamma \vdash \mathsf{begin}\ \ell(M) \rightsquigarrow \mathtt{T}$

[End-C] $$\Gamma \vdash \mathsf{end}\ \ell(M) \rightsquigarrow \begin{cases} [\![\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma)]\!] & \text{if } \ell(M) \in \Gamma \\ [\![\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma)]\!] \wedge \psi \wedge \phi \wedge \\ ((\bigwedge_{x:\tau \in \Gamma} (\forall \xi' : \forall \xi'' : \forall x : \forall \mu : \\ \qquad ((\neg \mathsf{CanOk}(\tau, \xi') \vee \neg \mathsf{Eq}(\xi, \xi')) \wedge \\ \qquad (\neg \mathsf{InsCanOk}(\tau, x, \mu, \xi'') \vee \neg \mathsf{Eqi}(\xi, x, \mu, \xi'')))))) \Rightarrow \\ \qquad \mathsf{FAIL}(\xi)) \end{cases}$$
where $\ell(M) \twoheadrightarrow \xi; \psi; \phi$

TABLE 3.4.11. Constraint generation for correspondence assertions in ψ

$$[\![\tau_1 \leftrightarrows \tau_2]\!] = \mathsf{IsCha?}(\tau_1, \tau_2)$$
TABLE 3.4.12. Encodings

Lemma 3.4.6. *For all τ and ϕ where $\mathsf{IsAbs}(\tau', \tau, \mu, x, \tau'') \in \mathscr{L}(\phi)$, $\mathsf{PairVar}(x) \in \mathscr{L}(\phi)$, and $\mathscr{L}(\phi) \rhd \tau = \mathsf{Ok}(S)$ we have $\mathscr{L}(\phi) \rhd' \tau' = \mathsf{Ok}(R)$ and $\mathscr{L}(\phi) \lhd \mu = M$ such that $R = S[{}^{X}\!/_{M}]$*

Lemma 3.4.7. *For all τ' and ϕ where $\mathsf{IsAbs}(\tau', \tau, \mu, x, \tau'') \in \mathscr{L}(\phi)$, $\mathsf{PairVar}(x) \in \mathscr{L}(\phi)$, and $\mathscr{L}(\phi) \rhd' \tau' = \mathsf{Ok}(S)$ we have $\mathscr{L}(\phi) \rhd \tau = \mathsf{Ok}(R)$ and $\mathscr{L}(\phi) \lhd \mu = M$ such that $S[{}^{M}\!/_{\chi}] = R$*

Lemma 3.4.8.

(1) *The relations* Eq, *Eqa, and Eqi*-6 *are equivalence-relations*
(2) *For all constraints ϕ for which $\Phi = \mathscr{L}(\phi)$, type variables a and b, if $\mathsf{Eq}(a, b) \in \Phi$ then $\Phi \rhd a = \Phi \rhd b$*
(3) *For all $M \twoheadrightarrow \mu_m; \psi_m; \varphi_m$ and $N \twoheadrightarrow \mu_n; \psi_n; \varphi_n$ if $\Phi = \mathscr{L}(\psi_{\{m,n\}} \wedge \varphi_{\{m,n\}})$ then $\mathsf{Eq}(\mu_m, \mu_n) \in \Phi \Rightarrow \Phi \lhd \mu_m = \Phi \lhd \mu_n$*
(4) *For all abstract message M with index χ_m, variable μ_m, and constraint ϕ_m and abstract message N with index χ_n, variable μ_n and constraint ϕ_n if $\Phi = \mathscr{L}(\phi_m \wedge \phi_n)$ then $\mathsf{Eqa}(\chi_m, \mu_m, \chi_n, \mu_n) \in \Phi \Rightarrow \Phi\square(\chi_m, \mu_m) = \Phi\square(\chi_n, \mu_n)$*
(5) *For all instantiated message M with index χ_m, μ'_m, variable μ_m, and constraint ϕ_m and message N with variable μ_n and constraint ϕ_n if $\Phi = \mathscr{L}(\phi_m \wedge \phi_n)$ then $\mathsf{Eqi}(\mu_n, \chi_m, \mu'_m, \mu_m) \in \Phi \Rightarrow \Phi\Diamond(\chi_m, \mu'_m, \mu_m) = \Phi \lhd \mu_n$*
(6) *For all instantiated message M with index χ_m, μ'_m, variable μ_m, and constraint ϕ_m and instantiated message N with index χ_n, μ'_n, variable μ_n, and constraint ϕ_n if $\Phi = \mathscr{L}(\phi_m \wedge \phi_n)$ then $\mathsf{Eqi}(\chi_m, \mu'_m, \mu_m, \chi_n, \mu'_n, \mu_n) \in \Phi \Rightarrow \Phi\Diamond(\chi_m, \mu'_m, \mu_m) = \Phi\Diamond(\chi_n, \mu'_n, \mu_n)$*

$\forall a : \mathsf{Eq}(a, a)$
$\forall a : \forall b : \mathsf{Eq}(a, b) \Rightarrow \mathsf{Eq}(b, a)$
$\forall a : \forall b : \forall c : (\mathsf{Eq}(a, b) \wedge \mathsf{Eq}(b, c)) \Rightarrow \mathsf{Eq}(a, c)$

$\forall t_1 : \forall t_2 : (\mathsf{IsUn}(t_1) \wedge \mathsf{Eq}(t_1, t_2)) \Rightarrow \mathsf{IsUn}(t_2)$
$\forall t_1 : \forall t_2 : (\mathsf{IsOk?}(t_1) \wedge \mathsf{Eq}(t_1, t_2)) \Rightarrow \mathsf{IsOk?}(t_2)$
$\forall t_1 : \forall t_2 : (\mathsf{IsOk}(t_1) \wedge \mathsf{Eq}(t_1, t_2)) \Rightarrow \mathsf{IsOk}(t_2)$

$\forall t_1 : \forall t_2 : \forall u_1 : \forall u_2 : (\mathsf{IsCha?}(t_1, t_2) \wedge \mathsf{Eq}(t_1, u_1) \wedge \mathsf{Eq}(t_2, u_2)) \Rightarrow \mathsf{IsCha?}(u_1, u_2)$
$\forall t_1 : \forall t_2 : \forall u_1 : \forall u_2 : (\mathsf{IsCha}(t_1, t_2) \wedge \mathsf{Eq}(t_1, u_1) \wedge \mathsf{Eq}(t_2, u_2)) \Rightarrow \mathsf{IsCha}(u_1, u_2)$
$\forall t_1 : \forall t_2 : \forall t_3 : (\mathsf{IsCha?}(t_1, t_2) \wedge \mathsf{IsCha?}(t_1, t_3)) \Rightarrow \mathsf{Eq}(t_2, t_3)$
$\forall t_1 : \forall t_2 : \forall t_3 : (\mathsf{IsCha}(t_1, t_2) \wedge \mathsf{IsCha}(t_1, t_3)) \Rightarrow \mathsf{Eq}(t_2, t_3)$

$\forall t_1 : \forall t_2 : \forall u_1 : \forall u_2((\mathsf{IsKey?}(t_1, t_2) \wedge \mathsf{Eq}(t_1, u_1) \wedge \mathsf{Eq}(t_2, u_2)) \Rightarrow \mathsf{IsKey?}(u_1, u_2))$
$\forall t_1 : \forall t_2 : \forall u_1 : \forall u_2((\mathsf{IsKey}(t_1, t_2) \wedge \mathsf{Eq}(t_1, u_1) \wedge \mathsf{Eq}(t_2, u_2)) \Rightarrow \mathsf{IsKey}(u_1, u_2))$
$\forall t_1 : \forall t_2 : \forall u : ((\mathsf{IsKey?}(u, t_1) \wedge \mathsf{IsKey?}(u, t_2)) \Rightarrow \mathsf{Eq}(t_1, t_2))$
$\forall t_1 : \forall t_2 : \forall u : ((\mathsf{IsKey}(u, t_1) \wedge \mathsf{IsKey}(u, t_2)) \Rightarrow \mathsf{Eq}(t_1, t_2))$

$\forall t : \forall x : \forall u : \forall t_f : \forall t_s : (\mathsf{IsPair?}(t, x, t_f, t_s) \wedge \mathsf{Eq}(t, u)) \Rightarrow \mathsf{IsPair?}(u, x, t_f, t_s)$
$\forall t : \forall x : \forall u : \forall t_f : \forall t_s : (\mathsf{IsPair}(t, x, t_f, t_s) \wedge \mathsf{Eq}(t, u)) \Rightarrow \mathsf{IsPair}(u, x, t_f, t_s)$
$\forall t : \forall x : \forall t_f : \forall u_f : \forall t_s : (\mathsf{IsPair?}(t, x, t_f, t_s) \wedge \mathsf{Eq}(t_f, u_f)) \Rightarrow \mathsf{IsPair?}(t, x, u_f, t_s)$
$\forall t : \forall x : \forall t_f : \forall u_f : \forall t_s : (\mathsf{IsPair}(t, x, t_f, t_s) \wedge \mathsf{Eq}(t_f, u_f)) \Rightarrow \mathsf{IsPair}(t, x, u_f, t_s)$
$\forall t : \forall x : \forall u : \forall x_t : \forall x_u : \forall t_f : \forall t_s : \forall u_f : \forall u_s : (\mathsf{IsPair}(t, x_t, t_f, t_s) \wedge \mathsf{IsPair}(u, x_u, u_f, u_s) \wedge \mathsf{Eq}(t, u)) \Rightarrow$
$(\mathsf{Eq}(t_f, u_f) \wedge \mathsf{Eq}(x_t, x_u) \wedge \mathsf{Eq}(t_s, u_s))$

$\forall t : \forall x : \forall t_f : \forall t'_s : \forall t_s : \forall a : \forall t''_s : (\mathsf{IsPair}(t, x, t_f, t'_s) \wedge \mathsf{IsAbs}(t'_s, t_s, a, x, t''_s) \wedge \mathsf{IsOk}(t_s)) \Rightarrow \mathsf{IsOk}(t'_s)$
$\forall t : \forall x : \forall t_f : \forall t'_s : \forall t_s : \forall a : \forall t''_s : (\mathsf{IsPair}(t, x, t_f, t'_s) \wedge \mathsf{IsAbs}(t'_s, t_s, a, x, t''_s) \wedge \mathsf{IsOk}(t'_s)) \Rightarrow \mathsf{IsOk}(t_s)$
$\forall t : \forall x_t : \forall t_f : \forall t_s : \forall u : \forall x_u : \forall u_f : \forall u_s : (\mathsf{IsPair}(t, x_t, t_f, t_s) \wedge \mathsf{IsPair}(u, x_u, u_f, u_s) \wedge \mathsf{Eq}(t, u) \wedge \mathsf{IsOk}(t_s)) \Rightarrow$
$\mathsf{IsOk}(u_s)$
$\forall t : \forall x : \forall t_f : \forall t'_s : \forall t_s : \forall a : \forall t''_s : (\mathsf{IsPair}(t, x, t_f, t'_s) \wedge \mathsf{IsAbs}(t'_s, t_s, a, x, t''_s) \wedge \mathsf{IsUn}(t_s)) \Rightarrow \mathsf{IsUn}(t'_s)$
$\forall t : \forall x : \forall t_f : \forall t'_s : \forall t_s : \forall a : \forall t''_s : (\mathsf{IsPair}(t, x, t_f, t'_s) \wedge \mathsf{IsAbs}(t'_s, t_s, a, x, t''_s) \wedge \mathsf{IsUn}(t'_s)) \Rightarrow \mathsf{IsUn}(t_s)$
$\forall t : \forall x_t : \forall t_f : \forall t_s : \forall u : \forall x_u : \forall u_f : \forall u_s : (\mathsf{IsPair}(t, x_t, t_f, t_s) \wedge \mathsf{IsPair}(u, x_u, u_f, u_s) \wedge \mathsf{Eq}(t, u) \wedge \mathsf{IsUn}(t_s)) \Rightarrow$
$\mathsf{IsUn}(u_s)$

$\forall t : \forall x : \forall t_f : \forall t_s : (\mathsf{IsOk?}(t) \wedge \mathsf{IsPair?}(t, x, t_f, t_s)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(t_f) \wedge \mathsf{IsUn}(t_s))$
$\forall t : \forall u : (\mathsf{IsOk?}(t) \wedge \mathsf{IsCha?}(t, u)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u))$
$\forall t : \forall u : (\mathsf{IsOk?}(t) \wedge \mathsf{IsKey?}(t, u)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u))$
$\forall t : \forall u_1 : \forall u_2 : (\mathsf{IsCha?}(t, u_1) \wedge \mathsf{IsKey?}(t, u_2)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u_1) \wedge \mathsf{IsUn}(u_2))$
$\forall t : \forall x : \forall u : \forall t_f : \forall t_s : (\mathsf{IsCha?}(t, u) \wedge \mathsf{IsPair?}(t, x, t_f, t_s)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u) \wedge \mathsf{IsUn}(t_f) \wedge \mathsf{IsUn}(t_s))$
$\forall t : \forall x : \forall u :: \forall t_f : \forall t_s : (\mathsf{IsKey?}(t, u) \wedge \mathsf{IsPair?}(t, x, t_f, t_s)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u) \wedge \mathsf{IsUn}(t_f) \wedge \mathsf{IsUn}(t_s))$

$\forall t_1 : \forall t_2 : \forall t_3 : \forall x : (\mathsf{IsUn}(t_1) \wedge \mathsf{IsPair?}(t_1, x, t_2, t_3)) \Rightarrow (\mathsf{IsUn}(t_2) \wedge \mathsf{IsUn}(t_3))$
$\forall t_1 : \forall t_2 : (\mathsf{IsUn}(t_1) \wedge \mathsf{IsCha?}(t_1, t_2)) \Rightarrow \mathsf{IsUn}(t_2)$
$\forall t_1 : \forall t_2 : (\mathsf{IsUn}(t_1) \wedge \mathsf{IsKey?}(t_1, t_2)) \Rightarrow \mathsf{IsUn}(t_2)$

$\forall t : \forall x : \forall u_1 : \forall u_2 : \forall t_f : \forall t_s : (\mathsf{IsOk?}(t) \wedge \neg\mathsf{IsCha?}(t, u_1) \wedge \neg\mathsf{IsKey?}(t, u_2) \wedge \neg\mathsf{IsPair?}(t, x, t_f, t_s) \wedge \neg\mathsf{IsUn}(t)) \Rightarrow$
$\mathsf{IsOk}(t)$
$\forall t : \forall x : \forall u_1 : \forall u_2 : \forall t_f : \forall t_s : (\neg\mathsf{IsOk?}(t) \wedge \mathsf{IsCha?}(t, u_1) \wedge \neg\mathsf{IsKey?}(t, u_2) \wedge \neg\mathsf{IsPair?}(t, x, t_f, t_s) \wedge \neg\mathsf{IsUn}(t)) \Rightarrow$
$\mathsf{IsCha}(t, u_1)$
$\forall t : \forall x : \forall u_1 : \forall u_2 : \forall t_f : \forall t_s : (\neg\mathsf{IsOk?}(t) \wedge \neg\mathsf{IsCha?}(t, u_1) \wedge \mathsf{IsKey?}(t, u_2) \wedge \neg\mathsf{IsPair?}(t, x, t_f, t_s) \wedge \neg\mathsf{IsUn}(t)) \Rightarrow$
$\mathsf{IsKey}(t, u_2)$
$\forall t : \forall x : \forall u_1 : \forall u_2 : \forall t_f : \forall t_s : (\neg\mathsf{IsOk?}(t) \wedge \neg\mathsf{IsCha?}(t, u_1) \wedge \neg\mathsf{IsKey?}(t, u_2) \wedge \mathsf{IsPair?}(t, x, t_f, t_s) \wedge$
$\mathsf{IsAbs?}(t_s, t'_s, a, x, t_f) \wedge \neg\mathsf{IsUn}(t)) \Rightarrow (\mathsf{IsPair}(t, x, t_f, t_s) \wedge \mathsf{IsAbs}(t_s, t'_s, a, x, t_f))$

$(\mathsf{AbsCanOk?}(t, x, xi) \wedge \mathsf{IsAbs}(ta', ta, ma, xa, ta'') \wedge \mathsf{CanOk?}(tc, xic) \wedge \mathsf{Eq}(t, ta') \wedge \mathsf{Eq}(x, xa) \wedge \mathsf{Eq}(ta, tc) \wedge$
$xi = xic \wedge (\forall t' : ((\neg\mathsf{Eq}(t, t') \vee \neg\mathsf{IsOkTerm}(t')) \vee (\exists x' : \exists xi' : \mathsf{Eq}(x, x') \wedge$
$\mathsf{Eqa}(x, xi, x', xi') \wedge \mathsf{AbsCanOk?}(t', x', xi')))))) \Rightarrow (\mathsf{AbsCanOk}(t, x, xi) \wedge \mathsf{CanOk}(tc, xic))$
$(\mathsf{AbsCanOk?}(t, x, xi) \wedge \mathsf{IsAbs}(ta', ta, ma, xa, ta'') \wedge \mathsf{InsCanOk?}(tic, xic, aic, xiic) \wedge \mathsf{Eq}(t, ta') \wedge \mathsf{Eq}(x, xa) \wedge$
$\mathsf{Eq}(ta, tic) \wedge xi = xiic \wedge (\forall t' : ((\neg\mathsf{Eq}(t, t') \vee \neg\mathsf{IsOkTerm}(t')) \vee (\exists x' : \exists xi' : \mathsf{Eq}(x, x') \wedge$
$\mathsf{Eqa}(x, xi, x', xi') \wedge \mathsf{AbsCanOk?}(t', x', xi')))))) \Rightarrow (\mathsf{AbsCanOk}(t, x, xi) \wedge \mathsf{InsCanOk}(tic, xic, aic, xiic))$
$(\mathsf{IsCha}(t_c, t) \wedge \mathsf{IsOk}(t) \wedge \mathsf{IsOkTerm}(t) \wedge \mathsf{CanOk?}(t, xi) \wedge$
$(\forall t' : ((\neg\mathsf{Eq}(t, t') \vee \neg\mathsf{IsOkTerm}(t')) \vee (\exists xi' : \mathsf{Eq}(xi, xi') \wedge \mathsf{CanOk?}(t', xi'))))) \Rightarrow \mathsf{CanOk}(t, xi)$
$(\mathsf{IsCha}(t_c, t) \wedge \mathsf{IsOk}(t) \wedge \mathsf{IsOkTerm}(t) \wedge \mathsf{InsCanOk?}(t, x, a, xi) \wedge (\forall t' : ((\neg\mathsf{Eq}(t, t') \vee \neg\mathsf{IsOkTerm}(t')) \vee$
$(\exists x' : \exists a' : \exists xi' : \mathsf{Eqi}(x, a, xi, x', a', xi') \wedge \mathsf{InsCanOk?}(t', x', a', xi'))))) \Rightarrow \mathsf{InsCanOk}(t, x, a, xi)$

$\forall x_1 : \forall x_2 : \forall l : \forall m_1 : \forall m_2 : (\mathsf{FrmlMsg}(x_1, l, m_1) \wedge \mathsf{FrmlMsg}(x_2, l, m_2) \wedge \mathsf{Eq}(m_1, m_2)) \Rightarrow \mathsf{Eq}(x_1, x_2)$
$\forall m : \forall m_1 : \forall m_2 : \forall n : \forall n_1 : \forall n_2 : (\mathsf{MsgPai}(m, m_1, m_2) \wedge \mathsf{MsgPai}(n, n_1, n_2) \wedge \mathsf{Eq}(m_1, n_1) \wedge \mathsf{Eq}(m_2, n_2)) \Rightarrow$
$\mathsf{Eq}(m, n)$
$\forall m : \forall m_1 : \forall m_2 : \forall n : \forall n_1 : \forall n_2 : (\mathsf{MsgEnc}(m, m_1, m_2) \wedge \mathsf{MsgEnc}(n, n_1, n_2) \wedge \mathsf{Eq}(m_1, n_1) \wedge \mathsf{Eq}(m_2, n_2)) \Rightarrow$
$\mathsf{Eq}(m, n)$
$\forall m_1 : \forall m_2 : (\mathsf{MsgOk}(m_1) \wedge \mathsf{MsgOk}(m_2)) \Rightarrow \mathsf{Eq}(m_1, m_2)$
$\forall m_1 : \forall m_2 : \forall m'_1 : \forall m'_2 : (\mathsf{MsgFst}(m_1, m_2) \wedge \mathsf{MsgFst}(m'_1, m'_2) \wedge \mathsf{Eq}(m_2, m'_2)) \Rightarrow \mathsf{Eq}(m_1, m'_1)$
$\forall m_1 : \forall m_2 : \forall m'_1 : \forall m'_2 : (\mathsf{MsgSnd}(m_1, m_2) \wedge \mathsf{MsgSnd}(m'_1, m'_2) \wedge \mathsf{Eq}(m_2, m'_2)) \Rightarrow \mathsf{Eq}(m_1, m'_1)$

TABLE 3.4.13. Axioms for the correspondence generated constraints (I)

$(\mathsf{CanOk?}(t, xi) \wedge \mathsf{IsAbs}(t', t, a, x, t'') \wedge \mathsf{FrmlMsg}(xi, l, m) \wedge \neg\mathsf{Eq}(a, m)) \Rightarrow$
$\quad (\mathsf{AbsCanOk?}(t', x, xi) \wedge \mathsf{AbsFrmlMsg}(x, xi, l, m) \wedge \mathsf{Abs}(a, x, m))$
$(\mathsf{CanOk?}(t, xi) \wedge \mathsf{IsAbs}(t', t, a, x, t'') \wedge \mathsf{FrmlMsg}(xi, l, m) \wedge \mathsf{Eq}(a, m)) \Rightarrow$
$\quad (\mathsf{AbsCanOk?}(t', x, xi) \wedge \mathsf{AbsFrmlMsg}(x, xi, l, x))$
$(\mathsf{InsCanOk?}(t, x, a, xi) \wedge \mathsf{IsAbs}(t', t, a_a, x_a, t'') \wedge \mathsf{InsFrmlMsg}(x, a, xi, l, m) \wedge \neg\mathsf{Eq}(a_a, m)) \Rightarrow$
$\quad (\mathsf{AbsCanOk?}(t', x_a, xi) \wedge \mathsf{AbsFrmlMsg}(x_a, xi, l, m) \wedge \mathsf{Abs}(a_a, x_a, x, a, m))$
$(\mathsf{InsCanOk?}(t, x, a, xi) \wedge \mathsf{IsAbs}(t', t, a_a, x_a, t'') \wedge \mathsf{InsFrmlMsg}(x, a, xi, l, m) \wedge \mathsf{Eq}(a_a, m)) \Rightarrow$
$\quad (\mathsf{AbsCanOk?}(t', x_a, xi) \wedge \mathsf{AbsFrmlMsg}(x_a, xi, l, x_a))$
$(\mathsf{AbsCanOk?}(t', x, xi) \wedge \mathsf{AbsFrmlMsg}(x, xi, l, m) \wedge \mathsf{IsAbs}(t'_a, t, a, x_a, t''_a) \wedge \mathsf{Eq}(x, x_a) \wedge \neg\mathsf{Eq}(x, m)) \Rightarrow$
$\quad (\mathsf{InsCanOk?}(t, x_a, a, xi) \wedge \mathsf{InsFrmlMsg}(x_a, a, xi, l, m) \wedge \mathsf{Ins}(x_a, a, m))$
$(\mathsf{AbsCanOk?}(t', x, xi) \wedge \mathsf{AbsFrmlMsg}(x, xi, l, m) \wedge \mathsf{IsAbs}(t'_a, t, a, x_a, t''_a) \wedge \mathsf{Eq}(x, x_a) \wedge \mathsf{Eq}(x, m)) \Rightarrow$
$\quad (\mathsf{InsCanOk?}(t, x_a, a, xi) \wedge \mathsf{InsFrmlMsg}(x_a, a, xi, l, a) \wedge \mathsf{Ins}(x_a, a, a))$

$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgPai}(m, m_1, m_2) \wedge \mathsf{Eq}(c, m) \wedge \mathsf{Eq}(a, m_1) \wedge \mathsf{Eq}(a, m_2)) \Rightarrow$
$\quad \mathsf{AbsMsgPai}(b, m, b, b)$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgPai}(m, m_1, m_2) \wedge \mathsf{Eq}(c, m) \wedge \mathsf{Eq}(a, m_1) \wedge \neg\mathsf{Eq}(a, m_2)) \Rightarrow$
$\quad (\mathsf{AbsMsgPai}(b, m, b, m_2) \wedge \mathsf{Abs}(a, b, m_2))$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgPai}(m, m_1, m_2) \wedge \mathsf{Eq}(c, m) \wedge \neg\mathsf{Eq}(a, m_1) \wedge \mathsf{Eq}(a, m_2)) \Rightarrow$
$\quad (\mathsf{AbsMsgPai}(b, m, m_1, b) \wedge \mathsf{Abs}(a, b, m_1))$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgPai}(m, m_1, m_2) \wedge \mathsf{Eq}(c, m) \wedge \neg\mathsf{Eq}(a, m_1) \wedge \neg\mathsf{Eq}(a, m_2)) \Rightarrow$
$\quad (\mathsf{AbsMsgPai}(b, m, m_1, m_2) \wedge \mathsf{Abs}(a, b, m_1) \wedge \mathsf{Abs}(a, b, m_2))$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgEnc}(m, m_1, m_2) \wedge \mathsf{Eq}(c, m) \wedge \mathsf{Eq}(a, m_1) \wedge \mathsf{Eq}(a, m_2)) \Rightarrow$
$\quad \mathsf{AbsMsgEnc}(b, m, b, b)$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgEnc}(m, m_1, m_2) \wedge \mathsf{Eq}(c, m) \wedge \mathsf{Eq}(a, m_1) \wedge \neg\mathsf{Eq}(a, m_2)) \Rightarrow$
$\quad (\mathsf{AbsMsgEnc}(b, m, b, m_2) \wedge \mathsf{Abs}(a, b, m_2))$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgEnc}(m, m_1, m_2) \wedge \mathsf{Eq}(c, m) \wedge \neg\mathsf{Eq}(a, m_1) \wedge \mathsf{Eq}(a, m_2)) \Rightarrow$
$\quad (\mathsf{AbsMsgEnc}(b, m, m_1, b) \wedge \mathsf{Abs}(a, b, m_1))$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgEnc}(m, m_1, m_2) \wedge \mathsf{Eq}(c, m) \wedge \neg\mathsf{Eq}(a, m_1) \wedge \neg\mathsf{Eq}(a, m_2)) \Rightarrow$
$\quad (\mathsf{AbsMsgEnc}(b, m, m_1, m_2) \wedge \mathsf{Abs}(a, b, m_1) \wedge \mathsf{Abs}(a, b, m_2))$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgFst}(m, m') \wedge \mathsf{Eq}(c, m) \wedge \mathsf{Eq}(a, m')) \Rightarrow \mathsf{AbsMsgFst}(b, m, b)$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgFst}(m, m') \wedge \mathsf{Eq}(c, m) \wedge \neg\mathsf{Eq}(a, m')) \Rightarrow (\mathsf{AbsMsgFst}(b, m, m') \wedge \mathsf{Abs}(a, b, m'))$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgSnd}(m, m') \wedge \mathsf{Eq}(c, m) \wedge \mathsf{Eq}(a, m')) \Rightarrow \mathsf{AbsMsgSnd}(b, m, b)$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgSnd}(m, m') \wedge \mathsf{Eq}(c, m) \wedge \neg\mathsf{Eq}(a, m')) \Rightarrow (\mathsf{AbsMsgSnd}(b, m, m') \wedge \mathsf{Abs}(a, b, m'))$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgOk}(m) \wedge \mathsf{Eq}(c, m) \wedge \mathsf{Eq}(a, m)) \Rightarrow \mathsf{AbsMsgOk}(b, b)$
$(\mathsf{Abs}(a, b, c) \wedge \mathsf{MsgOk}(m) \wedge \mathsf{Eq}(c, m) \wedge \neg\mathsf{Eq}(a, m)) \Rightarrow \mathsf{AbsMsgOk}(b, m)$

$(\mathsf{FrmlMsg}(xi, l, m) \wedge \mathsf{InsFrmlMsg}(x', m', xi', l, m'') \wedge \mathsf{Eqi}(m, x', m', m'')) \Rightarrow \mathsf{Eqi}(xi, x', m', xi')$
$(\mathsf{NamVar}(n) \wedge \mathsf{NamVar}(n') \wedge \mathsf{Eq}(n, n') \wedge \neg\mathsf{Eq}(n', x)) \Rightarrow \mathsf{Eqi}(n, x, m, n')$
$(\mathsf{MsgPai}(m'_1, m'_2, m'_3) \wedge \mathsf{InsMsgPai}(m_1, m_2, m_3, m_4, m_5) \wedge \mathsf{Eqi}(m'_2, m_1, m_2, m_4) \wedge$
$\quad \mathsf{Eqi}(m'_3, m_1, m_2, m_5)) \Rightarrow \mathsf{Eqi}(m'_1, m_1, m_2, m_3)$
$(\mathsf{MsgEnc}(m'_1, m'_2, m'_3) \wedge \mathsf{InsMsgEnc}(m_1, m_2, m_3, m_4, m_5) \wedge \mathsf{Eqi}(m'_2, m_1, m_2, m_4) \wedge$
$\quad \mathsf{Eqi}(m'_3, m_1, m_2, m_5)) \Rightarrow \mathsf{Eqi}(m'_1, m_1, m_2, m_3)$
$(\mathsf{MsgFst}(m'_1, m'_2) \wedge \mathsf{InsMsgFst}(m_1, m_2, m_3, m_4) \wedge \mathsf{Eqi}(m'_2, m_1, m_2, m_4)) \Rightarrow \mathsf{Eqi}(m'_1, m_1, m_2, m_3)$
$(\mathsf{MsgSnd}(m'_1, m'_2) \wedge \mathsf{InsMsgSnd}(m_1, m_2, m_3, m_4) \wedge \mathsf{Eqi}(m'_2, m_1, m_2, m_4)) \Rightarrow \mathsf{Eqi}(m'_1, m_1, m_2, m_3)$
$(\mathsf{MsgOk}(m'_1) \wedge \mathsf{InsMsgOk}(m_1, m_2, m_3) \wedge \mathsf{Eqi}(m'_1, m_1, m_2, m_3)) \Rightarrow \mathsf{Eqi}(m'_1, m_1, m_2, m_3)$

$(\mathsf{Eqi}(x, a, m, x', a', m') \wedge \mathsf{InsFrmlMsg}(x_1, a_1, xi_1, l, m) \wedge \mathsf{InsFrmlMsg}(x_2, a_2, xi_2, l, m') \wedge \mathsf{Eq}(x, x_1) \wedge \mathsf{Eq}(a, a_1) \wedge$
$\quad \mathsf{Eq}(x', x_2) \wedge \mathsf{Eq}(a', a_2)) \Rightarrow \mathsf{Eqi}(x, a, xi_1, x', a', xi_2)$
$(\mathsf{NamVar}(m) \wedge \mathsf{NamVar}(m') \wedge \mathsf{Eq}(m, m') \wedge \mathsf{NamVar}(x) \wedge \neg\mathsf{Eq}(m, x) \wedge \mathsf{NamVar}(x') \wedge \neg\mathsf{Eq}(m, x') \wedge$
$\quad \mathsf{NamVar}(a) \wedge \mathsf{NamVar}(a')) \Rightarrow \mathsf{Eqi}(x, a, m, x', a', m')$
$(\mathsf{Eqi}(x_1, a_1, m_1, x'_1, a'_1, m'_1) \wedge \mathsf{Eqi}(x_2, a_2, m_2, x'_2, a'_2, m'_2) \wedge \mathsf{Eq}(x_1, x_2) \wedge \mathsf{Eq}(x'_1, x'_2) \wedge \mathsf{Eq}(a_1, a_2) \wedge$
$\quad \mathsf{Eq}(a'_1, a'_2) \wedge \mathsf{InsMsgPai}(x, a, m, m_1, m_2) \wedge \mathsf{Eq}(x, x_1) \wedge \mathsf{Eq}(a, a_1) \wedge \mathsf{InsMsgPai}(x', a', m', m_1, m_2) \wedge$
$\quad \mathsf{Eq}(x', x'_1) \wedge \mathsf{Eq}(a', a'_1)) \Rightarrow \mathsf{Eqi}(x, a, m, x', a', m')$
$(\mathsf{Eqi}(x_1, a_1, m_1, x'_1, a'_1, m'_1) \wedge \mathsf{Eqi}(x_2, a_2, m_2, x'_2, a'_2, m'_2) \wedge \mathsf{Eq}(x_1, x_2) \wedge \mathsf{Eq}(x'_1, x'_2) \wedge \mathsf{Eq}(a_1, a_2) \wedge$
$\quad \mathsf{Eq}(a'_1, a'_2) \wedge \mathsf{InsMsgEnc}(x, a, m, m_1, m_2) \wedge \mathsf{Eq}(x, x_1) \wedge \mathsf{Eq}(a, a_1) \wedge \mathsf{InsMsgEnc}(x', a', m', m_1, m_2) \wedge$
$\quad \mathsf{Eq}(x', x'_1) \wedge \mathsf{Eq}(a', a'_1)) \Rightarrow \mathsf{Eqi}(x, a, m, x', a', m')$
$(\mathsf{Eqi}(x_1, a_1, m_1, x'_1, a'_1, m'_1) \wedge \mathsf{InsMsgFst}(x, a, m, m_1) \wedge \mathsf{Eq}(x, x_1) \wedge \mathsf{Eq}(a, a_1) \wedge \mathsf{InsMsgFst}(x', a', m', m'_1) \wedge$
$\quad \mathsf{Eq}(x', x'_1) \wedge \mathsf{Eq}(a', a'_1)) \Rightarrow \mathsf{Eqi}(x, a, m, x', a', m')$
$(\mathsf{Eqi}(x_1, a_1, m_1, x'_1, a'_1, m'_1) \wedge \mathsf{InsMsgSnd}(x, a, m, m_1) \wedge \mathsf{Eq}(x, x_1) \wedge \mathsf{Eq}(a, a_1) \wedge \mathsf{InsMsgSnd}(x', a', m', m'_1) \wedge$
$\quad \mathsf{Eq}(x', x'_1) \wedge \mathsf{Eq}(a', a'_1)) \Rightarrow \mathsf{Eqi}(x, a, m, x', a', m')$

$(\mathsf{NamVar}(m) \wedge \mathsf{NamVar}(m') \wedge \mathsf{Eq}(m, m') \wedge \mathsf{NamVar}(x) \wedge \mathsf{NamVar}(x') \wedge \mathsf{Eq}(x, x')) \Rightarrow \mathsf{Eqa}(x, m, x', m')$
$(\mathsf{Eqa}(x_1, m_1, x'_1, m'_1) \wedge \mathsf{Eqa}(x_2, m_2, x'_2, m'_2) \wedge \mathsf{Eq}(x_1, x_2) \wedge \mathsf{Eq}(x'_1, x'_2) \wedge \mathsf{AbsMsgPai}(x, m, m_1, m_2) \wedge$
$\quad \mathsf{Eq}(x, x_1) \wedge \mathsf{AbsMsgPai}(x', m', m'_1, m'_2) \wedge \mathsf{Eq}(x', x'_1)) \Rightarrow \mathsf{Eqa}(x, m, x', m')$
$(\mathsf{Eqa}(x_1, m_1, x'_1, m'_1) \wedge \mathsf{Eqa}(x_2, m_2, x'_2, m'_2) \wedge \mathsf{Eq}(x_1, x_2) \wedge \mathsf{Eq}(x'_1, x'_2) \wedge \mathsf{AbsMsgEnc}(x, m, m_1, m_2) \wedge$
$\quad \mathsf{Eq}(x, x_1) \wedge \mathsf{AbsMsgEnc}(x', m', m'_1, m'_2) \wedge \mathsf{Eq}(x', x'_1)) \Rightarrow \mathsf{Eqa}(x, m, x', m')$
$(\mathsf{Eqa}(x_1, m_1, x'_1, m'_1) \wedge \mathsf{AbsMsgFst}(x, m, m_1) \wedge \mathsf{Eq}(x, x_1) \wedge \mathsf{AbsMsgFst}(x', m', m'_1) \wedge \mathsf{Eq}(x', x'_1)) \Rightarrow$
$\quad \mathsf{Eqa}(x, m, x', m')$
$(\mathsf{Eqa}(x_1, m_1, x'_1, m'_1) \wedge \mathsf{AbsMsgSnd}(x, m, m_1) \wedge \mathsf{Eq}(x, x_1) \wedge \mathsf{AbsMsgSnd}(x', m', m'_1) \wedge \mathsf{Eq}(x', x'_1)) \Rightarrow$
$\quad \mathsf{Eqa}(x, m, x', m')$

TABLE 3.4.14. Axioms for the correspondence generated constraints (II)

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgPai}(c,d,m,m_1,m_2) \wedge \mathsf{Eq}(e,m) \wedge \mathsf{Eq}(a,m_1) \wedge \mathsf{Eq}(a,m_2)) \Rightarrow$
$\mathsf{AbsMsgPai}(b,m,b,b)$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgPai}(c,d,m,m_1,m_2) \wedge \mathsf{Eq}(e,m) \wedge \mathsf{Eq}(a,m_1) \wedge \neg\mathsf{Eq}(a,m_2)) \Rightarrow$
$(\mathsf{AbsMsgPai}(b,m,b,m_2) \wedge \mathsf{Abs}(a,b,c,d,m2))$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgPai}(c,d,m,m_1,m_2) \wedge \mathsf{Eq}(e,m) \wedge \neg\mathsf{Eq}(a,m_1) \wedge \mathsf{Eq}(a,m_2)) \Rightarrow$
$(\mathsf{AbsMsgPai}(b,m,m_1,b) \wedge \mathsf{Abs}(a,b,c,d,m_1))$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgPai}(c,d,m,m_1,m_2) \wedge \mathsf{Eq}(e,m) \wedge \neg\mathsf{Eq}(a,m_1) \wedge \neg\mathsf{Eq}(a,m_2)) \Rightarrow$
$(\mathsf{AbsMsgPai}(b,m,m_1,m_2) \wedge \mathsf{Abs}(a,b,c,d,m_1) \wedge \mathsf{Abs}(a,b,c,d,m_2))$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgEnc}(c,d,m,m_1,m_2) \wedge \mathsf{Eq}(e,m) \wedge \mathsf{Eq}(a,m_1) \wedge \mathsf{Eq}(a,m_2)) \Rightarrow$
$\mathsf{AbsMsgEnc}(b,m,b,b)$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgEnc}(c,d,m,m_1,m_2) \wedge \mathsf{Eq}(e,m) \wedge \mathsf{Eq}(a,m_1) \wedge \neg\mathsf{Eq}(a,m_2)) \Rightarrow$
$(\mathsf{AbsMsgEnc}(b,m,b,m_2) \wedge \mathsf{Abs}(a,b,c,d,m2))$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgEnc}(c,d,m,m_1,m_2) \wedge \mathsf{Eq}(e,m) \wedge \neg\mathsf{Eq}(a,m_1) \wedge \mathsf{Eq}(a,m_2)) \Rightarrow$
$(\mathsf{AbsMsgEnc}(b,m,m_1,b) \wedge \mathsf{Abs}(a,b,c,d,m_1))$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgEnc}(c,d,m,m_1,m_2) \wedge \mathsf{Eq}(e,m) \wedge \neg\mathsf{Eq}(a,m_1) \wedge \neg\mathsf{Eq}(a,m_2)) \Rightarrow$
$(\mathsf{AbsMsgEnc}(b,m,m_1,m_2) \wedge \mathsf{Abs}(a,b,c,d,m_1) \wedge \mathsf{Abs}(a,b,c,d,m_2))$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgFst}(f,g,m,m') \wedge \mathsf{Eq}(c,f) \wedge \mathsf{Eq}(d,g) \wedge \mathsf{Eq}(e,m) \wedge \mathsf{Eq}(a,m')) \Rightarrow$
$\mathsf{AbsMsgFst}(b,m,b)$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgFst}(f,g,m,m') \wedge \mathsf{Eq}(c,f) \wedge \mathsf{Eq}(d,g) \wedge \mathsf{Eq}(e,m) \wedge \neg\mathsf{Eq}(a,m')) \Rightarrow$
$(\mathsf{AbsMsgFst}(b,m,m') \wedge \mathsf{Abs}(a,b,c,d,m'))$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgSnd}(f,g,m,m') \wedge \mathsf{Eq}(c,f) \wedge \mathsf{Eq}(d,g) \wedge \mathsf{Eq}(e,m) \wedge \mathsf{Eq}(a,m')) \Rightarrow$
$\mathsf{AbsMsgSnd}(b,m,b)$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgSnd}(f,g,m,m') \wedge \mathsf{Eq}(c,f) \wedge \mathsf{Eq}(d,g) \wedge \mathsf{Eq}(e,m) \wedge \neg\mathsf{Eq}(a,m')) \Rightarrow$
$(\mathsf{AbsMsgSnd}(b,m,m') \wedge \mathsf{Abs}(a,b,c,d,m'))$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgOk}(f,g,m) \wedge \mathsf{Eq}(c,f) \wedge \mathsf{Eq}(d,g) \wedge \mathsf{Eq}(e,m) \wedge \mathsf{Eq}(a,m)) \Rightarrow$
$\mathsf{AbsMsgOk}(b,b)$

$(\mathsf{Abs}(a,b,c,d,e) \wedge \mathsf{InsMsgOk}(f,g,m) \wedge \mathsf{Eq}(c,f) \wedge \mathsf{Eq}(d,g) \wedge \mathsf{Eq}(e,m) \wedge \neg\mathsf{Eq}(a,m)) \Rightarrow$
$\mathsf{AbsMsgOk}(b,m)$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgPai}(b,c,d,e) \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,c) \wedge \mathsf{Eq}(x,d) \wedge \mathsf{Eq}(x,e)) \Rightarrow$
$(\mathsf{InsMsgPai}(x,a,m,a,a) \wedge \mathsf{Ins}(x,a,a))$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgPai}(b,c,d,e) \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,c) \wedge \mathsf{Eq}(x,d) \wedge \neg\mathsf{Eq}(x,e)) \Rightarrow$
$(\mathsf{InsMsgPai}(x,a,m,a,e) \wedge \mathsf{Ins}(x,a,a) \wedge \mathsf{Ins}(x,a,e))$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgPai}(b,c,d,e) \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,c) \wedge \neg\mathsf{Eq}(x,d) \wedge \mathsf{Eq}(x,e)) \Rightarrow$
$(\mathsf{InsMsgPai}(x,a,m,d,a) \wedge \mathsf{Ins}(x,a,d) \wedge \mathsf{Ins}(x,a,a))$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgPai}(b,c,d,e) \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,c) \wedge \mathsf{Eq}(x,d) \wedge \mathsf{Eq}(x,e)) \Rightarrow$
$(\mathsf{InsMsgPai}(x,a,m,d,e) \wedge \mathsf{Ins}(x,a,d) \wedge \mathsf{Ins}(x,a,e))$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgEnc}(b,c,d,e) \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,c) \wedge \mathsf{Eq}(x,d) \wedge \mathsf{Eq}(x,e)) \Rightarrow$
$(\mathsf{InsMsgEnc}(x,a,m,a,a) \wedge \mathsf{Ins}(x,a,a))$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgEnc}(b,c,d,e) \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,c) \wedge \mathsf{Eq}(x,d) \wedge \neg\mathsf{Eq}(x,e)) \Rightarrow$
$(\mathsf{InsMsgEnc}(x,a,m,a,e) \wedge \mathsf{Ins}(x,a,a) \wedge \mathsf{Ins}(x,a,e))$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgEnc}(b,c,d,e) \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,c) \wedge \neg\mathsf{Eq}(x,d) \wedge \mathsf{Eq}(x,e)) \Rightarrow$
$(\mathsf{InsMsgEnc}(x,a,m,d,a) \wedge \mathsf{Ins}(x,a,d) \wedge \mathsf{Ins}(x,a,a))$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgEnc}(b,c,d,e) \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,c) \wedge \mathsf{Eq}(x,d) \wedge \mathsf{Eq}(x,e)) \Rightarrow$
$(\mathsf{InsMsgEnc}(x,a,m,d,e) \wedge \mathsf{Ins}(x,a,d) \wedge \mathsf{Ins}(x,a,e))$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgFst}(b,n,n') \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,n) \wedge \mathsf{Eq}(x,n')) \Rightarrow$
$(\mathsf{InsMsgFst}(x,a,n,a) \wedge \mathsf{Ins}(x,a,a))$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgFst}(b,n,n') \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,n) \wedge \neg\mathsf{Eq}(x,n')) \Rightarrow$
$(\mathsf{InsMsgFst}(x,a,n,n') \wedge \mathsf{Ins}(x,a,n'))$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgSnd}(b,n,n') \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,n) \wedge \mathsf{Eq}(x,n')) \Rightarrow$
$(\mathsf{InsMsgSnd}(x,a,n,a) \wedge \mathsf{Ins}(x,a,a))$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgSnd}(b,n,n') \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,n) \wedge \neg\mathsf{Eq}(x,n')) \Rightarrow$
$(\mathsf{InsMsgSnd}(x,a,n,n') \wedge \mathsf{Ins}(x,a,n'))$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgOk}(b,n) \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,n) \wedge \mathsf{Eq}(x,n)) \Rightarrow \mathsf{InsMsgOk}(x,a,a)$

$(\mathsf{Ins}(x,a,m) \wedge \mathsf{AbsMsgOk}(b,n) \wedge \mathsf{Eq}(x,b) \wedge \mathsf{Eq}(m,n) \wedge \neg\mathsf{Eq}(x,n)) \Rightarrow \mathsf{InsMsgOk}(x,a,n)$

TABLE 3.4.15. Axioms for the correspondence generated constraints (III)

$$\frac{\mathsf{IsPair}(\tau, x, \tau_1, \tau_2) \in \Phi \qquad \Phi \rhd \tau_1 = T \qquad \Phi \rhd' (x, \tau_2) = U}{\Phi \rhd \tau = \mathsf{Pair}(\chi : T, U)}$$

$$\frac{\exists \tau' : \mathsf{IsKey}(\tau, \tau') \in \Phi \qquad \Phi \rhd \tau' = T}{\Phi \rhd \tau = \mathsf{Key}(T)}$$

$$\frac{\exists \tau' : \mathsf{IsCha}(\tau, \tau') \in \Phi \qquad \Phi \rhd \tau' = T}{\Phi \rhd \tau = \mathsf{Ch}(T)}$$

$$\frac{\mathsf{IsOk}(\tau) \in \Phi \qquad S = \{\Phi \lhd \xi \mid \mathsf{CanOk}(\tau, \xi) \in \Phi\} \cup \{\Phi \Diamond (x, \mu, \xi) \mid \mathsf{InsCanOk}(\tau, x, \mu, \xi) \in \Phi\}}{\Phi \rhd \tau = \mathsf{Ok}(S)}$$

$$\frac{\nexists \vec{\tau} : \mathsf{IsX}(\tau, \vec{\tau}) \in \Phi}{\Phi \rhd \tau = T} \qquad \mathtt{T} \rhd \tau = T \qquad \frac{\mathsf{IsUn}(\tau) \in \Phi}{\Phi \rhd \tau = \mathsf{Un}}$$

$$\frac{\nexists \vec{\tau} : \mathsf{IsX}(\tau, \vec{\tau}) \in \Phi}{\Phi \rhd' (x, \tau) = T} \qquad \frac{\mathsf{IsOk}(\tau) \in \Phi \qquad S = \{\Phi \square (x, \xi) \mid \mathsf{AbsCanOk}(\tau, x, \xi) \in \Phi\}}{\Phi \rhd' (x, \tau) = \mathsf{Ok}(S)}$$

$$\frac{\mathsf{IsUn}(\tau) \in \Phi}{\Phi \rhd' (x, \tau) = \mathsf{Un}}$$

TABLE 3.4.16. Reconstuction of types

$$\frac{\mathsf{NamVar}(\mu) \in \Phi \qquad \mathsf{PairVar}(\mu) \notin \Phi}{\Phi \lhd \mu = \mu}$$

$$\frac{\mathsf{PairVar}(\mu) \in \Phi}{\Phi \lhd \mu = \chi}$$

$$\frac{\mathsf{MsgPai}(\mu_1, \mu_2, \mu_3) \in \Phi \qquad \Phi \lhd \mu_2 = M_2 \qquad \Phi \lhd \mu_3 = M_3}{\Phi \lhd \mu_1 = \mathsf{pair}(M_2, M_3)}$$

$$\frac{\mathsf{MsgFst}(\mu_1, \mu_2) \in \Phi \qquad \Phi \lhd \mu_2 = M}{\Phi \lhd \mu_1 = \mathsf{fst}\ M}$$

$$\frac{\mathsf{MsgSnd}(\mu_1, \mu_2) \in \Phi \qquad \Phi \lhd \mu_2 = M}{\Phi \lhd \mu_1 = \mathsf{snd}\ M}$$

$$\frac{\mathsf{MsgOk}(\mu) \in \Phi}{\Phi \lhd \mu = \mathsf{ok}}$$

$$\frac{\mathsf{MsgEnc}(\mu_1, \mu_2, \mu_3) \in \Phi \qquad \Phi \lhd \mu_2 = M_2 \qquad \Phi \lhd \mu_3 = M_3}{\Phi \lhd \mu_1 = \{M_2\}_{M_3}}$$

$$\frac{\mathsf{FrmlMsg}(\xi, \ell, \mu) \in \Phi \qquad \Phi \lhd \mu = M}{\Phi \lhd \xi = \ell(M)}$$

TABLE 3.4.17. Reconstruction of terms

$$\frac{\mathsf{NamVar}(\mu_2) \in \Phi \qquad \mathsf{PairVar}(\mu_2) \notin \Phi}{\Phi\Box(\mu_1, \mu_2) = \mu_2}$$

$$\frac{\mathsf{PairVar}(\mu_2) \in \Phi}{\Phi\Box(\mu_1, \mu_2) = \chi}$$

$$\frac{\mathsf{AbsMsgPai}(\mu_1, \mu_2, \mu_3, \mu_4) \in \Phi \qquad \Phi\Box(\mu_1, \mu_3) = M_3 \qquad \Phi\Box(\mu_1, \mu_4) = M_4}{\Phi\Box(\mu_1, \mu_2) = \mathsf{pair}(M_3, M_4)}$$

$$\frac{\mathsf{AbsMsgFst}(\mu_1, \mu_2, \mu_3) \in \Phi \qquad \Phi\Box(\mu_1, \mu_3) = M}{\Psi\Box(\mu_1, \mu_2) = \mathsf{fst}\ M}$$

$$\frac{\mathsf{AbsMsgSnd}(\mu_1, \mu_2, \mu_3) \in \Phi \qquad \Phi\Box(\mu_1, \mu_3) = M}{\Psi\Box(\mu_1, \mu_2) = \mathsf{snd}\ M}$$

$$\frac{\mathsf{AbsMsgOk}(\mu_1, \mu_2) \in \Phi}{\Phi\Box(\mu_1, \mu_2) = \mu_2}$$

$$\frac{\mathsf{AbsMsgEnc}(\mu_1, \mu_2, \mu_3, \mu_4) \in \Phi \qquad \Phi\Box(\mu_1, \mu_3) = M_3 \qquad \Phi\Box(\mu_1, \mu_4) = M_4}{\Phi\Box(\mu_1, \mu_2) = \{M_3\}_{M_4}}$$

$$\frac{\mathsf{AbsFrmlMsg}(\mu_1, \xi, \ell, \mu_2) \in \Phi \qquad \Phi\Box(\mu_1, \mu_2) = M}{\Phi\Box(\mu_1, \xi) = \ell(M)}$$

$$\frac{\mathsf{NamVar}(\mu_3) \in \Phi \qquad \mathsf{PairVar}(\mu_3) \notin \Phi}{\Phi\Diamond(\mu_1, \mu_2, \mu_3) = \mu_3}$$

$$\frac{\mathsf{PairVar}(\mu_3) \in \Phi}{\Phi\Diamond(\mu_1, \mu_2, \mu_3) = \chi}$$

$$\frac{\mathsf{InsMsgPai}(\mu_1, \mu_2, \mu_3, \mu_4, \mu_5) \in \Phi \qquad \Phi\Diamond(\mu_1, \mu_2, \mu_4) = M_4 \qquad \Phi\Diamond(\mu_1, \mu_2, \mu_5) = M_5}{\Phi\Diamond(\mu_1, \mu_2, \mu_3) = \mathsf{pair}(M_4, M_5)}$$

$$\frac{\mathsf{InsMsgFst}(\mu_1, \mu_2, \mu_3, \mu_4) \in \Phi \qquad \Phi\Diamond(\mu_1, \mu_2, \mu_4) = M}{\Psi\Diamond(\mu_1, \mu_2, \mu_3) = \mathsf{fst}\ M}$$

$$\frac{\mathsf{InsMsgSnd}(\mu_1, \mu_2, \mu_3, \mu_4) \in \Phi \qquad \Phi\Diamond(\mu_1, \mu_2, \mu_4) = M}{\Psi\Diamond(\mu_1, \mu_2, \mu_3) = \mathsf{snd}\ M}$$

$$\frac{\mathsf{InsMsgOk}(\mu_1, \mu_2, \mu_3) \in \Phi}{\Phi\Diamond(\mu_1, \mu_2, \mu_3) = \mu_3}$$

$$\frac{\mathsf{InsMsgEnc}(\mu_1, \mu_2, \mu_3, \mu_4, \mu_5) \in \Phi \qquad \Phi\Diamond(\mu_1, \mu_2, \mu_4) = M_3 \qquad \Phi\Diamond(\mu_1, \mu_2, \mu_5) = M_5}{\Phi\Diamond(\mu_1, \mu_2, \mu_3) = \{M_4\}_{M_5}}$$

$$\frac{\mathsf{InsFrmlMsg}(\mu_1, \mu_2, \xi, \ell, \mu_3) \in \Phi \qquad \Phi\Diamond(\mu_1, \mu_2, \mu_3) = M}{\Phi\Diamond(\mu_1, \mu_2, \xi) = \ell(M)}$$

TABLE 3.4.18. Reconstruction of abstract and instantiated terms

3.4.2. Problems and limitations. We will now show why generation constraints for correspondence types using the ALFP is so difficult, and what we have done to circumvent these difficulties. We will also describe certain kinds of processes, which we are not able to provide type inference for.

3.4.2.1. *Pairs.* We have not managed to create a full type inference for every process, as we have not found a general way of representing dependent types. The main issue with representing dependent types is how to generate an encoding of T based on an encoding of T' or vice versa, when having a constraint $T = T'\{^{M}/_{x}\}$. The difficulty in this is that the ALFP does not allow $\exists$-qualifiers on the right side of an implication, since we otherwise could simply create axioms like

$$\forall t_1 : \forall t_1' : \forall t_2 : \forall x : \forall \mu : ((\mathsf{IsCha}(t_1, t_2) \wedge [\![t_1 = t_1'\{^{\mu}/_{x}\}]\!]) \Rightarrow \exists t_2' : (\mathsf{IsCha}(t_1', t_2') \wedge [\![t_2 = t_2'\{^{\mu}/_{x}\}]\!]))$$

But since this is not allowed, and ALFP requires a fixed set of names, we cannot simply create axioms that recursively create an encoding of the other type.

The problem becomes clearly visible when considering the encoding of messages in Ok-types: assume we have some effect $\mathsf{begin}\ \ell(\mathsf{pair}(a, b))$ and the following process

$$\overline{c}\,\mathsf{pair}(a, \mathsf{pair}(b, \mathsf{ok})).(\!|\mathbf{1}|\!).$$

The message encoding of $\ell(\mathsf{pair}(a, b))$ will be done at constraint-generation-time. However, it should be clear that the type of c must be

$$\mathsf{Ch}(\mathsf{Pair}(\chi_1 : \mathsf{Un}, \mathsf{Pair}(\chi_2 : \mathsf{Un}, \mathsf{Ok}(\ell(\mathsf{pair}(\chi_1, \chi_2)))))).$$

This will require the encoding of the effect $\ell(\mathsf{pair}(\chi_1, \chi_2))$ which we do not have, and since we are not allowed to create a new effect variable ξ and message variable μ for this, we are unable to encode this "double-abstracted" Ok-type. In addition we would have to be able to encode all the partial and full instantiations of this Ok-type as well, as required by the receiving process. Assuming the receiving process looks like

$$\underline{c}(\lambda y).(\nu y_1 : \tau_1, y_2 : \tau_2, y_3 : \tau_3)\mathbf{case}\ y\ \mathsf{as}\ \mathsf{pair}(y_1, \mathsf{pair}(y_2, y_3)) : \overline{c'}\,\mathsf{pair}(y_1, \mathsf{pair}(y_2, \mathsf{ok})).(\!|\mathbf{1}|\!),$$

then obviously the type of τ_3 must be $\mathsf{Ok}(\ell(\mathsf{pair}(y_1, y_2)))$ which again cannot be generated at constraint-generation-time, and the type assigned to the type variable τ_{ok} in the necessary constraint $\mathsf{IsPair}(\tau_{2,\mathsf{ok}}, \chi', \tau_2, \tau_{\mathsf{ok}})$ would have to be $\mathsf{Pair}(\chi' : \mathsf{Un}, \mathsf{Ok}(\ell(\mathsf{pair}(y_1, \chi'))))$ which again would require a non-constraint-generation-time produced encoding, namely the encoding of $\ell(\mathsf{pair}(y_1, \chi'))$.

For this reason we chose to restrict the right-hand-side of pairs to only containing elements which have the types Un or Ok. This means that one cannot have pairs, channels, keys and so forth as the right-hand-side element of pairs, and this limits the levels of abstraction to one. We could also have chosen to consider any other fixed number of abstraction levels, but since none of them allows for an unlimited level of abstraction, one is enough to illustrate the idea. While this may at first seem rather restrictive, we believe it still provides enough expressive power to be useful. For instance [9] showed that the correspondence type system we consider is more powerful than the simpler versions, which only consider latent effects. Following the idea presented in this paper, we could encode latent effects like

$$[\![\overline{M}\ N.\mathbf{1}]\!] = \overline{M}\,\mathsf{pair}(N, \mathsf{ok}).\mathbf{1}$$
$$[\![\underline{M}(\lambda \vec{x})N.P]\!] = \underline{M}(\lambda \vec{x})N.(\nu p : \tau_p, q : \tau_q)(\mathbf{case}\ N\ \mathsf{as}\ \mathsf{pair}(p, q) : P[^{p}/_{N}]).$$

This encoding would satisfy the criteria, and we can thus conclude that despite the limitations introduced, we can still fully express type systems such as the latent effects.

In order to deal with the abstraction for Ok-types we introduce two new sets of constraints for Ok-types and messages, namely the sets of constraints prefixed by Ins and Abs respectively. The Abs-constraints are used to denote an abstract Ok-type and its corresponding abstracted effects, while the Ins-constraints denotes an instantiation of an abstraction. The idea behind these constraints is to construct the new effects—either the abstraction or instantiation—using an index consisting of either what we abstract, or what

we instantiate. Since each pair has its own abstraction variable, we know these indices will be unique and thus enough. Would one want to allow for two levels of abstraction, one would use a "double index" consisting of the possible combinations of abstraction and instantiation.

We can summarise the two new sets of constraints as seen below. We do not include all cases, but the idea should be clear from the examples below. Moreover we assume the variable μ denotes the message M, τ denotes the type T, and ξ the effect E.

Constraint	Meaning
$\mathsf{CanOk?}(\tau, \xi)$	The type T is allowed to include the effect E
$\mathsf{AbsCanOk?}(\tau, x, \xi)$	The type T is allowed to include the effect $E[^{x}/_{M}]$ some message M
$\mathsf{InsCanOk?}(\tau, x, \mu, \xi)$	The type T is allowed to include the effect $E[^{M}/_{x}]$
$\mathsf{MsgPai}(\mu, \mu_1, \mu_2)$	The message $\mathsf{pair}(M_1, M_2)$
$\mathsf{AbsMsgPai}(x, \mu, \mu_1, \mu_2)$	The message $\mathsf{pair}(M_1[^{x}/_{M'}], M_2[^{x}/_{M'}])$ for some message M'
$\mathsf{InsMsgPai}(x, \mu', \mu, \mu_1, \mu_2)$	The message $\mathsf{pair}(M_1[^{M'}/_{x}], M_2[^{M'}/_{x}])$

It is worth noticing that the seemingly arbitrary message we abstract away for x in the Abs constraint is in fact not arbitrary. Whenever we generate a new abstraction—in other words whenever we have a pair containing an ok-term on the right-hand-side—we also construct the constraint $\mathsf{Abs}(M, x, M')$ which denotes "construct the message $M'[^{x}/_{M}]$". Since we only do this once for each pair, and each pairs abstraction-variable is unique it is enough to index the abstraction with x. In the case of Ins this is not good enough: we could have several receivers of the same abstraction and each would generate their own unique instantiation.

3.4.2.2. *Sending oks.* Another problem one has to consider is if a channel is used more than once. If we know that all channels are just used once, for a single communication, then we can freely assign all of the assertions of the sender to the resulting Ok-type inside the Ch-type. However if the channel is used twice we cannot do this simple assignment. Consider for instance the the following processes, where we assume that assertions are scoped by parenthesis:

$$P_1 = (\mathsf{begin}\ \ell(x) \mid \overline{p}\,\mathsf{pair}(x, \mathsf{ok}).(\!|\mathbf{1}|\!)) \mid \cdots \mid (\mathsf{begin}\ \ell(y) \mid \overline{p}\,\mathsf{pair}(y, \mathsf{ok}).(\!|\mathbf{1}|\!)),$$

$$P_2 = (\mathsf{begin}\ \ell(\mathsf{pair}(x_1, x_2)) \mid \overline{p}\,\mathsf{pair}(x_1, \mathsf{ok}).(\!|\mathbf{1}|\!)) \mid \cdots \mid (\mathsf{begin}\ \ell(\mathsf{pair}(y_1, y_2)) \mid \overline{p}\,\mathsf{pair}(y_1, \mathsf{ok}).(\!|\mathbf{1}|\!)),$$

$$P_3 = (\mathsf{begin}\ \ell(x) \mid \overline{p}\,\mathsf{pair}(x, \mathsf{ok}).(\!|\mathbf{1}|\!)) \mid \cdots \mid (\mathsf{begin}\ \ell(\mathsf{pair}(y_1, y_2)) \mid \mathsf{begin}\ \ell(y_1) \mid \overline{p}\,\mathsf{pair}(y_1, \mathsf{ok}).(\!|\mathbf{1}|\!)).$$

Let us consider what channel-type we should assign to p in each of the cases. For P_1 we can easily see that both usages of p is to send a pair with the type $\mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\ell(\chi)))$. This is true as the message in the assertions $\ell(x)$ and $\ell(y)$ is also the element on the left-hand-side of the corresponding pairs, and thus are abstracted away in the Pair-type. Since both usages of p are sending something of the same type, we can thus conclude that $p : \mathsf{Ch}(\mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\ell(\chi))))$. P_2 is similar, but we here reach a disagreement between the two usages of the channel: for the left-hand-side we can conclude the type of the pair is $\mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\ell(\mathsf{pair}(\chi, x_2))))$ and for the right-hand-side we instead have $\mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\ell(\mathsf{pair}(\chi, y_2))))$. It is clear that these two assertions are not the same; while they are identical in structure they differ in the names in the effect namely x_2 and

y_2 respectively. Since the two Ok-types are not identical we are left with no choice but assigning the type $\mathsf{Ch}(\mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\emptyset)))$ to p and hope the assertions are not needed by the receivers. Finally for P_3 we have a combination of the two previous cases. Clearly for the left-hand-side we are sending something of the type $\mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\ell(\chi)))$ and for the right-hand-side we are sending something of the type $\mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\ell(\mathsf{pair}(\chi, y_2)), \ell(\chi)))$. By comparing the Ok-types within the pairs, we can see that they agree on one assertions, namely $\ell(\chi)$. We thus conclude that the type of p should allow the senders to send pairs containing the common or shared assertions, and we have $p : \mathsf{Ch}(\mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\ell(\chi))))$. The right-hand-side is thus only allowed to include a subset of all its possible assertions, and we can hope the remaining were not required by the receiver.

The way we deal with figuring out which effects a channel is allowed to send, is to find all the instances of it being used to output something, and only allow the effect, which are available in *all* those environments.

Unfortunately, as the constraint generation rules [In-C] and [Out-C] do not keep track of whether a channel is sending or receiving, so we need to make this information available in another way. We therefore have decided to make a restriction to what one is allowed to output in a process. We disallow all processes to forward a name which has a Ok-type, and thus only allow ok-terms to have an Ok-type in an output. We illustrate this restriction with some examples as seen below. For simplicity we abuse the notation slightly.

$$\mathsf{begin}\ \ell(x) \mid \overline{x}\,\mathsf{ok}.(\!|1|\!) \mid \underline{x}(\lambda a)a.\overline{y}\,a.(\!|\mathbf{1}|\!) \qquad \text{Disallowed}$$
$$\mathsf{begin}\ \ell(x) \mid \overline{x}\,\mathsf{ok}.(\!|1|\!) \mid \underline{x}(\lambda a)a.\overline{y}\,\mathsf{ok}.(\!|\mathbf{1}|\!) \qquad \text{Allowed}$$

$$\mathsf{begin}\ \ell(x) \mid \overline{x}\,\mathsf{pair}(x, \mathsf{ok}).(\!|1|\!) \mid \underline{x}(\lambda a)a.a\ \mathsf{as}\ (b, c).\overline{y}\,c.(\!|\mathbf{1}|\!) \qquad \text{Disallowed}$$
$$\mathsf{begin}\ \ell(x) \mid \overline{x}\,\mathsf{pair}(x, \mathsf{ok}).(\!|1|\!) \mid \underline{x}(\lambda a)a.a\ \mathsf{as}\ (b, c).\overline{y}\,\mathsf{ok}.(\!|\mathbf{1}|\!) \qquad \text{Allowed}$$

$$\mathsf{begin}\ \ell(x) \mid \overline{x}\,\mathsf{pair}(x, \mathsf{ok}).(\!|1|\!) \mid \underline{x}(\lambda a)a.a\ \mathsf{as}\ (b, c).\overline{y}\,\mathsf{pair}(b, c).(\!|\mathbf{1}|\!) \qquad \text{Disallowed}$$
$$\mathsf{begin}\ \ell(x) \mid \overline{x}\,\mathsf{pair}(x, \mathsf{ok}).(\!|1|\!) \mid \underline{x}(\lambda a)a.a\ \mathsf{as}\ (b, c).\overline{y}\,\mathsf{pair}(b, \mathsf{ok}).(\!|\mathbf{1}|\!) \qquad \text{Allowed}$$

We introduce this restriction as it allow us to know that any output of an Ok-type comes from an ok-term and any input of an Ok-type is bound to a name. We can use this difference when dealing with the above mentioned problem of determining the common effects between several senders.

3.4.3. Example of type inference.

Let us consider the process

$$\begin{aligned} P = {} & (\mathsf{begin}\ \ell(\mathsf{pair}(x, y)) \mid \overline{p}\,\mathsf{pair}(x, \mathsf{ok}).(\!|\mathbf{1}|\!)) \mid \\ & \underline{p}(\lambda c)c.(\nu d : \tau_{11}, e : \tau_{12})(\mathbf{case}\ c\ \mathsf{as}\ \mathsf{pair}(d, e) : \overline{q}\,\mathsf{pair}(y, \mathsf{ok}).(\!|\mathbf{1}|\!) \mid \\ & \underline{q}(\lambda f)f.(\nu g : \tau_{17}, h : \tau_{18})(\mathbf{case}\ f\ \mathsf{as}\ \mathsf{pair}(g, h) : \mathsf{end}\ \ell(\mathsf{pair}(d, g)))) \end{aligned}$$

with the initial minimal environment

$$\Gamma = x : \tau_3, y : \tau_4, p : \tau_5, q : \tau_6.$$

Using the constraint generation presented above we get a conjunction of the following simple[1] constraints

[1] simple as in they are just predicates

$$
\begin{array}{lll}
\mathsf{IsCha?}(\tau_5,\tau_7) & \mathsf{IsPair?}(\tau_7,x_1,\tau_3,\tau_8') & \mathsf{Eq}(\tau_{10},\tau_{13}) \\
\mathsf{IsCha?}(\tau_5,\tau_{10}) & \mathsf{IsPair?}(\tau_{13},x_2,\tau_{11},\tau_{12}') & \mathsf{Eq}(\tau_{16},\tau_{19}) \\
\mathsf{IsCha?}(\tau_6,\tau_{14}) & \mathsf{IsPair?}(\tau_{14},x_3,\tau_4,\tau_{15}') & \mathsf{NamVar}(x_1) \\
\mathsf{IsCha?}(\tau_6,\tau_{16}) & \mathsf{IsPair?}(\tau_{19},x_4,\tau_{17},\tau_{18}') & \mathsf{NamVar}(x_2) \\
\mathsf{IsOk?}(\tau_8) & \mathsf{IsAbs?}(\tau_8',\tau_8,x,x_1,\tau_3) & \mathsf{NamVar}(x_3) \\
\mathsf{IsOk?}(\tau_{15}) & \mathsf{IsAbs?}(\tau_{12}',\tau_{12},d,x_2,\tau_{11}) & \mathsf{NamVar}(x_4) \\
\mathsf{CanOk?}(\tau_8,\xi_1) & \mathsf{IsAbs?}(\tau_{15}',\tau_{15},y,x_3,\tau_4) & \mathsf{NamVar}(x) \\
\mathsf{IsOkTerm}(\tau_8) & \mathsf{IsAbs?}(\tau_{18}',\tau_{18},g,x_4,\tau_{17}) & \mathsf{NamVar}(y) \\
\mathsf{IsOkTerm}(\tau_{15}) & \mathsf{FrmlMsg}(\xi_1,\ell,\mu_1) & \mathsf{NamVar}(g) \\
\mathsf{MsgPai}(\mu_1,x,y) & \mathsf{FrmlMsg}(\xi_2,\ell,\mu_2) & \mathsf{NamVar}(d) \\
\mathsf{MsgPai}(\mu_2,d,g) & &
\end{array}
$$

and the following complex[2] constraints

$$
\begin{array}{l}
\bigwedge_{\tau\in\{\tau_3,\tau_4,\tau_5,\tau_6\}} \\
\quad (\forall\xi' : (\mathsf{IsOk}(\tau)\wedge\mathsf{CanOk}(\tau,\xi'))\Rightarrow\mathsf{CanOk?}(\tau_8,\xi'))\wedge \\
\quad (\forall\xi' : \forall\chi : \forall\mu : (\mathsf{IsOk}(\tau)\wedge\mathsf{InsCanOk}(\tau,\chi,\mu,\xi')\wedge\mathsf{IsOk}(\tau_8)) \\
\qquad \Rightarrow \mathsf{InsCanOk?}(\tau_8,\chi,\mu,\xi'))
\end{array}
$$

$$
\begin{array}{l}
\bigwedge_{\tau\in\{\tau_3,\tau_4,\tau_5,\tau_6,\tau_{10},\tau_{11},\tau_{12}\}} \\
\quad (\forall\xi' : (\mathsf{IsOk}(\tau)\wedge\mathsf{CanOk}(\tau,\xi'))\Rightarrow\mathsf{CanOk?}(\tau_{15},\xi'))\wedge \\
\quad (\forall\xi' : \forall\chi : \forall\mu : (\mathsf{IsOk}(\tau)\wedge\mathsf{InsCanOk}(\tau,\chi,\mu,\xi')\wedge\mathsf{IsOk}(\tau_{15})) \\
\qquad \Rightarrow \mathsf{InsCanOk?}(\tau_{15},\chi,\mu,\xi'))
\end{array}
$$

$$
\begin{array}{l}
(\bigwedge_{\tau\in\{\tau_3,\tau_4,\tau_5,\tau_6,\tau_{10},\tau_{11},\tau_{12},\tau_{16},\tau_{17},\tau_{18}\}} \\
\quad (\forall\xi' : \forall\xi'' : \forall\chi,\forall\mu : \\
\qquad ((\neg\mathsf{CanOk}(\tau,\xi')\vee\mathsf{Eq}(\xi_2,\xi'))\wedge \\
\qquad (\neg\mathsf{InsCanOk}(\tau,\chi,\mu,\xi)\vee\neg\mathsf{Eqi}(\xi_2,\chi,\mu,\xi'')))))) \\
\Rightarrow \mathsf{FAIL}(\xi)
\end{array}
$$

By using the axioms defined above and using the Succinct Solver as our $\mathscr{L}$-function we arrive at the solution presented in Appendix A.1. Since the solution is a very large set of predicates we will not present it in its full length here, and merely refer to the relevant aspects of it as we proceed.

Remark. Since the Succinct Solver disallows $\forall x : P$ in *pre*s the axioms of row-group 10 of Table 3.4.13 and the constraint generated by [End-C] cannot be encoded. We thus, for this example, simplify the axioms to

$$(\mathsf{IsOk}(\tau)\wedge\mathsf{CanOk?}(\tau,\xi))\Rightarrow\mathsf{CanOk}(\tau,\xi)$$

$$(\mathsf{IsOk}(\tau)\wedge\mathsf{AbsCanOk?}(\tau,\chi,\xi))\Rightarrow\mathsf{AbsCanOk}(\tau,\chi,\xi)$$

and manually check the [End-C] constraint. We can rewrite the axioms for this very simple example, as we can easily verify that the channel is only used by one sender and receiver.

We first and foremost conclude that we arrive at a solution which in itself implies there exists some typing of the process, and it is thus well-typed. Secondly, we can conclude that since $\ell(\mathsf{pair}(x,y))\neq\ell(\mathsf{pair}(d,g))$ the typing is not just an assignment of the type Un to all type variables τ—this can also be verified easily be looking at the IsUn-relation in the solution.

Now that we have computed the solution $\mathscr{L}(\phi)=\Phi$ for our constraints, we can utilise Φ together with $\triangleright$ to infer the types of our initial environment. We derive the result manually, but this could easily be automated as well.

[2]complex as in they are not just predicates

$x : \tau_3$ **and** $y : \tau_4$**:** We now try to compute $\Phi \rhd \tau_3$ and $\Phi \rhd \tau_4$. We first investigate Φ for any IsX-constraints for the type variables, but for both τ_3 and τ_4 we have no IsX constraints. We can thus, by definition of $\rhd$ assign any type to τ_3 and τ_4. For simplicity we choose Un as this type is natural choice for an arbitrary type. We thus get $\Phi \rhd \tau_3 = \Phi \rhd \tau_4 = \mathsf{Un}$.

$p : \tau_5$**:** For τ_5 we have three IsX constraints, namely: $\mathsf{IsCha}(\tau_5, \tau_{10})$, $\mathsf{IsCha}(\tau_5, \tau_7)$, and $\mathsf{IsCha}(\tau_5, \tau_{13})$. Luckily, by the axioms, we know that implies $\tau_{10} = \tau_7 = \tau_{13}$ which we can easily verify by the solution; it contains $\mathsf{Eq}(\tau_{13}, \tau_{10})$, $\mathsf{Eq}(\tau_7, \tau_{10})$, and $\mathsf{Eq}(\tau_7, \tau_{13})$. Thus we are free to chose any of τ_7, τ_{10}, and τ_{13} to proceed; we choose τ_7. By $\rhd$ we can thus conclude $\Phi \rhd \tau_5 = \mathsf{Ch}(\Phi \rhd \tau_7)$. Again we are here presented with several versions of the same constraint, and we will not mention them all. We arbitrarily choose one and proceed: $\mathsf{IsPair}(\tau_7, x_1, \tau_{11}, \tau_8')$. We thus know $\Phi \rhd \tau_7 = \mathsf{Pair}(\chi : \Phi \rhd \tau_{11}, \Phi \rhd' \tau_8')$. We have no IsX constraint for τ_{11} so we conclude $\Phi \rhd \tau_{11} = \mathsf{Un}$. For τ_8' we have $\mathsf{IsOk}(\tau_8')$ and $\mathsf{AbsCanOk}(\tau_8', x_1, \xi_1)$, and we conclude $\Phi \rhd' \tau_8' = \mathsf{Ok}(\Phi\square(x_1, \xi_1))$. In order to compute $\Phi\square(x_1, \xi_1)$ we need an $\mathsf{AbsFrmlMsg}$-constraint indexed by x_1, ξ_1, in our case $\mathsf{AbsFrmlMsg}(x_1, \xi_1, \ell, \mu_1)$. We thus proceed to find a message-constraint indexed by x_1, μ_1 and we have $\mathsf{AbsMsgPai}(x_1, \mu_1, x_1, y)$. Since both $\mathsf{NamVar}(x_1)$ and $\mathsf{NamVar}(y)$ our construction of the message is complete. We also have $\mathsf{PairVar}(x_1)$ and we conclude $\Phi\square(x_1, \xi_1) = \ell(\mathsf{pair}(\chi, y))$ implying $\Phi \rhd' \tau_8' = \mathsf{Ok}(\ell(\mathsf{pair}(\chi, y)))$ which in turn gives us $\Phi \rhd \tau_7 = \mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\ell(\mathsf{pair}(\chi, y))))$ and we finally conclude that $\Phi \rhd \tau_5 = \mathsf{Ch}(\mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\ell(\mathsf{pair}(\chi, y))))))$.

$q : \tau_6$**:** This case is very similar to the case for $p : \tau_5$, and follows the same line of construction. For this reason we simply provide the final result immediately: $\Phi \rhd \tau_6 = \mathsf{Ch}(\mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\ell(\mathsf{pair}(d, \chi))))))$.

Our typing environment Γ instantiated using Φ and $\rhd$ is thus

$$\begin{aligned}\Gamma = \; & a : \mathsf{Ch}(\mathsf{Un}) \\ & b : \mathsf{Un} \\ & x : \mathsf{Un} \\ & y : \mathsf{Un} \\ & p : \mathsf{Ch}(\mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\ell(\mathsf{pair}(\chi, y))))) \\ & q : \mathsf{Ch}(\mathsf{Pair}(\chi : \mathsf{Un}, \mathsf{Ok}(\ell(\mathsf{pair}(d, \chi))))))\end{aligned}$$

which is exactly what one would expect.

CHAPTER 4

Linearly typed ψ-calculi

In this chapter we present our second generic type system, linearly typed ψ-calculi, in Section 4.1; describe our type inference in Section 4.2; and present two instantiations of the type system and inference in Sections 4.3 and 4.4.

4.1. A type system for a linearly typed ψ-calculi

We introduce the generic linear type system for the ψ-calculi first presented in [13]. This linear type system shares many of the same aspects as the simple type system introduced in Section 3.1, in particular the requirement of an instantiation and the compatibility relation $\leftharpoonup$ and all rules for type inference for terms, conditions, and assertions.

As previously we let the types of the type system be members of a nominal data type, as they may contain names. Moreover we define equality up to Kleene equality as defined in Definitions 4.1.1 and 4.1.2.

Definition 4.1.1 (Types). We assume a set of types $\mathcal{T}$ and the types to be members of a nominal data type.

Definition 4.1.2 (Type Kleene equality). We write $T_1 \doteq T_2$ of either both T_1 and T_2 are undefined or if $T_1 = T_2$

Unlike for the simple types we defined previously, we in the case of the linear types define our types to be a type structure. We define this structure together with a partial binary operator '+' for the types as seen in Definition 4.1.3. These operators will be utilised in order to decompose linear types into smaller entities.

Definition 4.1.3 (Type structure). A type structure is a set of types $\mathbb{T}$ together with a partial binary operator $+$ on $\mathbb{T}$ such that $T_1+T_2 \doteq T_2+T_1$ and $(T_1+T_2)+T_3 \doteq T_1+(T_2+T_3)$ for all $T_1, T_2, T_3 \in \mathbb{T}$.

As we are now dealing with type structures for which we can add information to the type, we can also order the types. This ordering is rather straightforward and is presented in Definition 4.1.4. In addition to this ordering we introduce the concept of unlimited types. The unlimited types are the types which we can add together with themselves and the result is exactly the same as the initial type. We formalise this in Definition 4.1.5.

Definition 4.1.4 (Ordering of types). We write $T_1 \leq T_2$ if either $T_1 = T_2$ or there exists a T such that $T_1 + T = T_2$.

Definition 4.1.5 (Unlimited types). A type T is unlimited if $T + T = T$ i.e. if T is idempotent under addition

Since we now have defined our types, we can proceed to define the type environments for the linear types. We define the linear type environments in Definition 4.1.6 where we once again let the type environments be partial functions from names to types, but unlike the simple type environments the linear type environments do not contain any assertions. We will handle the assertions separately. It is also worth noticing that we allow types to contain type environments of their own, thus providing a foundation for dependent types. Consider for instance the type $A = \mathsf{Pair}(x : T, U)$ from Section 3.4. By allowing types to have their own environments the type could be written as $A = \mathsf{Pair}(x, U)$ together with the environment $\Gamma_A = x : T$.

Definition 4.1.6 (Type environment). A type environment, Γ, is a partial function from names to types. We write environments as $a : T_1, b : T_2, \ldots$. We allow types to contain type environments and for any type T we let Γ_T denote the type environment associated with T.

As previously, we also introduce the concept of well-formed environments. The definition of well-formed environments—Definition 4.1.7—is very similar to the one presented in Definition 3.1.1. The most notable difference is the requirement that the type environment of the types in the environment are well-formed with regards to the surrounding environment; they cannot assign different types to the same names, and if a name is utilised in a type it must either be typed within the type's type environment or the surrounding environment. Using the example type from before, and assuming we use the environment Γ as the surrounding environment, we would thus require either $\Gamma(x) = \Gamma_A(x) = T$ or $x \notin \mathsf{dom}(\Gamma)$ and $\Gamma_A(x) = T$.

Definition 4.1.7 (Well-formed environment). A type environment Γ is well-formed if whenever $\Gamma = x : T, \Gamma'$ then

(1) $x \notin \mathsf{dom}(\Gamma')$
(2) $\forall y \in \mathsf{dom}(\Gamma_T) \cap \mathsf{dom}(\Gamma) : \Gamma_T(y) = \Gamma(y)$
(3) $\forall z \in \mathsf{names}(T) \setminus \mathsf{dom}(\Gamma_T) : z \in \mathsf{dom}(\Gamma')$

Moreover we also introduce the concept of well-formedness for assertions. Since assertions themselves does not give rise to a meaningful well-formedness property, we define well-formedness for assertions with respect to a type environment as seen in Definition 4.1.8. Ensuring that our assertions are well-formed with respect to our given environment, ensures that we are not asserting anything about names we do not know. From this point on, we generally always assume $\Gamma \heartsuit \Psi$ for any Γ and Ψ unless stating otherwise.

Definition 4.1.8 (Well-formed assertion). An assertion Ψ is well-formed with regards to an type environment Γ, denoted $\Gamma \heartsuit \Psi$, if $\mathsf{names}(\Psi) \subseteq \mathsf{dom}(\Gamma)$

We now proceed to define composition, and thus in some sense also decomposition, of environments and define unlimited environments. In order for two environments to be composable we require that the types of any names they share must be composable i.e. if $\Gamma_1(a) = T_1$ and $\Gamma_2(a) = T_2$ then in order for $\Gamma_1 + \Gamma_2$ to be defined, $T_1 + T_2$ must be defined and this type is the type assigned to a in the composed environment. This is formalised in Definition 4.1.9. An unlimited environment as we define them in Definition 4.1.10, just as for types, is an environment that can be composed with itself and the result is identical to the initial environment.

Definition 4.1.9 (Type environment addition). Let Γ_1 and Γ_2 be type environments. The sum of Γ_1 and Γ_2 is defined as the type environment $\Gamma_1 + \Gamma_2$ such that

$$(\Gamma_1 + \Gamma_2)(x) = \begin{cases} \Gamma_1(x) & x \in \mathsf{dom}(\Gamma_1) \setminus \mathsf{dom}(\Gamma_2) \\ \Gamma_2(x) & x \in \mathsf{dom}(\Gamma_2) \setminus \mathsf{dom}(\Gamma_1) \\ \Gamma_1(x) + \Gamma_2(x) & \textit{otherwise} \end{cases}$$

and $\Gamma_1 + \Gamma_2$ must be well-formed if Γ_1 and Γ_2 are.

Definition 4.1.10 (Unlimited type environments). A type environment Γ is unlimited if $\Gamma = \Gamma + \Gamma$

Again, just as for assertions and types, we can introduce an ordering of the environments. We do this in Definition 4.1.11.

Definition 4.1.11 (Ordering of environments). We write $\Gamma_1 \leq \Gamma_2$ if there exists a Γ such that $\Gamma_1 + \Gamma = \Gamma_2$

We can now introduce the type judgements used by the type system. We have three kinds of judgements, namely one for assertions, terms, and processes respectively. We define the judgements as seen in Definition 4.1.12.

Definition 4.1.12 (Type judgements for assertions, terms, and processes). The type judgement for assertions is of the form

$$\Gamma, \Psi' \vdash \Psi$$

The type judgement for conditions is of the form

$$\Gamma, \Psi \vdash \sigma$$

The type judgement for terms is of the form

$$\Gamma, \Psi \vdash M : T$$

The type judgement for processes is of the form

$$\Gamma, \Psi \vdash P$$

We write $\Gamma, \Psi \vdash \mathcal{J}$ for an arbitrary judgement.

As we can order the environments, and since we have just defined the type of judgements used, we can introduce the notion of a minimal type judgement. A minimal type judgement is a type judgement for which there exists no smaller environment for which the term, process, or assertion can be well-typed. We formalise this in Definition 4.1.13.

Definition 4.1.13 (Minimal type judgement). Let Γ be and environment, Ψ an assertion, M some term, and T some type. We say the type judgement $\Gamma, \Psi \vdash M : T$ is minimal, denoted $\Gamma, \Psi \vdash_{\text{min}} M : T$, if for every $\Gamma' \leq \Gamma$ and $\Psi' \leq \Psi$ we have $\Gamma', \Psi' \nvdash M : T$

Finally we introduce the compatibility relation $\leftharpoonup\!\rho$ in Definition 4.1.14. This relation serves the same purpose as it did in the simple type system introduced in Section 3.1, namely it describes the relationship between the type of the channel and the type of the information send on the channel.

Definition 4.1.14 (Compatibility relation). We introduce the predicates $\leftharpoonup\!\rho$, $\leftharpoonup\!\rho^-$, and $\leftharpoonup\!\rho^+$ to describe which types of values can be carried by channels of a given type. We distinguish between input capability $\leftharpoonup\!\rho^-$ and output capability $\leftharpoonup\!\rho^+$. If $T_1 \leftharpoonup\!\rho^- T_2$ and $T_1 \leftharpoonup\!\rho^+ T_2$ we write $T_1 \leftharpoonup\!\rho\ T_2$.

4.1.1. Criteria for type rules.

Assertion invariance. As assertions are identified up to $\simeq$ this must also be the case for the type system. This implies that if we have a valid judgement $\Gamma, \Psi \vdash \mathcal{J}$ and $\Psi \simeq \Psi'$ then the judgement $\Gamma, \Psi' \vdash \mathcal{J}$ must also be valid.

The empty assertion. In ψ-calculi the empty assertion $\mathbf{1}$ serves as the empty process. We therefore require the following typing rule for the empty assertion

$$[\mathsf{T\text{-}One}] \qquad \Gamma \vdash \mathbf{1} \text{ where } \Gamma \text{ is unlimited.}$$

This rule ensures the empty assertion can only be typed in an environment where no limited resources remain.

Substitutivity. We require that the substitution property for judgements holds for any instantiation of the type system. The substitution property for judgements is defined as seen in Definition 4.1.15.

Definition 4.1.15 (Substitution property for judgements). Suppose $\Gamma + \vec{x} : \vec{T}, \Psi \vdash \mathcal{J}$ with $\vec{x} \cap \mathsf{dom}(\Gamma) = \emptyset$, $\mathsf{fn}(\mathcal{J}) \subseteq \vec{x}$, and $\Gamma_i, \Psi_i \vdash M_i : T_i$ for all $i \in [1, |\vec{x}|]$. Then $\Gamma_0, \Psi_0 \vdash \mathcal{J}[\vec{x} := \vec{M}]$ where

$$\Gamma_0 = \Gamma + \sum_{1 \leq i \leq |\vec{x}|} \Gamma_i$$

and

$$\Psi_0 = \Psi \otimes \bigotimes_{1 \leq i \leq |\vec{x}|} \Psi_i$$

$$[\mathsf{T\text{-}In}]\quad \frac{\Gamma_1 + \vec{x} : \vec{T}, \Psi_1 \vdash P \quad \Gamma_2, \Psi_2 \vdash_{\min} (\lambda\vec{x})N : \vec{T} \to U_o \quad \Gamma_3, \Psi_3 \vdash_{\min} M : U_s}{\Gamma, \Psi \vdash \underline{M}(\lambda\vec{x})N.P} \quad \text{where } \begin{array}{l} U_s \leftrightarrow\!\!\rho^{-} U_o \\ \Gamma = \Gamma_1 + \Gamma_2 + \Gamma_3 \\ \Psi = \Psi_1 \otimes \Psi_2 \otimes \Psi_3 \end{array}$$

$$[\mathsf{T\text{-}Out}]\quad \frac{\Gamma_1, \Psi_1 \vdash_{\min} M : T_s \quad \Gamma_2, \Psi_2 \vdash_{\min} N : T_o \quad \Gamma_3, \Psi_3 \vdash P}{\Gamma, \Psi \vdash \overline{M}\, N.P} \quad \text{where } \begin{array}{l} T_s \leftrightarrow\!\!\rho^{+} T_o \\ \Gamma = \Gamma_1 + \Gamma_2 + \Gamma_3 \\ \Psi = \Psi_1 \otimes \Psi_2 \otimes \Psi_3 \end{array}$$

$$[\mathsf{T\text{-}Par}]\quad \frac{\Gamma_1 + \Gamma_{P_2}, \Psi_1 \otimes \Psi'_{P_2} \vdash P_1 \quad \Gamma_2 + \Gamma_{P_1}, \Psi_2 \otimes \Psi'_{P_1} \vdash P_2}{\Gamma, \Psi \vdash P_1 \mid P_2} \quad \text{where } \begin{array}{l} \Gamma = \Gamma_1 + \Gamma_2 \\ \Psi = \Psi_1 \otimes \Psi_2 \\ \mathcal{F}(P_1) = (\Gamma_{P_1}, \Psi_{P_1}) \\ \mathcal{F}(P_2) = (\Gamma_{P_2}, \Psi_{P_2}) \\ \Psi'_{P_1} \leq \Psi_{P_1} \\ \Psi'_{P_2} \leq \Psi_{P_2} \end{array}$$

$$[\mathsf{T\text{-}Rep}]\quad \frac{\Gamma, \Psi \vdash P}{\Gamma, \Psi \vdash *P} \quad \text{where } \begin{array}{l} \Gamma \text{ is unlimited} \\ \Psi \text{ idempotent} \end{array}$$

$$[\mathsf{T\text{-}Res}]\quad \frac{\Gamma + x : T, \Psi \vdash P}{\Gamma, \Psi \div x \vdash (\nu x : T)P} \qquad [\mathsf{T\text{-}Ass}]\quad \frac{\Gamma, \Psi' \vdash \Psi}{\Gamma, \Psi' \vdash (\!|\Psi|\!)}$$

$$[\mathsf{T\text{-}Cas}]\quad \frac{\Gamma, \Psi \vdash \sigma_i \qquad \Gamma, \Psi \vdash P_i \qquad 1 \leq i \leq k}{\Gamma, \Psi \vdash \mathbf{case}\ \sigma_1 : P_1, \ldots, \sigma_k : P_k}$$

$$[\mathsf{T\text{-}Wea}]\quad \frac{\Gamma, \Psi_1 \vdash P \qquad \mathcal{C}(P, \Psi_1)}{\Gamma, \Psi_1 \otimes \Psi_2 \vdash P} \qquad [\mathsf{T\text{-}Pat}]\quad \frac{\Gamma, \vec{x} : \vec{T}, \Psi \vdash N : U}{\Gamma, \Psi \vdash (\lambda\vec{x})N : (\vec{T} \to U)}$$

TABLE 4.1.1. Typing rules for linearly typed ψ-calculi

4.1.2. Typing rules. We are now in a position to introduce the typing rules for the linear type system for ψ-calculi. The typing rules are defined in Table 4.1.1, and we proceed briefly explain the reasoning behind them. For more information we refer to [13].

In [T-Par] we distribute the bindings according to the rules of context split. The rule is in spirit similar to the one presented in Table 3.1.2, and frame is defined the same way. But we here use $\Psi'_{P_1} \leq \Psi_{P_1}$ and thus allow, but do not require, the parallel components to utilise assertions from the other parallel components. In [T-Res] we make sure to remove any and all assertions involving the restricted name. We do this since the name is not known outside the scope, and consequently the assertions involving this name become partially unknown and thus unusable. Finally [T-Wea] allows us to add assertions to a typing assuming the assertion used to satisfy P satisfies some condition $\mathcal{C}$ with regards to the processes P.

4.2. Constraint generation for linearly typed ψ-calculi

Just as we did in Section 3.2 we need to convert the typing rules into constraint generation rules. The generation rules we present in this section are very similar to the ones found in the aforementioned section. The main difference lies in the fact that we now have type environment and assertions as two different parts of the type check, and

the requirement of minimal type checks. Before we can present the constraint generation rules, we require the definition of a predicate Ħ that takes a type environment Γ in which each name is assigned a type variable, and then returns a new environment Γ' such that $\mathsf{dom}(\Gamma) = \mathsf{dom}(\Gamma')$ and $\mathsf{ran}(\Gamma) \cap \mathsf{ran}(\Gamma') = \emptyset$ i.e. the predicates ensures we have a copy of the input environment where each assigned type variable has been changed to some hitherto unused variable. We formalise this in Definition 4.2.1.

Definition 4.2.1 (Fresh environment Ħ(Γ)). Let Γ be an environment such that $\mathsf{dom}(\Gamma) \subseteq \mathcal{N}$ and $\mathsf{ran}(\Gamma) \subseteq \mathsf{TVar}$. Then we say that Ħ($\Gamma$) = Γ' if $\mathsf{dom}(\Gamma) = \mathsf{dom}(\Gamma')$, $\mathsf{ran}(\Gamma) \cap \mathsf{ran}(\Gamma') = \emptyset$, and $\mathsf{ran}(\Gamma')$ is fresh i.e. hitherto unused in the constraint generation

We use this new predicate Ħ in our constraint generation to simulate the context split. Instead of splitting the environment—as we at constraint generation time cannot know which binds belongs where nor can we know how to split the types—we instead create a new copy of the environment. We can then afterwards compare the types assigned to a name in each of the environments, and then determine if a split was possible and assuming it is, how the types must have been in the complete environment. In order for this approach to work we need to introduce an additional type similar to the unit assertion $\mathbf{1}$; this type is going to represent when a name is not bound in an environment. Before we formally define this type as seen in Definition 4.2.3, we would like to give some intuition to its uses in a small example below.

Because of the rule [T-Wea] and the use of $\Psi' \leq \Psi$ in [T-Par] we cannot know, during the constraint generation, what assertion we will be using in the typing of a process. We therefore encode these assertions using constraints and assertion variables, denoted by ψ. This means we need a new definition of frame, $\mathcal{F}'$, seen in Definition 4.2.2, in which we use the same definitions of $\mathscr{L}$ and $\rhd$ as in Section 3.2.

Definition 4.2.2 (Constraint Frame, $\mathcal{F}'$). Given a process P, $\mathcal{F}'(P) = \langle \Gamma, \psi, \phi \rangle$ if

- $\mathcal{F}(P) = \langle \Gamma, \Psi \rangle$
- $\mathscr{L}(\phi) \rhd \psi = \Psi$

We now present the constraint generation rules, based on the typing rules of Table 4.1.1. They are of the form $\Gamma \vdash M \rightsquigarrow \tau; \psi; \phi$, where τ is a type variable representing the type of the term M, ψ is an assertion variable, representing the assertion used to type M and ϕ is the conjunction of constraints that must be satisfied in order for $\Gamma, \psi \vdash M : \tau$ to hold. The constraint generation rules are similar to the ones presented previously in Table 3.2.1 in Section 3.2. The main difference between the two, are the usages if $\leftharpoonup^-$ and $\leftharpoonup^+$ instead of $\leftharpoonup$, the encoding of not only types, but assertions, and the incorporation of [T-Wea] into each of the other rules. This is a necessity as [T-Wea] is not syntax directed, and we therefore cannot at constraint generation time know where to apply this rule. Let us consider a simplified version of the [T-Par] constraint generation case. Assume we have the processes $\overline{a}\, b.\mathbf{1} \mid \overline{a}\, c.\mathbf{1}$ and type variable environment Γ. As mentioned above we now simulate the context split using Ħ and we have $\Gamma_1 = \text{Ħ}(\Gamma)$ and $\Gamma_2 = \text{Ħ}(\Gamma)$. Assuming we generate constraints using $\Gamma_1 \vdash \overline{a}\, b.\mathbf{1} \rightsquigarrow \phi_1$ and $\Gamma_2 \vdash \overline{a}\, c.\mathbf{1} \rightsquigarrow \phi_2$ it should be quite clear that the name c does not appear in any of the constraints ϕ_1 while the name b does not appear in ϕ_2. For this reason we can assign the empty type ε to c in Γ_1 and b in Γ_2 giving us $\Gamma_1 = a : \tau_a^1, b : \tau_b^1, c : \varepsilon$ and $\Gamma_2 = a : \tau_a^2, b : \varepsilon, c : \tau_c^2$. Since we would like ε to represent the notion "this name should not appear in this environment", we can conclude that $\Gamma_1 + \Gamma_2 = a : (\tau_a^1 + \tau_a^2), b : (\tau_b^1 + \varepsilon), c : (\varepsilon + \tau_c^2) = a : (\tau_a^1 + \tau_a^2), b : \tau_b^1, c : \tau_c^2$.

Definition 4.2.3 (Empty type (ε)). Let ε be the type such that for any other type T we have $T + \varepsilon = T$

In Table 4.2.1, the constraints are represented as encodings, which will have to be converted into ALFP when the constraint generation is instantiated for a specific type system. The meaning of most of the encodings is obvious, with $[\![\mathsf{if}\ \mathcal{C}(P, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]$ being the only exception. This encoding is meant to represent a possible use of [T-Wea].

[C-In]
$$\frac{\begin{array}{c}\Gamma_M \vdash M \rightsquigarrow \tau_m; \psi_m; \phi_m \\ \Gamma_N \vdash (\lambda\vec{x})N \rightsquigarrow (\vec{\tau} \to \tau_n); \psi_n; \phi_n \\ \Gamma_P, \vec{x} : \vec{\tau} \vdash P \rightsquigarrow \psi_p; \phi_p\end{array}}{\begin{array}{l}\Gamma \vdash \underline{M}(\lambda\vec{x})N.P \rightsquigarrow \psi; \phi_m \wedge \phi_n \wedge \phi_p \\ \quad \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftrightarrows^{-} \tau_n]\!] \\ \quad \bigwedge_{\substack{\Gamma(x)=\tau_1 \\ \Gamma_M(x)=\tau_2 \\ \Gamma_N(x)=\tau_3 \\ \Gamma_P(x)=\tau_4}} [\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!] \\ \quad \wedge [\![\mathsf{if}\ \mathcal{C}(\underline{M}(\lambda\vec{x})N.P, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]\end{array}}$$
where $\Gamma_M = \text{Ь}(\Gamma)$, $\Gamma_N = \text{Ь}(\Gamma)$, $\Gamma_P = \text{Ь}(\Gamma)$

[C-Out]
$$\frac{\begin{array}{c}\Gamma_M \vdash M \rightsquigarrow \tau_m; \psi_m; \phi_m \\ \Gamma_N \vdash N \rightsquigarrow \tau_n; \psi_n; \phi_n \\ \Gamma_P \vdash P \rightsquigarrow \psi_p; \phi_p\end{array}}{\begin{array}{l}\Gamma \vdash \overline{M}\, N.P \rightsquigarrow \psi; \phi_m \wedge \phi_n \wedge \phi_p \\ \quad \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftrightarrows^{+} \tau_n]\!] \\ \quad \bigwedge_{\substack{\Gamma(x)=\tau_1 \\ \Gamma_M(x)=\tau_2 \\ \Gamma_N(x)=\tau_3 \\ \Gamma_P(x)=\tau_4}} [\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!] \\ \quad \wedge [\![\mathsf{if}\ \mathcal{C}(\overline{M}\, N.P, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]\end{array}}$$
where $\Gamma_M = \text{Ь}(\Gamma)$, $\Gamma_N = \text{Ь}(\Gamma)$, $\Gamma_P = \text{Ь}(\Gamma)$

[C-Par]
$$\frac{\begin{array}{c}\Gamma_P, \Gamma^Q \vdash P \rightsquigarrow \psi_p; \phi_p \\ \Gamma_Q, \Gamma^P \vdash Q \rightsquigarrow \psi_q; \phi_q\end{array}}{\begin{array}{l}\Gamma \vdash P \mid Q \rightsquigarrow \psi; \phi_p \wedge \phi_q \wedge \phi^p \wedge \phi^q \wedge [\![\psi' = \psi'_p \otimes \psi'_q]\!] \\ \quad \wedge [\![\psi_p = \psi'_p \otimes \psi^{q\prime}]\!] \wedge [\![\psi^{q\prime} \leq \psi^q]\!] \\ \quad \wedge [\![\psi_q = \psi'_q \otimes \psi^{p\prime}]\!] \wedge [\![\psi^{p\prime} \leq \psi^p]\!] \\ \quad \bigwedge_{\substack{\Gamma(x)=\tau_1 \\ \Gamma_P(x)=\tau_2 \\ \Gamma_Q(x)=\tau_3}} [\![\tau_1 = \tau_2 + \tau_3]\!] \\ \quad \wedge [\![\mathsf{if}\ \mathcal{C}(P \mid Q, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]\end{array}}$$
where $\Gamma_P = \text{Ь}(\Gamma)$, $\Gamma_Q = \text{Ь}(\Gamma)$, $\mathcal{F}'(P) = \langle \Gamma^P, \psi^p, \phi^p \rangle$, $\mathcal{F}'(Q) = \langle \Gamma^Q, \psi^q, \phi^q \rangle$

[C-Rep]
$$\frac{\Gamma \vdash P \rightsquigarrow \psi'; \phi_p}{\begin{array}{l}\Gamma \vdash *P \rightsquigarrow \psi; \phi_p \wedge [\![\psi' = \psi' \otimes \psi']\!] \\ \quad \bigwedge_{\tau \in \mathsf{ran}(\Gamma)} [\![\tau = \tau + \tau]\!] \\ \quad \wedge [\![\mathsf{if}\ \mathcal{C}(*P, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]\end{array}}$$

[C-Res]
$$\frac{\Gamma, x : \tau \vdash P \rightsquigarrow \psi_p; \phi_p}{\begin{array}{l}\Gamma \vdash (\nu x : \tau)P \rightsquigarrow \psi; \phi_p \wedge [\![\psi' = \psi_p \div x]\!] \\ \quad \wedge [\![\mathsf{if}\ \mathcal{C}((\nu x : \tau)P, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]\end{array}}$$

[C-Ass]
$$\frac{\Gamma \vdash \Psi \rightsquigarrow \psi_a; \phi_a}{\Gamma \vdash (\!|\Psi|\!) \rightsquigarrow \psi; \phi_a \wedge [\![\mathsf{if}\ \mathcal{C}((\!|\Psi|\!), \psi_a)\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi_a)\ \mathsf{else}\ \psi = \psi_a]\!]}$$

[C-Cas]
$$\frac{\begin{array}{c}\Gamma \vdash \sigma_i \rightsquigarrow \psi_{si}; \phi_{si} \\ \Gamma \vdash P_i \rightsquigarrow \psi_{pi}; \phi_{pi}\end{array}}{\begin{array}{l}\Gamma \vdash \mathbf{case}\ \sigma_1 : P_1, \ldots, \sigma_k : P_k \rightsquigarrow \psi; \bigwedge_i(\phi_{si} \wedge \phi_{pi}) \\ \quad \wedge [\![\psi_{s1} = \cdots = \psi_{sk} = \psi_{p1} = \cdots = \psi_{pk} = \psi']\!] \\ \quad \wedge [\![\mathsf{if}\ \mathcal{C}(\mathbf{case}\ \sigma_1 : P_1, \ldots, \sigma_k : P_k, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]\end{array}}$$
where $1 \leq i \leq k$

[C-Pat]
$$\frac{\Gamma, \vec{x} : \vec{\tau} \vdash M \rightsquigarrow \tau_m; \psi_m; \phi_m}{\Gamma \vdash (\lambda\vec{x})M \rightsquigarrow (\vec{\tau} \to \tau_m); \psi_m; \phi_m}$$

TABLE 4.2.1. Constraint generation for linear types

Since we do not know when [T-Wea] may be used, we need to consider the possibility of it occurring between every syntax-directed rule, which is why the above encoding is found in all the rules.

Criteria for constraint generation. If we are to prove anything about our type inference, we need to define certain criteria regarding how to instantiate it. These greatly resemble the one seen in Theorem 3.2.4, but instead of just an encoding of $[\![\tau_1 \leftrightarrows \tau_2]\!]$ we have multiple different encodings. In order to ensure that the constraints generated in Table 4.2.1 are

enough, we need to ensure that a solution to such constraints, $\mathscr{L}(\phi)$, and the instantiation function, $\rhd$, not only models the constraints, but also respect the operators $+$, $\otimes$, $\div$, and so forth. We thus for instance require that "for any constraint generated by a process, ϕ, and any assertion variables, $\psi_1, \ldots, \psi_n$, if $\mathscr{L}(\phi) \rhd \psi_1 = \mathscr{L}(\phi) \rhd \psi_2 \otimes \cdots \otimes \mathscr{L}(\phi) \rhd \psi_n$ then $\mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \otimes \cdots \otimes \psi_n]\!])$ exists". The complete list of criteria is listed below, and should not be surprising.

(1) For any term M, there exist a type environment Γ, an assertion Ψ, and a type T such that $\Gamma, \Psi \vdash_{\min} M : T$ if and only if $\Gamma_M \vdash M \rightsquigarrow \tau; \psi; \phi$ such that $\mathscr{L}(\phi)$ is defined, $\mathscr{L}(\phi) \rhd \tau = T$, $\mathscr{L}(\phi) \rhd \psi = \Psi$, $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_M \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_M(x)$
(2) For any condition σ, there exists a type environment Γ and an assertion Ψ such that $\Gamma, \Psi \vdash \sigma$, if and only if $\Gamma_\sigma \vdash \sigma \rightsquigarrow \psi; \phi$ such that $\mathscr{L}(\phi)$ is defined, $\mathscr{L}(\phi) \rhd \psi = \Psi$ and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_\sigma \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_\sigma(x)$
(3) For any assertion Ψ, there exists a type environment Γ and an assertion Ψ' such that $\Gamma, \Psi' \vdash \Psi$, if and only if $\Gamma_\Psi \vdash \Psi \rightsquigarrow \psi; \phi$ such that $\mathscr{L}(\phi)$ is defined, $\mathscr{L}(\phi) \rhd \psi = \Psi'$ and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_\Psi \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_\Psi(x)$
(4) For any type variables τ_1 and τ_2, and any constraint ϕ, if $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle^- \tau_2]\!])$ is defined, then $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle^- \tau_2]\!]) \rhd \tau_1 \leftarrowtriangle^- \mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle^- \tau_2]\!]) \rhd \tau_2$.
(5) For any constraint generated by a process, ϕ, and any type variables, τ_1 and τ_2, if $\mathscr{L}(\phi) \rhd \tau_1 \leftarrowtriangle^- \mathscr{L}(\phi) \rhd \tau_2$ then $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle^- \tau_2]\!])$ exists.
(6) For any type variables τ_1 and τ_2, and any constraint ϕ, if $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle^+ \tau_2]\!])$ is defined, then $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle^+ \tau_2]\!]) \rhd \tau_1 \leftarrowtriangle^+ \mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle^+ \tau_2]\!]) \rhd \tau_2$.
(7) For any constraint generated by a process, ϕ, and any type variables, τ_1 and τ_2, if $\mathscr{L}(\phi) \rhd \tau_1 \leftarrowtriangle^+ \mathscr{L}(\phi) \rhd \tau_2$ then $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle^+ \tau_2]\!])$ exists.
(8) For any type variables $\tau_1, \ldots, \tau_n$, and any constraint ϕ, if $\mathscr{L}(\phi \wedge [\![\tau_1 = \tau_2 + \cdots + \tau_n]\!])$ is defined, then $\mathscr{L}(\phi \wedge [\![\tau_1 = \tau_2 + \cdots + \tau_n]\!]) \rhd \tau_1 = \mathscr{L}(\phi \wedge [\![\tau_1 = \tau_2 + \cdots + \tau_n]\!]) \rhd \tau_2 + \cdots + \mathscr{L}(\phi \wedge [\![\tau_1 = \tau_2 + \cdots + \tau_n]\!]) \rhd \tau_n$.
(9) For any constraint generated by a process, ϕ, and any type variables, $\tau_1, \ldots, \tau_n$, if $\mathscr{L}(\phi) \rhd \tau_1 = \mathscr{L}(\phi) \rhd \tau_2 + \cdots + \mathscr{L}(\phi) \rhd \tau_n$ then $\mathscr{L}(\phi \wedge [\![\tau_1 = \tau_2 + \cdots + \tau_n]\!])$ exists.
(10) For any assertion variables $\psi_1, \ldots, \psi_n$, and any constraint ϕ, if $\mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \otimes \cdots \otimes \psi_n]\!])$ is defined, then $\mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \otimes \cdots \otimes \psi_n]\!]) \rhd \psi_1 = \mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \otimes \cdots \otimes \psi_n]\!]) \rhd \psi_2 \otimes \cdots \otimes \mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \otimes \cdots \otimes \psi_n]\!]) \rhd \psi_n$.
(11) For any constraint generated by a process, ϕ, and any assertion variables, $\psi_1, \ldots, \psi_n$, if $\mathscr{L}(\phi) \rhd \psi_1 = \mathscr{L}(\phi) \rhd \psi_2 \otimes \cdots \otimes \mathscr{L}(\phi) \rhd \psi_n$ then $\mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \otimes \cdots \otimes \psi_n]\!])$ exists.
(12) For any assertion variables ψ_1 and ψ_2, and any constraint ϕ, if $\mathscr{L}(\phi \wedge [\![\psi_1 \leq \psi_2]\!])$ is defined, then $\mathscr{L}(\phi \wedge [\![\psi_1 \leq \psi_2]\!]) \rhd \psi_1 \leq \mathscr{L}(\phi \wedge [\![\psi_1 \leq \psi_2]\!]) \rhd \psi_2$.
(13) For any constraint generated by a process, ϕ, and any assertion variables, ψ_1 and ψ_2, if $\mathscr{L}(\phi) \rhd \psi_1 \leq \mathscr{L}(\phi) \rhd \psi_2$ then $\mathscr{L}(\phi \wedge [\![\psi_1 \leq \psi_2]\!])$ exists.
(14) For any assertion variables $\psi_1, \ldots, \psi_n$, and any constraint ϕ, if $\mathscr{L}(\phi \wedge [\![\psi_1 = \cdots = \psi_n]\!])$ is defined, then $\mathscr{L}(\phi \wedge [\![\psi_1 = \cdots = \psi_n]\!]) \rhd \psi_1 = \cdots = \mathscr{L}(\phi \wedge [\![\psi_1 = \cdots = \psi_n]\!]) \rhd \psi_n$.
(15) For any constraint generated by a process, ϕ, and any assertion variables, $\psi_1, \ldots, \psi_n$, if $\mathscr{L}(\phi) \rhd \psi_1 = \cdots = \mathscr{L}(\phi) \rhd \psi_n$ then $\mathscr{L}(\phi \wedge [\![\psi_1 = \cdots = \psi_n]\!])$ exists.
(16) For any assertion variables ψ_1 and ψ_2, any $x \in \mathcal{N}$, and any constraint ϕ, if $\mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \div x]\!])$ is defined, then $\mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \div x]\!]) \rhd \psi_1 = \mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \div x]\!]) \rhd \psi_2 \div x$.
(17) For any constraint generated by a process, ϕ, any assertion variables, ψ_1 and ψ_2, and any $x \in \mathcal{N}$, if $\mathscr{L}(\phi) \rhd \psi_1 = \mathscr{L}(\phi) \rhd \psi_2 \div x$ then $\mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \div x]\!])$ exists.

(18) For any assertion variables ψ_1 and ψ_2, any process P, and any constraint ϕ, if $\mathscr{L}(\phi \wedge [\![\mathsf{if}\ \mathcal{C}(P,\psi_1)\ \mathsf{then}\ \mathsf{Weak}(\psi_2,\psi_1)\ \mathsf{else}\ \psi_2 = \psi_1]\!])$ is defined, then either $\mathcal{C}(P, \mathscr{L}(\phi \wedge [\![\mathsf{if}\ \mathcal{C}(P,\psi_1)\ \mathsf{then}\ \mathsf{Weak}(\psi_2,\psi_1)\ \mathsf{else}\ \psi_2 = \psi_1]\!]) \rhd \psi_1)$ holds and $\mathscr{L}(\phi \wedge [\![\mathsf{if}\ \mathcal{C}(P,\psi_1)\ \mathsf{then}\ \mathsf{Weak}(\psi_2,\psi_1)\ \mathsf{else}\ \psi_2 = \psi_1]\!]) \rhd \psi_1 = \mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \div x]\!]) \rhd \psi_2 \otimes \Psi_3$ for some Ψ_3 allowed by [T-Weak] or $\mathscr{L}(\phi \wedge [\![\mathsf{if}\ \mathcal{C}(P,\psi_1)\ \mathsf{then}\ \mathsf{Weak}(\psi_2,\psi_1)\ \mathsf{else}\ \psi_2 = \psi_1]\!]) \rhd \psi_1 = \mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \div x]\!]) \rhd \psi_2$.

(19) For any constraint generated by a process, ϕ, any assertion variables, ψ_1 and ψ_2, and any process P, if either $\mathcal{C}(P, \mathscr{L}(\phi \wedge [\![\mathsf{if}\ \mathcal{C}(P,\psi_1)\ \mathsf{then}\ \mathsf{Weak}(\psi_2,\psi_1)\ \mathsf{else}\ \psi_2 = \psi_1]\!]) \rhd \psi_1)$ holds and $\mathscr{L}(\phi \wedge [\![\mathsf{if}\ \mathcal{C}(P,\psi_1)\ \mathsf{then}\ \mathsf{Weak}(\psi_2,\psi_1)\ \mathsf{else}\ \psi_2 = \psi_1]\!]) \rhd \psi_1 = \mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \div x]\!]) \rhd \psi_2 \otimes \Psi_3$ for some Ψ_3 allowed by [T-Weak] or $\mathscr{L}(\phi \wedge [\![\mathsf{if}\ \mathcal{C}(P,\psi_1)\ \mathsf{then}\ \mathsf{Weak}(\psi_2,\psi_1)\ \mathsf{else}\ \psi_2 = \psi_1]\!]) \rhd \psi_1 = \mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \div x]\!]) \rhd \psi_2$ then $\mathscr{L}(\phi \wedge [\![\psi_1 = \psi_2 \div x]\!])$ exists.

Now that these criteria have been established, we put forth Theorem 4.2.4 and prove it for any instantiation of the type generation which satisfies the above criteria in Appendix B.2.

Theorem 4.2.4. *For any process P, there exists a type environment Γ and an assertion Ψ such that $\Gamma, \Psi \vdash P$, if and only if $\Gamma_P \vdash P \rightsquigarrow \psi; \phi$ such that $\mathscr{L}(\phi)$ is defined, $\mathscr{L}(\phi)\psi \rhd \Psi$, and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P$ and $\mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_P(x)$*

4.3. Simple linear types as a ψ-calculus

In this section we present an encoding of the simple linearly typed π-calculus as a typed ψ-calculus. We choose the type system originally introduced by Kobayashi et al. in [15]. The type system is simple in the regard that we only have channel types and denote non-channels by channel types with no capabilities i.e. they cannot be used for either input or output. In addition each type is associated with a multiplicity: either the type has the multiplicity 1 which denotes that it can (and must) be used exactly once or it can have the multiplicity ω which allows the channel to be used freely (or not even used at all).

The nominal data types and operations, the type system, the syntax of the π-calculus, and finally the encoding of the π-calculus as a ψ-calculus are presented in Tables 4.3.1 to 4.3.4.

We also need to define the typing rules for terms, assertions, and conditions. However, since we have no assertions—aside from **1**—there are no additional typing rules for assertions. For conditions we have simply written b in the [If]-encoding, and the actual typing rules would depend on what one is allowed to write in b. For this example however this is of little importance, and we will simply leave the typing rules for conditions undefined. Finally for terms we have only a single simple rule as shown in Table 4.3.5. For simplicity we also include the constraint generation rule for variables in the same table.

We now proceed to define the required constraint encodings as seen in Table 4.3.6. Since we have no assertions for this simple type system, all constraints related to assertions are simply ignored.

The last required definition, namely the one for the type and assertions reconstruction function is defined by the rules seen in Table 4.3.7. It is worth noticing there are no

T:	$\mathcal{N}$
C:	$\{a \leftrightarrow b \mid a, b \in \mathbf{T}\}$
A:	**1**
$\otimes$:	undefined
1:	**1**
$\vDash$:	undefined

TABLE 4.3.1. Nominal data types and operations

$T ::=$		Type
	$p^m T$	channel type
$p ::=$		Capability
	$\emptyset$	no capability
	$\{!\}$	output capability
	$\{?\}$	input capability
	$\{!,?\}$	both input and output capability
$m ::=$		Multiplicity
	1	linear
	ω	unlimited

TABLE 4.3.2. Types for the ψ-calculus encoding the π_F-calculus

$P ::=$		Process
	$\mathbf{0}$	inactivity
	$P \mid P$	parallel composition
	$x!\vec{x}.P$	output
	$x?\vec{x}.P$	input
	$(\nu x : T)P$	restriction
	$*x?\vec{x}$	replicated input
	$\mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$	if-then-else construct
x		Name

TABLE 4.3.3. The syntax of the simple linearly typed π-calculus

[Nil]	$[\![\mathbf{0}]\!] = (\!|\mathbf{1}|\!)$	[Par]	$[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$
[Out]	$[\![x!\vec{y}.P]\!] = \overline{x}\,\vec{y}.[\![P]\!]$	[In]	$[\![x?\vec{y}.P]\!] = x(\lambda\vec{y})\vec{y}.[\![P]\!]$
[Res]	$[\![(\nu x : T)P]\!] = (\nu x : T)[\![P]\!]$		
[If]	$[\![\mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2]\!] = \mathbf{case}\ b : [\![P_1]\!]\ ,\ \neg b : [\![P_2]\!]$		

TABLE 4.3.4. Encoding of π in ψ

[Var]	$\dfrac{\Gamma(n) = T}{\Gamma \vdash n : T}$
[Ast]	undefined
[Con]	undefined
[Var-C]	$\dfrac{\Gamma(n) = \tau}{\Gamma \vdash n : \tau \rightsquigarrow \tau; \psi; \mathtt{T}}$

TABLE 4.3.5. Typing rules for terms, assertions, and conditions

reconstruction rules for assertions as we have no assertions in this simple type system. The remaining rules are straight forward and should not require any additional explanations.

$[\![\tau_1 \leftarrow\!\!\rho^{-} \tau_2]\!] = \mathsf{In}(\tau_1, \tau_2)$	$[\![\tau_1 \leftarrow\!\!\rho^{+} \tau_2]\!] = \mathsf{Out}(\tau_1, \tau_2)$
$[\![\tau_1 = \tau_2 + \tau_3]\!] = \mathsf{Add2}(\tau_1, \tau_2, \tau_3)$	$[\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!] = \mathsf{Add3}(\tau_1, \tau_2, \tau_3, \tau_4)$

TABLE 4.3.6. Constraint encoding

$$\frac{\mathsf{Emp}(\tau) \notin \Psi \qquad \Phi \overset{cap}{\rhd} \tau = p \qquad \Phi \overset{mul}{\rhd} \tau = m \qquad \Phi \overset{val}{\rhd} \tau = T}{\Phi \rhd \tau = p^m T}$$

$$\frac{\mathsf{Emp}(\tau) \in \Psi}{\Phi \rhd \tau = \varepsilon}$$

$$\frac{\nexists \mathsf{In}(\tau, \tau') \in \Phi \qquad \nexists \mathsf{Out}(\tau, \tau'') \in \Phi}{\Phi \overset{cap}{\rhd} \tau = \emptyset}$$

$$\frac{\mathsf{In}(\tau, \tau') \in \Phi \qquad \nexists \mathsf{Out}(\tau, \tau'') \in \Phi}{\Phi \overset{cap}{\rhd} \tau = \{?\}}$$

$$\frac{\nexists \mathsf{In}(\tau, \tau') \in \Phi \qquad \mathsf{Out}(\tau, \tau'') \in \Phi}{\Phi \overset{cap}{\rhd} \tau = \{!\}}$$

$$\frac{\mathsf{In}(\tau, \tau') \in \Phi \qquad \mathsf{Out}(\tau, \tau'') \in \Phi}{\Phi \overset{cap}{\rhd} \tau = \{!, ?\}}$$

$$\frac{\mathsf{In}(\tau, \tau') \in \Phi \qquad \Phi \rhd \tau' = T}{\Phi \overset{val}{\rhd} \tau = T} \qquad \frac{\nexists \mathsf{Un}(\tau) \in \Phi}{\Phi \overset{mul}{\rhd} \tau = 1}$$

$$\frac{\mathsf{Out}(\tau, \tau') \in \Phi \qquad \Phi \rhd \tau' = T}{\Phi \overset{val}{\rhd} \tau = T} \qquad \frac{\mathsf{Un}(\tau) \in \Phi}{\Phi \overset{mul}{\rhd} \tau = \omega}$$

$$\frac{\nexists \mathsf{In}(\tau, \tau') \in \Phi \qquad \nexists \mathsf{Out}(\tau, \tau'') \in \Phi}{\Phi \overset{val}{\rhd} \tau = \varepsilon}$$

TABLE 4.3.7. Type and assertions reconstruction function

We now only need to define the axioms used to derive the solution to the constraints and thus the ability to infer the types. We present the axioms in Table 4.3.8. We introduce the constraints NotEmp and Emp to denote if we are allowed to assign the type ε to a type variable. For instance if we assume we have some process $\underline{a}(\lambda x)x.P$ where $x \notin \mathsf{names}(P)$ we could accidentally end up assigning the empty type to x, since we never generate any type constraints for x. This would however be a mistake; recall that we use ε to denote that a name should not appear in an environment, and obviously x should appear in the environment. Not only should x appear in the environment used to type check P, it should appear in at least one chain of environments all the way down to a $\mathbf{1}$. This is the case as we cannot remove names arbitrarily from environments. The constraint $\mathsf{NotEmp}(\tau)$ denotes that ε cannot be assigned to the type variable τ whereas $\mathsf{Emp}(\tau)$ allows this assignment.

$\forall a : \mathsf{Eq}(a, a)$ $\forall a : \forall b : \mathsf{Eq}(a, b) \Rightarrow \mathsf{Eq}(b, a)$ $\forall a : \forall b : \forall c : (\mathsf{Eq}(a, b) \wedge \mathsf{Eq}(b, c)) \Rightarrow \mathsf{Eq}(a, c)$
$\forall t : \forall t_1 : \forall t_2 : (\mathsf{In}(t, t_1) \wedge \mathsf{In}(t, t_2)) \Rightarrow \mathsf{Eq}(t_1, t_2)$ $\forall t : \forall t_1 : \forall t_2 : (\mathsf{Out}(t, t_1) \wedge \mathsf{Out}(t, t_2)) \Rightarrow \mathsf{Eq}(t_1, t_2)$ $\forall t : \forall t_1 : \forall t_2 : (\mathsf{In}(t, t_1) \wedge \mathsf{Out}(t, t_2)) \Rightarrow \mathsf{Eq}(t_1, t_2)$
$\forall t_1 : \forall t_2 : \forall t' : (\mathsf{In}(t_1, t') \wedge \mathsf{Eq}(t_1, t_2)) \Rightarrow \mathsf{In}(t_2, t')$ $\forall t_1 : \forall t_2 : \forall t' : (\mathsf{Out}(t_1, t') \wedge \mathsf{Eq}(t_1, t_2)) \Rightarrow \mathsf{Out}(t_2, t')$ $\forall t_1 : \forall t_2 : (\mathsf{NotEmp}(t_1) \wedge \mathsf{Eq}(t_1, t_2)) \Rightarrow \mathsf{NotEmp}(t_2)$
$\forall t : \forall t' : \mathsf{In}(t, t') \Rightarrow \mathsf{NotEmp}(t')$ $\forall t : \forall t' : \mathsf{Out}(t, t') \Rightarrow \mathsf{NotEmp}(t')$
$\forall t_1 : \forall t_2 : \forall t_3 : (\mathsf{Add2}(t_1, t_2, t_3) \wedge \mathsf{NotEmp}(t_2)) \Rightarrow \mathsf{NotEmp}(t_1)$ $\forall t_1 : \forall t_2 : \forall t_3 : (\mathsf{Add2}(t_1, t_2, t_3) \wedge \mathsf{NotEmp}(t_3)) \Rightarrow \mathsf{NotEmp}(t_1)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : (\mathsf{Add2}(t_1, t_2, t_3) \wedge \mathsf{NotEmp}(t_1) \wedge \neg\mathsf{NotEmp}(t_2)) \Rightarrow \mathsf{NotEmp}(t_3)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : (\mathsf{Add3}(t_1, t_2, t_3, t_4) \wedge \mathsf{NotEmp}(t_2)) \Rightarrow \mathsf{NotEmp}(t_1)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : (\mathsf{Add3}(t_1, t_2, t_3, t_4) \wedge \mathsf{NotEmp}(t_3)) \Rightarrow \mathsf{NotEmp}(t_1)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : (\mathsf{Add3}(t_1, t_2, t_3, t_4) \wedge \mathsf{NotEmp}(t_4)) \Rightarrow \mathsf{NotEmp}(t_1)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : (\mathsf{Add3}(t_1, t_2, t_3, t_4) \wedge \mathsf{NotEmp}(t_1)$ $\wedge\neg\mathsf{NotEmp}(t_2) \wedge \neg\mathsf{NotEmp}(t_3)) \Rightarrow \mathsf{NotEmp}(t_4)$
$\forall t_1 : \forall t_2 : \forall t_3 : \forall t : (\mathsf{Add2}(t_1, t_2, t_3) \wedge \mathsf{In}(t_2, t)) \Rightarrow \mathsf{In}(t_1, t)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t : (\mathsf{Add2}(t_1, t_2, t_3) \wedge \mathsf{In}(t_3, t)) \Rightarrow \mathsf{In}(t_1, t)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t : (\mathsf{Add2}(t_1, t_2, t_3) \wedge \mathsf{Out}(t_2, t)) \Rightarrow \mathsf{Out}(t_1, t)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t : (\mathsf{Add2}(t_1, t_2, t_3) \wedge \mathsf{Out}(t_3, t)) \Rightarrow \mathsf{Out}(t_1, t)$
$\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : \forall t : (\mathsf{Add3}(t_1, t_2, t_3, t_4) \wedge \mathsf{In}(t_2, t)) \Rightarrow \mathsf{In}(t_1, t)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : \forall t : (\mathsf{Add3}(t_1, t_2, t_3, t_4) \wedge \mathsf{In}(t_3, t)) \Rightarrow \mathsf{In}(t_1, t)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : \forall t : (\mathsf{Add3}(t_1, t_2, t_3, t_4) \wedge \mathsf{In}(t_4, t)) \Rightarrow \mathsf{In}(t_1, t)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : \forall t : (\mathsf{Add3}(t_1, t_2, t_3, t_4) \wedge \mathsf{Out}(t_2, t)) \Rightarrow \mathsf{Out}(t_1, t)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : \forall t : (\mathsf{Add3}(t_1, t_2, t_3, t_4) \wedge \mathsf{Out}(t_3, t)) \Rightarrow \mathsf{Out}(t_1, t)$ $\forall t_1 : \forall t_2 : \forall t_3 : \forall t_4 : \forall t : (\mathsf{Add3}(t_1, t_2, t_3, t_4) \wedge \mathsf{Out}(t_4, t)) \Rightarrow \mathsf{Out}(t_1, t)$
$\forall t : \forall t_1 : \forall t_2 : (\mathsf{In}(t, t_1) \wedge \mathsf{In}(t, t_2) \wedge t_1 \neq t_2) \Rightarrow \mathsf{Un}(t)$ $\forall t : \forall t_1 : \forall t_2 : (\mathsf{Out}(t, t_1) \wedge \mathsf{Out}(t, t_2) \wedge t_1 \neq t_2) \Rightarrow \mathsf{Un}(t)$ $\forall t : \mathsf{Add2}(t, t, t) \Rightarrow \mathsf{Un}(t)$ $\forall t : \mathsf{Add3}(t, t, t, t) \Rightarrow \mathsf{Un}(t)$
$\forall t : ((\forall t' : \neg\mathsf{In}(t, t')) \wedge (\forall t' : \neg\mathsf{Out}(t, t')) \wedge \neg\mathsf{NotEmp}(t)) \Rightarrow \mathsf{Emp}(t)$
$\forall t_1 : \forall t_2 : (\mathsf{In}(t_1, t_2) \wedge \mathsf{Eq}(t_1, t_2)) \Rightarrow \mathsf{Fail}(t_1)$ $\forall t_1 : \forall t_2 : (\mathsf{Out}(t_1, t_2) \wedge \mathsf{Eq}(t_1, t_2)) \Rightarrow \mathsf{Fail}(t_1)$

TABLE 4.3.8. Axioms for simple linearly types

4.3.1. Example. Let us now consider the simple process

$$P = \overline{x}\, y.(\!|\mathbf{1}|\!) \mid \underline{x}(\lambda a)a.\overline{a}\, b.\overline{a}\, c.(\!|\mathbf{1}|\!) \mid \underline{y}(\lambda d)d.\underline{y}(\lambda e)e.(\!|\mathbf{1}|\!)$$

for which we wish to conduct type inference.

In order to do this we first construct the minimal typing environment for the process, using the free names of P, giving us

$$\Gamma = x : \tau_1, y : \tau_2, b : \tau_3, c : \tau_4.$$

Using the constraint generation rules of Tables 4.2.1 and 4.3.5 we generate the following constraints:

$\mathsf{Add3}(\tau_1, \tau_1^1, \tau_1^2, \tau_1^3)$
$\mathsf{Add3}(\tau_2, \tau_2^1, \tau_2^2, \tau_2^3)$
$\mathsf{Add3}(\tau_3, \tau_3^1, \tau_3^2, \tau_3^3)$
$\mathsf{Add3}(\tau_4, \tau_4^1, \tau_4^2, \tau_4^3)$

$\overline{x}\,y.(\!\|\mathbf{1}\|\!)$	$\underline{x}(\lambda a)a.\overline{a}\,b.\overline{a}\,c.(\!\|\mathbf{1}\|\!)$	$\underline{y}(\lambda d)d.\underline{y}(\lambda e)e.(\!\|\mathbf{1}\|\!)$
$\mathsf{Out}(\tau_1^{11}, \tau_2^{12})$	$\mathsf{In}(\tau_1^{21}, \tau_5)$	$\mathsf{In}(\tau_2^{31}, \tau_6)$
$\mathsf{Add3}(\tau_1^1, \tau_1^{11}, \tau_1^{12}, \tau_1^{13})$	$\mathsf{Add3}(\tau_1^2, \tau_1^{21}, \tau_1^{22}, \tau_1^{23})$	$\mathsf{Add3}(\tau_1^3, \tau_1^{31}, \tau_1^{32}, \tau_1^{33})$
$\mathsf{Add3}(\tau_2^1, \tau_2^{11}, \tau_2^{12}, \tau_2^{13})$	$\mathsf{Add3}(\tau_2^2, \tau_2^{21}, \tau_2^{22}, \tau_2^{23})$	$\mathsf{Add3}(\tau_2^3, \tau_2^{31}, \tau_2^{32}, \tau_2^{33})$
$\mathsf{Add3}(\tau_3^1, \tau_3^{11}, \tau_3^{12}, \tau_3^{13})$	$\mathsf{Add3}(\tau_3^2, \tau_3^{21}, \tau_3^{22}, \tau_3^{23})$	$\mathsf{Add3}(\tau_3^3, \tau_3^{31}, \tau_3^{32}, \tau_3^{33})$
$\mathsf{Add3}(\tau_4^1, \tau_4^{11}, \tau_4^{12}, \tau_4^{13})$	$\mathsf{Add3}(\tau_4^2, \tau_4^{21}, \tau_4^{22}, \tau_4^{23})$	$\mathsf{Add3}(\tau_4^3, \tau_4^{31}, \tau_4^{32}, \tau_4^{33})$
	$\mathsf{Out}(\tau_5^{231}, \tau_3^{232})$	$\mathsf{In}(\tau_2^{331}, \tau_7)$
	$\mathsf{Add3}(\tau_1^{23}, \tau_1^{231}, \tau_1^{232}, \tau_1^{233})$	$\mathsf{Add3}(\tau_1^{33}, \tau_1^{331}, \tau_1^{332}, \tau_1^{333})$
	$\mathsf{Add3}(\tau_2^{23}, \tau_2^{231}, \tau_2^{232}, \tau_2^{233})$	$\mathsf{Add3}(\tau_2^{33}, \tau_2^{331}, \tau_2^{332}, \tau_2^{333})$
	$\mathsf{Add3}(\tau_3^{23}, \tau_3^{231}, \tau_3^{232}, \tau_3^{233})$	$\mathsf{Add3}(\tau_3^{33}, \tau_3^{331}, \tau_3^{332}, \tau_3^{333})$
	$\mathsf{Add3}(\tau_4^{23}, \tau_4^{231}, \tau_4^{232}, \tau_4^{233})$	$\mathsf{Add3}(\tau_4^{33}, \tau_4^{331}, \tau_4^{332}, \tau_4^{333})$
	$\mathsf{Add3}(\tau_5, \tau_5^{231}, \tau_5^{232}, \tau_5^{233})$	$\mathsf{Add3}(\tau_6, \tau_6^{331}, \tau_6^{332}, \tau_6^{333})$
	$\mathsf{Out}(\tau_5^{2331}, \tau_4^{2332})$	
	$\mathsf{Add3}(\tau_1^{233}, \tau_1^{2331}, \tau_1^{2332}, \tau_1^{2333})$	
	$\mathsf{Add3}(\tau_2^{233}, \tau_2^{2331}, \tau_2^{2332}, \tau_2^{2333})$	
	$\mathsf{Add3}(\tau_3^{233}, \tau_3^{2331}, \tau_3^{2332}, \tau_3^{2333})$	
	$\mathsf{Add3}(\tau_4^{233}, \tau_4^{2331}, \tau_4^{2332}, \tau_4^{2333})$	
	$\mathsf{Add3}(\tau_5^{233}, \tau_5^{2331}, \tau_5^{2332}, \tau_5^{2333})$	

Using the axioms of Table 4.3.8 we construct the solution found in Appendix A.2. In the solution we let t_abc_xyz denote the variable τ_{abc}^{xyz}. We can now use this solution, which we shall denote Φ and the function $\rhd$ to determine the types assigned to the type variables τ_1, τ_2, τ_3, and τ_4.

$x : \tau_1$**:** In order to compute $\Psi \rhd \tau_1$ we need to compute $\Phi \overset{cap}{\rhd} \tau_1$, $\Phi \overset{mul}{\rhd} \tau_1$, and $\Phi \overset{val}{\rhd} \tau_1$.

$\overset{cap}{\rhd}$**:** Since we have $\mathsf{In}(\tau_1, \tau_5) \in \Phi$ and $\mathsf{Out}(\tau_1, \tau_2^{12}) \in \Phi$ we can conclude that $\Phi \overset{cap}{\rhd} \tau_1 = \{!, ?\}$.

$\overset{mul}{\rhd}$**:** Since we have no $\mathsf{Un}(\tau_1)$ in Φ we can conclude $\Phi \overset{mul}{\rhd} \tau_1 = 1$.

$\overset{val}{\rhd}$**:** Since we have $\mathsf{In}(\tau_1, \tau_5) \in \Phi$ we need to compute $\Phi \rhd \tau_5$. This computation follows the same pattern, and we, for simplicity, immediately present the result, namely $\Phi \rhd \tau_5 = \{!\}^\omega(\emptyset^1\varepsilon)$. This implies that $\Phi \overset{val}{\rhd} \tau_1 = \{!\}^\omega(\emptyset^1\varepsilon)$.

Since we have computed the capabilities, multiplicity, and value type of τ_1 we can conclude that $\Phi \rhd \tau_1 = \{!, ?\}^1(\{!\}^\omega(\emptyset^1\varepsilon))$

$y : \tau_2$**:** In order to compute $\Psi \rhd \tau_2$ we need to compute $\Phi \overset{cap}{\rhd} \tau_2$, $\Phi \overset{mul}{\rhd} \tau_2$, and $\Phi \overset{val}{\rhd} \tau_2$.

$\overset{cap}{\rhd}$**:** Since we have $\mathsf{In}(\tau_2, \tau_6) \in \Phi$ and $\mathsf{Out}(\tau_2, \tau_4^{2332}) \in \Phi$ we can conclude that $\Phi \overset{cap}{\rhd} \tau_2 = \{!, ?\}$.

$\overset{mul}{\rhd}$**:** Since we have $\mathsf{Un}(\tau_2) \in \Phi$ we can conclude $\Phi \overset{mul}{\rhd} \tau_1 = \omega$.

$\overset{val}{\rhd}$**:** Since we have $\mathsf{In}(\tau_2, \tau_6) \in \Phi$ we need to compute $\Phi \rhd \tau_6$. This computation follows the same pattern, and we, for simplicity, immediately present the result, namely $\Phi \rhd \tau_6 = \emptyset^1\varepsilon$. This implies that $\Phi \overset{val}{\rhd} \tau_2 = \emptyset^1\varepsilon$.

Since we have computed the capabilities, multiplicity, and value type of τ_2 we can conclude that $\Phi \rhd \tau_2 = \{!, ?\}^\omega(\emptyset^1\varepsilon)$

$b : \tau_3$**:** This computation is similar to the above, and since we have no In, Out, or Un constraints for τ_3 we can easily conclude $\Phi \rhd \tau_3 = \emptyset^1\varepsilon$

$c : \tau_4$**:** This computation is similar to the above, and since we have no In, Out, or Un constraints for τ_4 we can easily conclude $\Phi \rhd \tau_4 = \emptyset^1\varepsilon$

$$[\mathsf{T\text{-}In}]\quad \frac{\begin{array}{l}\Gamma_1 + \vec{x} : \vec{T}, \Psi_1 \vdash P \\ \Gamma_2, \Psi_2 \vdash_{\min} (\lambda \vec{x}) N : \vec{T} \to U_o \\ \Gamma_3, \Psi_3 \vdash_{\min} M : U_s\end{array}}{\Gamma, \Psi \vdash \underline{M}(\lambda \vec{x}) N.P} \quad \text{where} \quad \begin{array}{l} U_s \leftrightarrow\!\!\!\!\rho^{-} U_o \\ \leftrightarrow\!\!\!\!\rho^{-} (\Psi, \Psi_1, \Psi_2, \Psi_3) \\ \Gamma = \Gamma_1 + \Gamma_2 + \Gamma_3 \\ \Psi = \Psi_1 \otimes \Psi_2 \otimes \Psi_3 \end{array}$$

$$[\mathsf{T\text{-}Out}]\quad \frac{\begin{array}{l}\Gamma_1, \Psi_1 \vdash_{\min} M : T_s \\ \Gamma_2, \Psi_2 \vdash_{\min} N : T_o \\ \Gamma_3, \Psi_3 \vdash P\end{array}}{\Gamma, \Psi \vdash \overline{M}\, N.P} \quad \text{where} \quad \begin{array}{l} T_s \leftrightarrow\!\!\!\!\rho^{+} T_o \\ \leftrightarrow\!\!\!\!\rho^{+} (\Psi, \Psi_1, \Psi_2, \Psi_3) \\ \Gamma = \Gamma_1 + \Gamma_2 + \Gamma_3 \\ \Psi = \Psi_1 \otimes \Psi_2 \otimes \Psi_3 \end{array}$$

TABLE 4.4.1. New typing rules for linearly typed ψ-calculi

4.4. Ensuring termination for value-passing as a ψ-calculus

In [13] Hüttel attempts to create an instantiation of the linearly typed ψ-calculi to ensure termination in value-passing processes inspired by [3], but we have found that this instantiation does not work, allowing the typing

$$a : \mathsf{Ch}^1, b : \mathsf{Ch}^1, c : 1, 3 \vdash *\underline{a}(\lambda x) x.\bar{b}\, x.(\!|\mathbf{1}|\!) \mid *\underline{b}(\lambda y) y.\bar{a}\, y.(\!|\mathbf{1}|\!) \mid \overline{a}\, c.(\!|\mathbf{1}|\!),$$

even though the process obviously does not terminate. One reason it does not work is because the definition of $\otimes$: $m \otimes \mathsf{chan}(m) = n$ for $n > m$ and $n \otimes k = \max(n, k)$ is not associative—$(\mathsf{chan}(1) \otimes 1) \otimes 2 \leq 2$ and $\mathsf{chan}(1) \otimes (1 \otimes 2)$ is undefined—as required by the definition of ψ-calculi. The replicated input rule he presents also cannot be derived from the typing rules for replication and input, and in addition cannot be used to ensure termination in the same way as in [3]. Finally, the proposition supposed to prove that the type system ensures termination would also hold for a type system, which types everything with the same type, which would obviously not ensure termination.

Since this instantiation does not work, we have instead attempted to make our own instantiation to ensure termination, but found it very tricky. The idea behind the type system is to assign a level to each type, and only in- and output on a channel guarded by replicated input on another, if the first channel has a lower level than the one used in the replicated input. Since we needed a way for the type system to know when an input or output directly followed replication, and to be able to ensure a correlation between the assertion used to type the channel being used and the rest of the process in an input or output. We also needed to prevent replicated output, parallel composition at so forth, so processes like $*\underline{a}(\lambda b) b.P \mid *\overline{a}\, c.Q$ could not be typed. The first we managed to do in the instantiation using a relation, $\leftrightarrow\!\!\!\!\rho\ (\Psi, \Psi_1, \Psi_2, \Psi_3)$, on the assertions. The relation is similar to $T \leftrightarrow\!\!\!\!\rho\ U$ for types, and is used to describe the relationship between the assertions for the type check of the process with the input/out, the channel, the term, and the remaining process respectively. As a result, we have altered the typing rules for input and output slightly, so that they now make use of this new relation, and we present those in Table 4.4.1. It is worth mentioning that the addition of this relation have no effect on the proofs of safety and subject reduction seen in [13]. As for preventing replication before anything other than input, e.g. preventing $*(P \mid Q)$, we resort to make a restriction on the syntax for this particular ψ-calculus, preventing such processes. While it might be possible to enforce this restriction using the type system itself, we have been unable to achieve this and thus simply disallow such processes instead. We therefore for the remainder of this section assume that replication can *only* occur before an input process i.e. as $*\underline{a}(\lambda x) x.P$.

We can then define an instantiation of the type system that ensures termination. The idea is to assign a level to each type such that channels only send types with a level that is at most equal to their own, and ensure that any channel used for input or output guarded by a replicated input would have a lower level than the one used for the replicated input.

We have a set of types defined by

$$T ::= \mathsf{Ch}^n \mid n$$

Where Ch^n is the type of channels allowed to send names of type m for $m \leq n$. We also define type addition as $T + T = T$ for all types T.

We then define the assertions used in this type system here:

$$\Psi ::= [n, m] \mid *[n, m] \mid \mathsf{chan}(n) \text{ where } n, m \in \mathbb{N}.$$

We use $[n, m]$ to assert a process which is not immediately preceded by and does not start with a replication wherein all types have a level of at most m, we use $*[n, m]$ to assert a process which is immediately preceded by or starts with a replication wherein all types have a level of at most m, and we use $\mathsf{chan}(n)$ to type channels of type Ch^n. In addition we use of $r[n, m]$ to denote that an assertion can be either $[n, m]$ or $*[n, m]$.

We then define our assertion compatibility predicate in Definition 4.4.1. It ensures that when $*[n, m]$ is used to assert input the $r_1[n_1, m_1]$ used to assert the rest of the process has a lower level m_1 than the $\mathsf{chan}(n_3)$ used to assert the channel the input is received on. The compatibility predicate for types, seen in Definition 4.4.2, ensures that channels only send names with a level smaller or equal to their own.

DEFINITION 4.4.1 (Assertion $\leftrightarrow\!\!\!\!\rho$). We define the compatibility of assertions like this:

$$\begin{array}{ll} \leftrightarrow\!\!\!\!\rho^- & (*[n, m], r[n_1, m_1], [n_2, m_2], \mathsf{chan}(n_3)) \quad \text{if } m_1 < n_3 \\ \leftrightarrow\!\!\!\!\rho^- & ([n, m], r[n_1, m_1], [n_2, m_2], \mathsf{chan}(n_3)) \\ \leftrightarrow\!\!\!\!\rho^+ & ([n, m], r[n_1, m_1], [n_2, m_2], \mathsf{chan}(n_3)) \end{array}$$

DEFINITION 4.4.2 (Type $\leftrightarrow\!\!\!\!\rho$). We define the compatibility of types like this:

$$\mathsf{Ch}^m \leftrightarrow\!\!\!\!\rho\ n \text{ where } m \geq n$$

The last thing we need to define before we present the typing rules for terms assertions and conditions is assertion composition, seen in Definition 4.4.3. These are designed to ensure that $*[n, m] \otimes *[n, m] = *[n, m]$ and $[n, m] \otimes [n, m]$ is not defined, but that these can otherwise be used the same ways.

DEFINITION 4.4.3 (Assertion composition). Let assertion composition be defined as

$$[n, m] = \begin{cases} \mathsf{chan}(n_1) \otimes \mathsf{chan}(n_2) & n_1, n_2 \leq m \\ \mathsf{chan}(n_1) \otimes r_2[n_2, m_2] & m_2 \leq m, \\ & n_1 \leq m, \\ & n_2 \leq m_2 \\ r_1[n_1, m_1] \otimes r_2[n_2, m_2] & m_1, m_2 \leq m, \\ & n_i \leq m_i, \\ & r_1[n_1, m_1] \neq r_2[n_2, m_2] \end{cases}$$

$$*[n, m] = \begin{cases} \mathsf{chan}(n_1) \otimes \mathsf{chan}(n_2) & n_1, n_2 \leq m \\ \mathsf{chan}(n_1) \otimes r_2[n_2, m_2] & m_2 \leq m, \\ & n_1 \leq m, \\ & n_2 \leq m_2 \\ r_1[n_1, m_1] \otimes r_2[n_2, m_2] & m_1, m_2 \leq m, \\ & n_i \leq m_i, \end{cases}$$

[Nam]	$\Gamma, [n,m] \vdash x : T$	if $\Gamma(x) = T$
[Cha]	$\Gamma, \mathsf{chan}(n) \vdash a : \mathsf{Ch}^n$	if $\Gamma(a) = \mathsf{Ch}^n$
[Com]	$\dfrac{\Gamma, [n,m] \vdash M_i : n_i \qquad 1 \le i \le \lvert\vec{M}\rvert}{\Gamma, [n,m] \vdash f(\vec{M}) : n}$	if $n_i \le n$ for $1 \le i \le \lvert\vec{M}\rvert$

TABLE 4.4.2. Typing rules for terms, assertions, and conditions

[C-In]
$$\frac{\begin{array}{l}\Gamma_M \vdash M \rightsquigarrow \tau_m; \psi_m; \phi_m \\ \Gamma_N \vdash (\lambda\vec{x})N \rightsquigarrow (\vec{\tau} \to \tau_n); \psi_n; \phi_n \\ \Gamma_P, \vec{x} : \vec{\tau} \vdash P \rightsquigarrow \psi_p; \phi_p\end{array}}{\begin{array}{l}\Gamma \vdash \underline{M}(\lambda\vec{x})N.P \rightsquigarrow \psi; \phi_m \wedge \phi_n \wedge \phi_p \\ \quad \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftharpoonup^{\rho^-} \tau_n]\!] \\ \quad \bigwedge_{\substack{\Gamma(x)=\tau_1 \\ \Gamma_M(x)=\tau_2 \\ \Gamma_N(x)=\tau_3 \\ \Gamma_P(x)=\tau_4}} [\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!] \\ \quad \wedge [\![\mathsf{if}\ \mathcal{C}(\underline{M}(\lambda\vec{x})N.P, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!] \\ \quad \wedge [\![\leftharpoonup^{\rho^-} (\psi, \psi_p, \psi_n, \psi_m)]\!]\end{array}} \quad \text{where} \quad \begin{array}{l}\Gamma_M = Ь(\Gamma) \\ \Gamma_N = Ь(\Gamma) \\ \Gamma_P = Ь(\Gamma)\end{array}$$

[C-Out]
$$\frac{\begin{array}{l}\Gamma_M \vdash M \rightsquigarrow \tau_m; \psi_m; \phi_m \\ \Gamma_N \vdash N \rightsquigarrow \tau_n; \psi_n; \phi_n \\ \Gamma_P \vdash P \rightsquigarrow \psi_p; \phi_p\end{array}}{\begin{array}{l}\Gamma \vdash \overline{M}\, N.P \rightsquigarrow \psi; \phi_m \wedge \phi_n \wedge \phi_p \\ \quad \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftharpoonup^{\rho^+} \tau_n]\!] \\ \quad \bigwedge_{\substack{\Gamma(x)=\tau_1 \\ \Gamma_M(x)=\tau_2 \\ \Gamma_N(x)=\tau_3 \\ \Gamma_P(x)=\tau_4}} [\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!] \\ \quad \wedge [\![\mathsf{if}\ \mathcal{C}(\overline{M}\, N.P, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!] \\ \quad \wedge [\![\leftharpoonup^{\rho^+} (\psi, \psi_p, \psi_n, \psi_m)]\!]\end{array}} \quad \text{where} \quad \begin{array}{l}\Gamma_M = Ь(\Gamma) \\ \Gamma_N = Ь(\Gamma) \\ \Gamma_P = Ь(\Gamma)\end{array}$$

TABLE 4.4.3. New constraint generation rules for linearly typed ψ-calculi

Finally we present our typing rules for terms in Table 4.4.2. They resemble the ones seen in [13] a great deal.

We can now put forth Theorem 4.4.4 and prove it in Appendix B.2.2.

Theorem 4.4.4 (Termination)**.** *If there exists a* Γ *and* Ψ *such that* $\Gamma, \Psi \vdash P$ *then* P *terminates*

4.4.1. Constraint encoding. Before we can present the encoding of the constraints generated by the constraint generation rules of Table 4.2.1, we first need to define the constraint generation rules for terms and the new input and output rules. The constraint generation rules for input and output are identical to the ones defined in Table 4.2.1 with the exception of the introduction of the new side condition $\leftharpoonup^{\rho}(\Psi, \Psi_1, \Psi_2, \Psi_3)$. The constraint generation rules for terms are equally straight forward. We present the new constraint generation rules for input and output, and the constraint generation rules for terms in Tables 4.4.3 and 4.4.4 respectively.

$$[\text{C-Var}]\quad \frac{\Gamma(x) = \tau}{\begin{array}{l}\Gamma \vdash x \rightsquigarrow \tau; \psi; \mathsf{Ass}(\psi,\rho,\eta,\mu) \wedge \mathsf{Lin}(\rho) \\ \quad \wedge(\mathsf{Chan}(\tau) \Rightarrow (\mathsf{AssChan}(\psi) \wedge \mathsf{Eq}(\tau,\eta))) \\ \quad \wedge(\mathsf{AssChan}(\psi) \Rightarrow \mathsf{Chan}(\tau))\end{array}}$$

$$[\text{C-Com}]\quad \frac{\Gamma \vdash M_i \rightsquigarrow \tau_i; \psi_i; \phi_i \qquad 1 \leq i \leq |\vec{M}|}{\begin{array}{l}\Gamma \vdash f(\vec{M}) \rightsquigarrow \tau; \psi; \bigwedge_i \phi_i \\ \quad \wedge \mathsf{Ass}(\psi,\rho,\eta,\mu) \wedge \mathsf{Eq}(\eta,\tau) \\ \quad \wedge \bigwedge_i(\forall\rho_i : \forall\eta_i : \forall\mu_i : (\mathsf{Ass}(\psi_i,\rho_i,\eta_i,\mu_i) \Rightarrow \mathsf{LessEq}(\eta_i,\eta)))\end{array}}$$

TABLE 4.4.4. Constraint generation rules for terms

$\mathsf{Ass}(\psi,\rho,\eta,\mu)$	$\mathsf{AssChan}(\psi)$	$\mathsf{Rep}(\rho)$	$\mathsf{Lin}(\rho)$
$\mathsf{Chan}(\tau)$	$\mathsf{Val}(\tau)$	$\mathsf{Less}(a,b)$	$\mathsf{LessEq}(a,b)$
Eq			

TABLE 4.4.5. Constraint language for termination

We now proceed to define the constraint language we will utilise as seen in Table 4.4.5, and explain what the relations will denote. For assertions, as we in linearly typed ψ-calculi have to encode those, we make use of the following constraints: $\mathsf{Ass}(\psi,\rho,\eta,\mu)$ and $\mathsf{AssChan}(\psi)$. The relation $\mathsf{Ass}(\psi,\rho,\eta,\mu)$ denotes that the assertion variable ψ represents the assertion $\rho[\eta,\mu]$. The relation $\mathsf{AssChan}(\psi)$ denotes that the corresponding $\mathsf{Ass}(\psi,\rho,\eta,\mu)$-relation is to be read as $\mathsf{chan}(\eta)$ instead of $\rho[\eta,\mu]$. For the multiplicity variables ρ we utilise the relations $\mathsf{Rep}(\rho)$ and $\mathsf{Lin}(\rho)$ to denote whether ρ is $*$ or nothing. We once again let τ denote type variables and for the types we make use of the two relations $\mathsf{Chan}(\tau)$ and $\mathsf{Val}(\tau)$ to denote either τ is of the form Ch^n or n for some $n \in \mathbb{N}$. For the variables that denote numbers i.e. τ, η, and μ we utilise the relations $\mathsf{Less}(a,b)$ and $\mathsf{LessEq}(a,b)$ to denote "$a < b$" and "$a \leq b$" respectively. Lastly, as previously we let $\mathsf{Eq}(a,b)$ denote that two variables are equal, and we will include axioms to ensure that the relation Eq will be an equivalence relation and that it carries information correctly e.g. $\mathsf{Val}(\tau_1) \wedge \mathsf{Eq}(\tau_1,\tau_2) \Rightarrow \mathsf{Val}(\tau_2)$ and so forth.

In Table 4.4.6 we present the encoding of the constraints generated by the constraint generation rules for processes as presented in Tables 4.2.1 and 4.4.3.

These encodings and axioms should not be surprising, when one considers the meaning of the predicates. One side effect of these axioms, which may seem a bit odd is that because an $\mathsf{Eq}(\tau,\eta)$ predicate is generated in [C-Var] and [C-Com], the axiom $\forall\tau_1 : \forall\tau_2 : ((\mathsf{Chan}(\tau_1) \wedge \mathsf{Eq}(\tau_1,\tau_2)) \Rightarrow \mathsf{Chan}(\tau_2))$ creates a $\mathsf{Chan}(\eta)$ predicate, even though it obviously makes no sense to have a $\rho[\mathsf{Ch}^\eta,\mu]$ assertion. But we can simply define $\rhd$ in a way that does not allow such a predicate, as seen in Table 4.4.8. Since ρ does not appear in any of the criteria from Section 4.1.1 it does not matter whether $\mathscr{L}(\phi) \rhd \rho = \mathsf{Ch}^n$. Only the reconstruction of the type- and assertion variables matters, and these will obviously not be influenced by $\mathsf{Chan}(\rho)$. The type reconstruction rules are otherwise as expected, with $\overset{level}{\rhd}$ using Less and LessEq to find the minimum possible value for levels of types and in assertions.

$[\![\leftarrow\!\rho^- (\psi, \psi_1, \psi_2, \psi_3)]\!] =$
$\mathsf{Ass}(\psi, \rho, \eta, \mu) \wedge \mathsf{Ass}(\psi_1, \rho_1, \eta_1, \mu_1) \wedge \mathsf{Ass}(\psi_2, \rho_2, \eta_2, \mu_2) \wedge \mathsf{Ass}(\psi_3, \rho_3, \eta_3, \mu_3) \wedge$
$(\mathsf{Rep}(\rho) \Rightarrow (\mathsf{Rep}(\rho_3) \wedge \mathsf{Less}(\mu_1, \eta_3))) \wedge (\mathsf{Lin}(\rho) \Rightarrow \mathsf{Lin}(\rho_3)) \wedge \mathsf{Lin}(\rho_2) \wedge$
$\mathsf{AssChan}(\psi_3)$

$[\![\leftarrow\!\rho^+ (\psi, \psi_1, \psi_2, \psi_3)]\!] =$
$\mathsf{Ass}(\psi, \rho, \eta, \mu) \wedge \mathsf{Ass}(\psi_1, \rho_1, \eta_1, \mu_1) \wedge \mathsf{Ass}(\psi_2, \rho_2, \eta_2, \mu_2) \wedge \mathsf{Ass}(\psi_3, \rho_3, \eta_3, \mu_3) \wedge$
$\mathsf{Lin}(\rho) \wedge \mathsf{Lin}(\rho_2) \wedge \mathsf{Lin}(\rho_3) \wedge \mathsf{AssChan}(\psi_3)$

$[\![\tau_1 \leftarrow\!\rho^- \tau_2]\!] =$
$\mathsf{Chan}(\tau_1) \wedge \mathsf{Val}(\tau_2) \wedge \mathsf{LessEq}(\tau_2, \tau_1)$

$[\![\tau_1 \leftarrow\!\rho^+ \tau_2]\!] =$
$\mathsf{Chan}(\tau_1) \wedge \mathsf{Val}(\tau_2) \wedge \mathsf{LessEq}(\tau_2, \tau_1)$

$[\![\tau_1 = \tau_2 + \tau_3]\!] =$
$\mathsf{Eq}(\tau_1, \tau_2) \wedge \mathsf{Eq}(\tau_2, \tau_3)$

$[\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!] =$
$\mathsf{Eq}(\tau_1, \tau_2) \wedge \mathsf{Eq}(\tau_2, \tau_3) \wedge \mathsf{Eq}(\tau_3, \tau_4)$

$[\![\psi_1 \leq \psi_2]\!] =$
$\mathsf{Eq}(\psi_1, \psi_2)$

$[\![\psi_1 = \psi_2 \div x]\!] =$
$\mathsf{Eq}(\psi_1, \psi_2)$

$[\![\mathsf{if}\ \mathcal{C}(P, \psi_2)\ \mathsf{then}\ \mathsf{Weak}(\psi_1, \psi_2)\ \mathsf{else}\ \ldots]\!] =$
$\mathsf{Eq}(\psi_1, \psi_2)$

$[\![\psi_1 = \psi_2 \otimes \psi_3]\!] =$
$\mathsf{Ass}(\psi_1, \rho_1, \eta_1, \mu_1) \wedge \mathsf{Ass}(\psi_2, \rho_2, \eta_2, \mu_2) \wedge \mathsf{Ass}(\psi_3, \rho_3, \eta_3, \mu_3) \wedge$
$\mathsf{LessEq}(\mu_2, \mu_1) \wedge \mathsf{LessEq}(\mu_3, \mu_1) \wedge \mathsf{LessEq}(\eta_2, \mu_2) \wedge$
$\mathsf{LessEq}(\eta_3, \mu_3) \wedge ((\mathsf{Eq}(\rho_2, \rho_3) \wedge \mathsf{Eq}(\eta_2, \eta_3) \wedge \mathsf{Eq}(\mu_2, \mu_3)) \Rightarrow \mathsf{Rep}(\rho_1))$
$(\mathsf{AssChan}(\psi_1) \Rightarrow \mathsf{Fail}(\psi_1))$

$[\![\psi_1 = \psi_2 \otimes \psi_3 \otimes \psi_4]\!] =$
$\mathsf{Ass}(\psi_1, \rho_1, \eta_1, \mu_1) \wedge \mathsf{Ass}(\psi_2, \rho_2, \eta_2, \mu_2) \wedge \mathsf{Ass}(\psi_3, \rho_3, \eta_3, \mu_3) \wedge \mathsf{Ass}(\psi_4, \rho_4, \eta_4, \mu_4) \wedge$
$\mathsf{LessEq}(\mu_2, \mu_1) \wedge \mathsf{LessEq}(\mu_3, \mu_1) \wedge \mathsf{LessEq}(\mu_4, \mu_1)$
$\mathsf{LessEq}(\eta_2, \mu_2) \wedge \mathsf{LessEq}(\eta_3, \mu_3) \wedge \mathsf{LessEq}(\eta_4, \mu_4)$
$(\mathsf{AssChan}(\psi_1) \Rightarrow \mathsf{Fail}(\psi_1))$

TABLE 4.4.6. Constraint encoding

$\forall a : \mathsf{Eq}(a, a)$
$\forall a : \forall b : \mathsf{Eq}(a, b) \Rightarrow \mathsf{Eq}(b, a)$
$\forall a : \forall b : \forall c : (\mathsf{Eq}(a, b) \wedge \mathsf{Eq}(b, c)) \Rightarrow \mathsf{Eq}(a, c)$

$\forall \psi : \forall \rho_1 : \forall \rho_2 : \forall \eta_1 : \forall \eta_2 : \forall \mu_2 : \forall \mu_3 :$
$((\mathsf{Ass}(\psi, \rho_1, \eta_1, \mu_1) \wedge \mathsf{Ass}(\psi, \rho_2, \eta_2, \mu_2)) \Rightarrow$
$(\mathsf{Eq}(\rho_1, \rho_2) \wedge \mathsf{Eq}(\eta_1, \eta_2) \wedge \mathsf{Eq}(\mu_1, \mu_2)))$

$\forall \psi_1 : \forall \psi_2 : (\mathsf{Eq}(\psi_1, \psi_2) \Rightarrow$
$(\forall \rho_1 : \forall \rho_2 : \forall \eta_1 : \forall \eta_2 : \forall \mu_2 : \forall \mu_3 :$
$(\mathsf{Ass}(\psi_1, \rho_1, \eta_1, \mu_1) \wedge \mathsf{Ass}(\psi_2, \rho_2, \eta_2, \mu_2)) \Rightarrow$
$(\mathsf{Eq}(\rho_1, \rho_2) \wedge \mathsf{Eq}(\eta_1, \eta_2) \wedge \mathsf{Eq}(\mu_1, \mu_2))))$

$\forall \psi_1 : \forall \psi_2 : ((\mathsf{AssChan}(\psi_1) \wedge \mathsf{Eq}(\psi_1, \psi_2)) \Rightarrow \mathsf{AssChan}(\psi_2))$
$\forall \rho_1 : \forall \rho_2 : ((\mathsf{Lin}(\rho_1) \wedge \mathsf{Eq}(\rho_1, \rho_2)) \Rightarrow \mathsf{Lin}(\rho_2))$
$\forall \rho_1 : \forall \rho_2 : ((\mathsf{Rep}(\rho_1) \wedge \mathsf{Eq}(\rho_1, \rho_2)) \Rightarrow \mathsf{Rep}(\rho_2))$
$\forall \tau_1 : \forall \tau_2 : ((\mathsf{Chan}(\tau_1) \wedge \mathsf{Eq}(\tau_1, \tau_2)) \Rightarrow \mathsf{Chan}(\tau_2))$
$\forall \tau_1 : \forall \tau_2 : ((\mathsf{Val}(\tau_1) \wedge \mathsf{Eq}(\tau_1, \tau_2)) \Rightarrow \mathsf{Val}(\tau_2))$
$\forall \rho_1 : \forall \rho_2 : ((\mathsf{Rep}(\rho_1) \wedge \mathsf{Rep}(\rho_2)) \Rightarrow \mathsf{Eq}(\rho_1, \rho_2))$
$\forall \rho_1 : \forall \rho_2 : ((\mathsf{Lin}(\rho_1) \wedge \mathsf{Lin}(\rho_2)) \Rightarrow \mathsf{Eq}(\rho_1, \rho_2))$

$\forall a : \forall b : \forall c : ((\mathsf{Less}(a, b) \wedge \mathsf{Eq}(a, c)) \Rightarrow \mathsf{Less}(c, b))$
$\forall a : \forall b : \forall c : ((\mathsf{Less}(a, b) \wedge \mathsf{Eq}(b, c)) \Rightarrow \mathsf{Less}(a, c))$
$\forall a : \forall b : \forall c : ((\mathsf{Less}(a, b) \wedge \mathsf{Less}(b, c)) \Rightarrow \mathsf{Less}(a, c))$
$\forall a : \forall b : \forall c : ((\mathsf{Less}(a, b) \wedge \mathsf{LessEq}(b, c)) \Rightarrow \mathsf{Less}(a, c))$
$\forall a : \forall b : \forall c : ((\mathsf{Less}(a, b) \wedge \mathsf{LessEq}(b, a)) \Rightarrow \mathsf{Fail}(a)$

$\forall a : \forall b : \forall c : ((\mathsf{LessEq}(a, b) \wedge \mathsf{LessEq}(b, c)) \Rightarrow \mathsf{LessEq}(a, c))$
$\forall a : \forall b : \forall c : ((\mathsf{LessEq}(a, b) \wedge \mathsf{Eq}(b, c)) \Rightarrow \mathsf{LessEq}(a, c))$
$\forall a : \forall b : \forall c : ((\mathsf{LessEq}(a, b) \wedge \mathsf{Eq}(a, c)) \Rightarrow \mathsf{LessEq}(c, b))$
$\forall a : \forall b : ((\mathsf{LessEq}(a, b) \wedge \mathsf{LessEq}(b, a)) \Rightarrow \mathsf{Eq}(a, b))$

$\forall a : \forall b : ((\mathsf{Less}(a, b) \wedge \mathsf{Less}(b, a)) \Rightarrow \mathsf{Fail}(a, b))$
$\forall \rho : ((\mathsf{Rep}(\rho) \wedge \mathsf{Lin}(\rho)) \Rightarrow \mathsf{Fail}(\rho))$
$\forall \tau : ((\mathsf{Chan}(\tau) \wedge \mathsf{Val}(\tau)) \Rightarrow \mathsf{Fail}(\tau)$

TABLE 4.4.7. Axioms

$$\frac{\Phi \overset{level}{\rhd} \tau = n \qquad \mathsf{Val}(\tau) \in \Phi}{\Phi \rhd \tau = n}$$

$$\frac{\Phi \overset{level}{\rhd} \tau = n \qquad \mathsf{Chan}(\tau) \in \Phi}{\Phi \rhd \tau = \mathsf{Ch}^n}$$

$$\frac{\Phi \overset{level}{\rhd} \mu = n \qquad \mathsf{Ass}(\psi, \rho, \eta, \mu) \in \Phi \qquad \mathsf{AssChan}(\psi) \in \Phi}{\Phi \rhd \psi = \mathsf{chan}(n)}$$

$$\frac{\Phi \overset{level}{\rhd} \eta \leq n \leq m \qquad \Phi \overset{level}{\rhd} \mu = m \qquad \mathsf{Ass}(\psi, \rho, \eta, \mu) \in \Phi \qquad \mathsf{Lin}(\rho) \in \Phi}{\Phi \rhd \psi = [n, m]}$$

$$\frac{\Phi \overset{level}{\rhd} \eta \leq n \leq m \qquad \Phi \overset{level}{\rhd} \mu = m \qquad \mathsf{Ass}(\psi, \rho, \eta, \mu) \in \Phi \qquad \mathsf{Rep}(\rho) \in \Phi}{\Phi \rhd \psi = *[n, m]}$$

$$\frac{\nexists b : \mathsf{Less}(b, a) \in \Phi \text{ or } \mathsf{LessEq}(b, a) \in \Phi \text{ where } \mathsf{Eq}(b, a) \notin \Phi}{\Phi \overset{level}{\rhd} a = 0}$$

$$\frac{\begin{array}{l}\exists b : \mathsf{Less}(b, a) \in \Phi \text{ or } \mathsf{LessEq}(b, a) \in \Phi \\ n = \max\left(\left\{ m \;\middle|\; \begin{array}{c} \exists b : \mathsf{LessEq}(b, a) \in \Phi \text{ and } m = \Phi \overset{level}{\rhd} b \text{ or} \\ \mathsf{Less}(b, a) \in \Phi \text{ and } m = 1 + \Phi \overset{level}{\rhd} b \end{array} \right\}\right)\end{array}}{\Phi \overset{level}{\rhd} a = n}$$

TABLE 4.4.8. Type and assertions reconstruction function

4.4.2. Example. We now show an example of type inference on the simple process

$$P = *\underline{a}(\lambda\vec{x})x..\overline{b}\,x. \mid \overline{a}\,c..$$

In order to generate constraints, we provide the environment $\Gamma = a : \tau_1, b : \tau_2, c : \tau_3$. We use the constraint generating rules to get a conjunction of the following constraints:

$\mathsf{Ass}(\psi_1, \rho_{18}, \eta_{18}, \mu_{18})$ $\mathsf{Ass}(\psi_{11}, \rho_{19}, \eta_{19}, \mu_{19})$ $\mathsf{Ass}(\psi_{12}, \rho_{20}, \eta_{20}, \mu_{20})$
$\mathsf{LessEq}(\mu_{19}, \mu_{18})$ $\mathsf{LessEq}(\mu_{20}, \mu_{18})$ $\mathsf{LessEq}(\eta_{19}, \mu_{19})$
$\mathsf{LessEq}(\eta_{20}, \mu_{20})$
$(\mathsf{Eq}(\rho_{19}, \rho_{20}) \wedge \mathsf{Eq}(\eta_{19}, \eta_{20}) \wedge \mathsf{Eq}(\mu_{19}, \mu_{20})) \Rightarrow \mathsf{Rep}(\rho_{18})$
$\mathsf{AssChan}(\psi_1) \Rightarrow \mathsf{Fail}(\psi_1)$ $\mathsf{Ass}(\psi_{23}, \rho_{21}, \eta_{21}, \mu_{21})$ $\mathsf{Ass}(\psi_{13}, \rho_{22}, \eta_{22}, \mu_{22})$
$\mathsf{Ass}(\psi_{14}, \rho_{23}, \eta_{23}, \mu_{23})$ $\mathsf{LessEq}(\mu_{22}, \mu_{21})$ $\mathsf{LessEq}(\mu_{23}, \mu_{21})$
$\mathsf{LessEq}(\eta_{22}, \mu_{22})$ $\mathsf{LessEq}(\eta_{23}, \mu_{23})$
$(\mathsf{Eq}(\rho_{22}, \rho_{23}) \wedge \mathsf{Eq}(\eta_{22}, \eta_{23}) \wedge \mathsf{Eq}(\mu_{22}, \mu_{23})) \Rightarrow \mathsf{Rep}(\rho_{21})$
$\mathsf{AssChan}(\psi_{23}) \Rightarrow \mathsf{Fail}(\psi_{23})$ $\mathsf{Ass}(\psi_{17}, \rho_{24}, \eta_{24}, \mu_{24})$ $\mathsf{Ass}(\psi_{11}, \rho_{25}, \eta_{25}, \mu_{25})$
$\mathsf{Ass}(\psi_{13}, \rho_{26}, \eta_{26}, \mu_{26})$ $\mathsf{LessEq}(\mu_{25}, \mu_{24})$ $\mathsf{LessEq}(\mu_{26}, \mu_{24})$
$\mathsf{LessEq}(\eta_{25}, \mu_{25})$ $\mathsf{LessEq}(\eta_{26}, \mu_{26})$
$(\mathsf{Eq}(\rho_{25}, \rho_{26}) \wedge \mathsf{Eq}(\eta_{25}, \eta_{26}) \wedge \mathsf{Eq}(\mu_{25}, \mu_{26})) \Rightarrow \mathsf{Rep}(\rho_{24})$
$\mathsf{AssChan}(\psi_{17}) \Rightarrow \mathsf{Fail}(\psi_{17})$ $\mathsf{Eq}(\psi_{17}, \psi_1)$ $\mathsf{Eq}(\psi_{12}, \psi_{15})$
$\mathsf{Eq}(\psi_{14}, \psi_{16})$ $\mathsf{Eq}(\tau_1, \tau_{1_1})$ $\mathsf{Eq}(\tau_{1_1}, \tau_{1_2})$
$\mathsf{Eq}(\tau_2, \tau_{2_1})$ $\mathsf{Eq}(\tau_{2_1}, \tau_{2_2})$ $\mathsf{Eq}(\tau_3, \tau_{3_1})$
$\mathsf{Eq}(\tau_{3_1}, \tau_{3_2})$ $\mathsf{Ass}(\psi_2, \rho_{27}, \eta_{27}, \mu_{27})$ $\mathsf{Ass}(\psi_2, \rho_{28}, \eta_{28}, \mu_{28})$
$\mathsf{Ass}(\psi_2, \rho_{29}, \eta_{29}, \mu_{29})$ $\mathsf{LessEq}(\mu_{28}, \mu_{27})$ $\mathsf{LessEq}(\mu_{29}, \mu_{27})$
$\mathsf{LessEq}(\eta_{28}, \mu_{28})$ $\mathsf{LessEq}(\eta_{29}, \mu_{29})$
$(\mathsf{Eq}(\rho_{28}, \rho_{29}) \wedge \mathsf{Eq}(\eta_{28}, \eta_{29}) \wedge \mathsf{Eq}(\mu_{28}, \mu_{29})) \Rightarrow \mathsf{Rep}(\rho_{27})$
$\mathsf{AssChan}(\psi_2) \Rightarrow \mathsf{Fail}(\psi_2)$ $\mathsf{Eq}(\psi_1, \psi_2)$ $\mathsf{Ass}(\psi_3, \rho_3, \eta_3, \mu_3)$
$\mathsf{Val}(\tau_{1_{11}}) \Rightarrow \mathsf{Ass}(\psi_3, \rho_1, \eta_1, \mu_1)$ $\mathsf{Lin}(\rho_3)$
$\mathsf{Chan}(\tau_{1_{11}}) \Rightarrow (\mathsf{Ass}(\psi_3, \rho_2, \eta_2, \mu_2) \wedge \mathsf{AssChan}(\psi_3) \wedge \mathsf{Eq}(\tau_{1_{11}}, \eta_2))$
$\mathsf{AssChan}(\psi_3) \Rightarrow \mathsf{Chan}(\tau_{1_{11}})$ $\mathsf{Ass}(\psi_4, \rho_4, \eta_4, \mu_4)$ $\mathsf{Lin}(\rho_4)$
$\mathsf{Val}(\tau_4) \Rightarrow \mathsf{Ass}(\psi_4, \rho_5, \eta_5, \mu_5)$
$\mathsf{Chan}(\tau_4) \Rightarrow (\mathsf{Ass}(\psi_4, \rho_6, \eta_6, \mu_6) \wedge \mathsf{AssChan}(\psi_4) \wedge \mathsf{Eq}(\tau_4, \eta_6))$
$\mathsf{AssChan}(\psi_4) \Rightarrow \mathsf{Chan}(\tau_4)$ $\mathsf{Ass}(\psi_5, \rho_{30}, \eta_{30}, \mu_{30})$ $\mathsf{Ass}(\psi_3, \rho_{31}, \eta_{31}, \mu_{31})$
$\mathsf{Ass}(\psi_4, \rho_{32}, \eta_{32}, \mu_{32})$ $\mathsf{Ass}(\psi_6, \rho_{33}, \eta_{33}, \mu_{33})$ $\mathsf{LessEq}(\mu_{31}, \mu_{30})$
$\mathsf{LessEq}(\mu_{32}, \mu_{30})$ $\mathsf{LessEq}(\mu_{33}, \mu_{30})$ $\mathsf{AssChan}(\psi_5) \Rightarrow \mathsf{Fail}(\psi_5)$
$\mathsf{Chan}(\tau_{1_{11}})$ $\mathsf{Val}(\tau_4)$ $\mathsf{LessEq}(\tau_4, \tau_{1_{11}})$
$\mathsf{Eq}(\tau_{1_1}, \tau_{1_{11}})$ $\mathsf{Eq}(\tau_{1_{11}}, \tau_{1_{12}})$ $\mathsf{Eq}(\tau_{1_{12}}, \tau_{1_{13}})$
$\mathsf{Eq}(\tau_{2_1}, \tau_{2_{11}})$ $\mathsf{Eq}(\tau_{2_{11}}, \tau_{2_{12}})$ $\mathsf{Eq}(\tau_{2_{12}}, \tau_{2_{13}})$
$\mathsf{Eq}(\tau_{3_1}, \tau_{3_{11}})$ $\mathsf{Eq}(\tau_{3_{11}}, \tau_{3_{12}})$ $\mathsf{Eq}(\tau_{3_{12}}, \tau_{3_{13}})$
$\mathsf{Eq}(\psi_5, \psi_2)$ $\mathsf{Ass}(\psi_2, \rho_{42}, \eta_{42}, \mu_{42})$ $\mathsf{Ass}(\psi_6, \rho_{43}, \eta_{43}, \mu_{43})$
$\mathsf{Ass}(\psi_4, \rho_{44}, \eta_{44}, \mu_{44})$ $\mathsf{Ass}(\psi_3, \rho_{45}, \eta_{45}, \mu_{45})$ $\mathsf{Rep}(\rho_{42}) \Rightarrow \mathsf{Less}(\mu_{43}, \eta_{45})$
$\mathsf{Lin}(\rho_{42}) \Rightarrow \mathsf{Lin}(\rho_{45})$ $\mathsf{AssChan}(\psi_3)$ $\mathsf{Ass}(\psi_7, \rho_7, \eta_7, \mu_7)$
$\mathsf{Val}(\tau_{2_{131}}) \Rightarrow \mathsf{Ass}(\psi_7, \rho_8, \eta_8, \mu_8)$ $\mathsf{Lin}(\rho_7)$
$\mathsf{Chan}(\tau_{2_{131}}) \Rightarrow (\mathsf{Ass}(\psi_7, \rho_9, \eta_9, \mu_9) \wedge \mathsf{AssChan}(\psi_7) \wedge \mathsf{Eq}(\tau_{2_{131}}, \eta_9))$
$\mathsf{AssChan}(\psi_7) \Rightarrow \mathsf{Chan}(\tau_{2_{131}})$ $\mathsf{Ass}(\psi_8, \rho_{10}, \eta_{10}, \mu_{10})$ $\mathsf{Lin}(\rho_{10})$
$\mathsf{Val}(\tau_{4_{132}}) \Rightarrow \mathsf{Ass}(\psi_8, \rho_{11}, \eta_{11}, \mu_{11})$
$\mathsf{Chan}(\tau_{4_{132}}) \Rightarrow (\mathsf{Ass}(\psi_8, \rho_{12}, \eta_{12}, \mu_{12}) \wedge \mathsf{AssChan}(\psi_8) \wedge \mathsf{Eq}(\tau_{4_{132}}, \eta_{12})))$
$\mathsf{AssChan}(\psi_8) \Rightarrow \mathsf{Chan}(\tau_{4_{132}})$ $\mathsf{Ass}(\psi_{10}, \rho_{34}, \eta_{34}, \mu_{34})$ $\mathsf{Ass}(\psi_7, \rho_{35}, \eta_{35}, \mu_{35})$
$\mathsf{Ass}(\psi_8, \rho_{36}, \eta_{36}, \mu_{36})$ $\mathsf{Ass}(\psi_9, \rho_{37}, \eta_{37}, \mu_{37})$ $\mathsf{LessEq}(\mu_{35}, \mu_{34})$
$\mathsf{LessEq}(\mu_{36}, \mu_{34})$ $\mathsf{LessEq}(\mu_{37}, \mu_{34})$ $\mathsf{AssChan}(\psi_{10}) \Rightarrow \mathsf{Fail}(\psi_{10})$
$\mathsf{Chan}(\tau_{2_{131}})$ $\mathsf{Val}(\tau_{4_{132}})$ $\mathsf{LessEq}(\tau_{4_{132}}, \tau_{2_{131}})$
$\mathsf{Eq}(\tau_{1_{13}}, \tau_{1_{131}})$ $\mathsf{Eq}(\tau_{1_{131}}, \tau_{1_{132}})$ $\mathsf{Eq}(\tau_{1_{132}}, \tau_{1_{133}})$
$\mathsf{Eq}(\tau_{2_{13}}, \tau_{2_{131}})$ $\mathsf{Eq}(\tau_{2_{131}}, \tau_{2_{132}})$ $\mathsf{Eq}(\tau_{2_{132}}, \tau_{2_{133}})$
$\mathsf{Eq}(\tau_{3_{13}}, \tau_{3_{131}})$ $\mathsf{Eq}(\tau_{3_{131}}, \tau_{3_{132}})$ $\mathsf{Eq}(\tau_{3_{132}}, \tau_{3_{133}})$
$\mathsf{Eq}(\tau_4, \tau_{4_{131}})$ $\mathsf{Eq}(\tau_{4_{131}}, \tau_{4_{132}})$ $\mathsf{Eq}(\tau_{4_{132}}, \tau_{4_{133}})$

$\mathsf{Eq}(\psi_6, \psi_{10})$ $\mathsf{Ass}(\psi_6, \rho_{46}, \eta_{46}, \mu_{46})$ $\mathsf{Ass}(\psi_9, \rho_{47}, \eta_{47}, \mu_{47})$
$\mathsf{Ass}(\psi_8, \rho_{48}, \eta_{48}, \mu_{48})$ $\mathsf{Ass}(\psi_7, \rho_{49}, \eta_{49}, \mu_{49})$ $\mathsf{Lin}(\rho_{46})$
$\mathsf{Lin}(\rho_{48})$ $\mathsf{Lin}(\rho_{49})$ $\mathsf{AssChan}(\psi_7)$
$\mathsf{Val}(\tau_{1_{211}}) \Rightarrow \mathsf{Ass}(\psi_{19}, \rho_{14}, \eta_{14}, \mu_{14})$ $\mathsf{Ass}(\psi_{19}, \rho_{13}, \eta_{13}, \mu_{13})$ $\mathsf{Lin}(\rho_{13})$
$\mathsf{Chan}(\tau_{1_{211}}) \Rightarrow (\mathsf{Ass}(\psi_{19}, \rho_{15}, \eta_{15}, \mu_{15}) \wedge \mathsf{AssChan}(\psi_{19}) \wedge \mathsf{Eq}(\tau_{1_{211}}, \eta_{15}))$
$\mathsf{AssChan}(\psi_{19}) \Rightarrow \mathsf{Chan}(\tau_{1_{211}})$ $\mathsf{Ass}(\psi_{20}, \rho_{16}, \eta_{16}, \mu_{16})$ $\mathsf{Lin}(\rho_{16})$
$\mathsf{Val}(\tau_{3_{212}}) \Rightarrow \mathsf{Ass}(\psi_{20}, \rho_{17}, \eta_{17}, \mu_{17})$
$\mathsf{Chan}(\tau_{3_{212}}) \Rightarrow (\mathsf{Ass}(\psi_{20}, \rho_{18}, \eta_{18}, \mu_{18}) \wedge \mathsf{AssChan}(\psi_{20}) \wedge \mathsf{Eq}(\tau_{3_{212}}, \eta_{18}))$
$(\mathsf{AssChan}(\psi_{20}) \Rightarrow \mathsf{Chan}(\tau_{3_{212}}))$ $\mathsf{Ass}(\psi_{22}, \rho_{38}, \eta_{38}, \mu_{38})$ $\mathsf{Ass}(\psi_{19}, \rho_{39}, \eta_{39}, \mu_{39})$
$\mathsf{Ass}(\psi_{20}, \rho_{40}, \eta_{40}, \mu_{40})$ $\mathsf{Ass}(\psi_{21}, \rho_{41}, \eta_{41}, \mu_{41})$ $\mathsf{LessEq}(\mu_{39}, \mu_{38})$
$\mathsf{LessEq}(\mu_{40}, \mu_{38})$ $\mathsf{LessEq}(\mu_{41}, \mu_{38})$ $\mathsf{AssChan}(\psi_{22}) \Rightarrow \mathsf{Fail}(\psi_{22})$
$\mathsf{Chan}(\tau_{1_{211}})$ $\mathsf{Val}(\tau_{3_{212}})$ $\mathsf{LessEq}(\tau_{3_{212}}, \tau_{1_{211}})$
$\mathsf{Eq}(\tau_{1_{21}}, \tau_{1_{211}})$ $\mathsf{Eq}(\tau_{1_{211}}, \tau_{1_{212}})$ $\mathsf{Eq}(\tau_{1_{212}}, \tau_{1_{213}})$
$\mathsf{Eq}(\tau_{2_{21}}, \tau_{2_{211}})$ $\mathsf{Eq}(\tau_{2_{211}}, \tau_{2_{212}})$ $\mathsf{Eq}(\tau_{2_{212}}, \tau_{2_{213}})$
$\mathsf{Eq}(\tau_{3_{21}}, \tau_{3_{211}})$ $\mathsf{Eq}(\tau_{3_{211}}, \tau_{3_{212}})$ $\mathsf{Eq}(\tau_{3_{212}}, \tau_{3_{213}})$
$\mathsf{Eq}(\psi_{22}, \psi_{18})$ $\mathsf{Ass}(\psi_{18}, \rho_{50}, \eta_{50}, \mu_{50})$ $\mathsf{Ass}(\psi_{21}, \rho_{51}, \eta_{51}, \mu_{51})$
$\mathsf{Ass}(\psi_{20}, \rho_{52}, \eta_{52}, \mu_{52})$ $\mathsf{Ass}(\psi_{19}, \rho_{53}, \eta_{53}, \mu_{53})$ $\mathsf{Lin}(\rho_{50})$
$\mathsf{Lin}(\rho_{52})$ $\mathsf{Lin}(\rho_{53})$ $\mathsf{AssChan}(\psi_7)$
$\mathsf{Val}(\tau_4)$ $\mathsf{Eq}(\psi_{23}, \psi_{18})$ $\mathsf{Eq}(\tau_{1_2}, \tau_{1_{21}})$
$\mathsf{Eq}(\tau_{2_2}, \tau_{2_{21}})$ $\mathsf{Eq}(\tau_{3_2}, \tau_{3_{21}})$

Using the axioms found in Table 4.4.7 we construct the solution found in Appendix A.3. In the solution we let t_x_y denote τ_{x_y}, s_x denote ϕ_x, r_x denote ρ_x, n_x denote η_x, and m_x denote μ_x. We now use the solution found to reconstruct the types τ_1, τ_2, and τ_3 together with the initial assertion ψ_{17}. Due to a small error in the constraint generation, this assertion will be replicated, whereas it would under normal circumstances have been linear.

$a : \tau_1$**:** We want to compute the type of τ_1. We first investigate the solution Φ to determine whether tau_1 is a channel or a value. We find $\mathsf{Chan}(\tau_1)$. We now need to compute $\Phi \overset{level}{\rhd} \tau_1$. We now need to determine if there exists any variable with a value less than or equal to τ_1. We once again investiagte the solution and find: $\mathsf{Less}(\mu_{46}, \tau_1)$, $\mathsf{Less}(\mu_{33}, \tau_1)$, $\mathsf{Less}(\mu_{34}, \tau_1)$, $\mathsf{Less}(\mu_{43}, \tau_1)$, $\mathsf{LessEq}(\tau_{4_{131}}, \tau_1)$, $\mathsf{LessEq}(\tau_{4_{133}}, \tau_1)$, $\mathsf{LessEq}(\tau_3, \tau_1)$, $\mathsf{LessEq}(\tau_{3_{11}}, \tau_1)$, $\mathsf{LessEq}(\tau_{3_{213}}, \tau_1)$, $\mathsf{LessEq}(\tau_{3_{12}}, \tau_1)$, $\mathsf{LessEq}(\tau_{3_{211}}, \tau_1)$, $\mathsf{LessEq}(\tau_{3_{13}}, \tau_1)$, $\mathsf{LessEq}(\tau_{3_2}, \tau_1)$, $\mathsf{LessEq}(\tau_{3_{131}}, \tau_1)$, $\mathsf{LessEq}(\tau_{3_{21}}, \tau_1)$, $\mathsf{LessEq}(\tau_{3_{132}}, \tau_1)$, $\mathsf{LessEq}(\tau_{3_1}, \tau_1)$, $\mathsf{LessEq}(\tau_{3_{133}}, \tau_1)$, $\mathsf{LessEq}(\tau_{3_{212}}, \tau_1)$, $\mathsf{LessEq}(\tau_{4_{132}}, \tau_1)$, and $\mathsf{LessEq}(\tau_4, \tau_1)$. We thus have to compute the value of all of these variables, and select the maximum of those. We proceed by computing $\Phi \overset{level}{\rhd} \mu_{46}$. We once again need to determine if there exists any values less than or equal to μ_{46}. As before we investigate the solution, but find that all values less than or equal to μ_{46} are not larger than any other values, and by $\overset{level}{\rhd}$ we have $\mu_{46} = \max(\{0\}) = 0$. We apply a similar approach for the remaining variables and get $\mu_{33} = 0$, $\mu_{34} = 0$, $\mu_{43} = 0$, $\tau_4^{131} = 0$, $\tau_{4_{133}} = 0$, $\tau_3 = 0$, $\tau_{3_{11}} = 0$, $\tau_{3_{213}} = 0$, $\tau_{3_{12}} = 0$, $\tau_{3_{211}} = 0$, $\tau_{3_{13}} = 0$, $\tau_{3_2} = 0$, $\tau_{3_{131}} = 0$, $\tau_{3_{21}} = 0$, $\tau_{3_{132}} = 0$, $\tau_{3_1} = 0$, $\tau_{3_{133}} = 0$, $\tau_{3_{212}} = 0$, $\tau_{4_{132}} = 0$, $\tau_4 = 0$. We thus have $\Phi \overset{level}{\rhd} \tau_1 = \max(\{1, 0\}) = 1$. We can finally conclude $\Phi \rhd \tau_1 = \mathsf{Ch}^1$.

$b : \tau_2$**:** This computation is similar to the one for $a : \tau_1$. For simplciity we immediately present the result, namely $\Phi \rhd \tau_2 = \mathsf{Ch}^0$.

$c : \tau_3$**:** During the computation for $a : \tau_1$ we derived that $\Phi \rhd \tau_3 = 0$.

ψ_{17}**:** To compute $\Phi \rhd \psi_{17}$ we need to find its corresponding Ass-relation. We invesitgate Φ and find $\mathsf{Ass}(\psi_{17}, \rho_{24}, \eta_{24}, \mu_{24})$. We now determine if the assertion is linear or replicated. We investigate Φ and find $\mathsf{Rep}(\rho_{24})$ and we conclude the assertion is replicated. We need to compute $\Phi \overset{level}{\rhd} \eta_{24}$ and $\Phi \overset{level}{\rhd} \mu_{24}$. By a

similar approach as for $a : \tau_1$ we get $\Phi \overset{level}{\rhd} \eta_{24} = 0$ and $\Phi \overset{level}{\rhd} \mu_{24} = 1$, implying $\Phi \rhd \psi_{17} = *[0, 1]$.

CHAPTER 5

Conclusion

In this report we succeeded in creating general type inference algorithms for both generic type systems considered, and also proved the algorithms to be correct. We used the approach of constraint generation where we used the ALFP as the constraint language, since this is a decidable fragment of first order logic. While successful, we had serious problems using the algorithms on more complicated instantiations of the type systems, as we discovered ALFP was not a suitable language for certain conditions. For instance, the dependent types caused problems because the ALFP does not allow the creation of new names/variables while solving, requiring any encoding of a type T' defined as $T' = T[^{a}/_{b}]$ to utilise only names used in the encoding of T. While this may in some cases be enough to infer whether a solution exists or not, it becomes increasingly problematic when one wants to use the solution to reconstruct the type T'. We managed a partial solution to this problem, as detailed in Section 3.4.2. It is possible that one can incorporate ideas such as the ones presented by Felty and Miller in [4] to correctly handle fully dependent types with no restrictions.

We also found that using ALFP when attempting to find the optimum solution rather than just an arbitrary solution was problematic. If we consider a type system like [1] in which we can always type a process—in this case by simply assign the type Pub to everything—the question is not simply whether there exists a typing or not, but instead what is the most strict typing we can use? Strictness in case of [1] would be proportional to how many secret types we have in our typing. This was problematic as ALFP is not able to compare e.g. the number of times names appears in a specific relation.

Lastly we also discovered that the stratification requirements from the ALFP become problematic when one have circular dependencies in ones constraints. Stratification becomes problematic as it states that the rank of any query must be less than or equal to the rank of the assertions, and the rank of negative queries must be less than the rank of the assertions. Thus it easily becomes impossible to formulate cyclic constraints involving negative queries, as we for instance cannot have $\neg P \Rightarrow Q$ and $\neg Q \Rightarrow P$. This implies that if we have for instance that a type is either linear or unlimited and we assume we have two predicates on type variables denoting this, we cannot construct constraints that ensures that either $\mathsf{Lin}(\tau)$ or $\mathsf{Un}(\tau)$ as these constraint will utilise a negative query on the counterpart (similar to the example above). The problem can also arise if we know $\neg P \Rightarrow Q$, but in another constraint we have $(\ldots \wedge Q \wedge \ldots) \Rightarrow (\ldots \wedge P \wedge \ldots)$. Clearly $\mathit{rank}(P) < \mathit{rank}(Q)$ by the first constraint, but $\mathit{rank}(Q) \leq \mathit{rank}(P)$ by the second constraint. Obviously no assignment of natural numbers to P and Q can satisfy this.

Results. We created a method of generating ALFP constraints for type inference for two generic type systems—simply typed ψ-calculi and linearly typed ψ-calculi—and proved that these methods worked. We showed examples of instantiations of the generic type systems and type inference, and proved that the instantiations of the type inference worked. We also looked at the attempt at creating a linearly typed ψ-calculus to mimic the type system by Deng and Sangiorgi [3] as presented by Hüttel in [13], but found several problems with this instantiation. We have detailed the problems and presented a possible solution to this in Section 4.4. We have not formally proved the correctness of you proposed solution due to time constraints.

Future work. As detailed above, we found it difficult to use ALFP to find solutions for a number of instantiations of the generic type systems, and it would therefore most likely be a sensible idea to consider alternative logics as constraint languages. In addition, since all the concepts used in the generic constraint generation for both type systems can be used for any language, it might be possible to replace $\wedge$ with a different symbol for conjunction, thereby allowing different logics to express constraints for different instantiations of the type systems depending on what is best suited. This might make it easier to deal with issues like dependent types or optimisation of solutions.

In addition we would want to formally prove that the constraints and axioms presented in Section 4.4.1 are in fact enough to correctly infer the types for the corresponding type system.

Bibliography

[1] Martín Abadi and Bruno Blanchet. "Secrecy types for asymmetric communication". In: *Theor. Comput. Sci.* 3.298 (2003), pp. 387–415. DOI: 10.1016/S0304-3975(02)00863-0. URL: http://dx.doi.org/10.1016/S0304-3975(02)00863-0 (cit. on p. 59).

[2] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. "Psi-calculi: Mobile Processes, Nominal Data, and Logic". In: *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*. IEEE Computer Society, 2009, pp. 39–48. ISBN: 9780769537467. DOI: 10.1109/LICS.2009.20. URL: http://dx.doi.org/10.1109/LICS.2009.20 (cit. on pp. 1, 9).

[3] Yuxin Deng and Davide Sangiorgi. "Ensuring termination by typability". In: *Inf. Comput.* 204.7 (2006), pp. 1045–1082. DOI: 10.1016/j.ic.2006.03.002. URL: http://dx.doi.org/10.1016/j.ic.2006.03.002 (cit. on pp. 49, 59, 95).

[4] Amy P. Felty and Dale Miller. "Encoding a Dependent-Type Lambda-Calculus in a Logic Programming Language". In: *10th International Conference on Automated Deduction, Kaiserslautern, FRG, July 24-27, 1990, Proceedings*. Ed. by Mark E. Stickel. Vol. 449. Lecture Notes in Computer Science. Springer, 1990, pp. 221–235. ISBN: 3540528857. DOI: 10.1007/3-540-52885-7_90. URL: http://dx.doi.org/10.1007/3-540-52885-7_90 (cit. on p. 59).

[5] Cormac Flanagan, Stephen N. Freund, and Marina Lifshin. "Type inference for atomicity". In: *Proceedings of TLDI'05: 2005 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation, Long Beach, CA, USA, January 10, 2005*. Ed. by J. Gregory Morrisett and Manuel Fähndrich. ACM, 2005, pp. 47–58. ISBN: 1581139993. DOI: 10.1145/1040294.1040299. URL: http://doi.acm.org/10.1145/1040294.1040299 (cit. on p. 2).

[6] Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. "A type discipline for authorization policies". In: *ACM Trans. Program. Lang. Syst.* 29.5 (2007). DOI: 10.1145/1275497.1275500. URL: http://doi.acm.org/10.1145/1275497.1275500 (cit. on pp. 1, 2, 17, 18, 20).

[7] Philippa Gardner and Lucian Wischik. "Explicit Fusions". In: *Mathematical Foundations of Computer Science 2000, 25th International Symposium, MFCS 2000, Bratislava, Slovakia, August 28 - September 1, 2000, Proceedings*. Ed. by Mogens Nielsen and Branislav Rovan. Vol. 1893. Lecture Notes in Computer Science. Springer, 2000, pp. 373–382. ISBN: 3540679014. DOI: 10.1007/3-540-44612-5_33. URL: http://dx.doi.org/10.1007/3-540-44612-5_33 (cit. on pp. 1, 14).

[8] Simon J. Gay. "A Sort Inference Algorithm for the Polyadic Pi-Calculus". In: *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Charleston, South Carolina, USA, January 1993*. Ed. by Mary S. Van Deusen and Bernard Lang. ACM Press, 1993, pp. 429–438. ISBN: 0897915607. DOI: 10.1145/158511.158701. URL: http://doi.acm.org/10.1145/158511.158701 (cit. on p. 9).

[9] Andrew D. Gordon, Hans Hüttel, and René Rydhof Hansen. "Type Inference for Correspondence Types". In: *Electr. Notes Theor. Comput. Sci.* 242.3 (2009), pp. 21–36. DOI: 10.1016/j.entcs.2009.07.079. URL: http://dx.doi.org/10.1016/j.entcs.2009.07.079 (cit. on pp. 2, 11, 17, 18, 20, 31).

[10] Andrew D. Gordon and Alan Jeffrey. "Authenticity by Typing for Security Protocols". In: *Journal of Computer Security* 11.4 (2003), pp. 451–520. URL: http://content.iospress.com/articles/journal-of-computer-security/jcs189 (cit. on p. 1).

[11] Hans Hüttel. *Computing Effects for Correspondence Types*. http://vbn.aau.dk/ws/files/18916151/fcs2009.pdf. 2009 (cit. on pp. 2, 22).

[12] Hans Hüttel. "Typed ψ-calculi". In: *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*. Ed. by Joost-Pieter Katoen and Barbara König. Vol. 6901. Lecture Notes in Computer Science. Springer, 2011, pp. 265–279. ISBN: 9783642232169. DOI: 10.1007/978-3-642-23217-6_18. URL: http://dx.doi.org/10.1007/978-3-642-23217-6_18 (cit. on pp. 2, 3, 5, 9, 10).

[13] Hans Hüttel. "Types for Resources in ψ-calculi". In: *Trustworthy Global Computing - 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers*. Ed. by Martín Abadi and Alberto Lluch-Lafuente. Vol. 8358. Lecture Notes in Computer Science. Springer, 2013, pp. 83–102. ISBN: 9783319051185. DOI: 10.1007/978-3-319-05119-2_6. URL: http://dx.doi.org/10.1007/978-3-319-05119-2_6 (cit. on pp. 2, 3, 37, 40, 49, 51, 59).

[14] Atsushi Igarashi and Naoki Kobayashi. "A generic type system for the Pi-calculus". In: *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*. Ed. by Chris Hankin and Dave Schmidt. ACM, 2001, pp. 128–141. ISBN: 1581133367. DOI: 10.1145/360204.360215. URL: http://doi.acm.org/10.1145/360204.360215 (cit. on p. 2).

[15] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. "Linearity and the pi-calculus". In: *ACM Trans. Program. Lang. Syst.* 21.5 (1999), pp. 914–947. DOI: 10.1145/330249.330251. URL: http://doi.acm.org/10.1145/330249.330251 (cit. on pp. 1, 2, 44).

[16] Barbara König. "Analysing Input/Output-Capabilities of Mobile Processes with a Generic Type System". In: *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*. Ed. by Ugo Montanari, José D. P. Rolim, and Emo Welzl. Vol. 1853. Lecture Notes in Computer Science. Springer, 2000, pp. 403–414. ISBN: 3540677151. DOI: 10.1007/3-540-45022-X_34. URL: http://dx.doi.org/10.1007/3-540-45022-X_34 (cit. on p. 2).

[17] Robin Milner. *A Calculus of Communicating Systems*. Vol. 92. Lecture Notes in Computer Science. Springer, 1980. ISBN: 3540102353. DOI: 10.1007/3-540-10235-3. URL: http://dx.doi.org/10.1007/3-540-10235-3 (cit. on p. 1).

[18] Robin Milner. *The polyadic π-calculus: a tutorial*. Springer, 1993 (cit. on pp. 1, 2).

[19] Flemming Nielson, Hanne Riis Nielson, Hongyan Sun, Mikael Buchholtz, René Rydhof Hansen, Henrik Pilegaard, and Helmut Seidl. "The Succinct Solver Suite". In: *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*. Ed. by Kurt Jensen and Andreas Podelski. Vol. 2988. Lecture Notes in Computer Science. Springer, 2004, pp. 251–265. ISBN: 354021299X. DOI: 10.1007/978-3-540-24730-2_21. URL: http://dx.doi.org/10.1007/978-3-540-24730-2_21 (cit. on pp. 7, 8).

[20] Flemming Nielson, Helmut Seidl, and Hanne Riis Nielson. "A Succinct Solver for ALFP". In: *Nord. J. Comput.* 9.4 (2002), pp. 335–372 (cit. on pp. 7, 13).

[21] Luca Padovani. "Type Reconstruction for the Linear π-Calculus with Composite and Equi-Recursive Types". In: *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*. Ed. by Anca Muscholl. Vol. 8412.

Lecture Notes in Computer Science. Springer, 2014, pp. 88–102. ISBN: 9783642548291. DOI: 10.1007/978-3-642-54830-7_6. URL: http://dx.doi.org/10.1007/978-3-642-54830-7_6 (cit. on p. 2).

[22] Jens Palsberg and Michael I. Schwartzbach. "Object-Oriented Type Inference". In: *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'91), Sixth Annual Conference, Phoenix, Arizona, USA, October 6-11, 1991, Proceedings.* Ed. by Andreas Paepcke. ACM, 1991, pp. 146–161. ISBN: 0201554178. DOI: 10.1145/117954.117965. URL: http://doi.acm.org/10.1145/117954.117965 (cit. on p. 2).

[23] James Riely and Matthew Hennessy. "A Typed Language for Distributed Mobile Processes (Extended Abstract)". In: *POPL '98, Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, CA, USA, January 19-21, 1998.* Ed. by David B. MacQueen and Luca Cardelli. ACM, 1998, pp. 378–390. ISBN: 0897919793. DOI: 10.1145/268946.268978. URL: http://doi.acm.org/10.1145/268946.268978 (cit. on p. 1).

[24] Martín Abadi and Andrew D. Gordon. "A Calculus for Cryptographic Protocols: The Spi Calculus". In: *CCS '97, Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, April 1-4, 1997.* Ed. by Richard Graveman, Philippe A. Janson, Clifford Neumann, and Li Gong. ACM, 1997, pp. 36–47. ISBN: 0897919122. DOI: 10.1145/266420.266432. URL: http://doi.acm.org/10.1145/266420.266432 (cit. on p. 1).

[25] Hongyan Sun. *User's Guide for the Succinct Solver (V2.0).* http://www2.imm.dtu.dk/cs_SuccinctSolver/download.htm (cit. on p. 8).

Appendix

APPENDIX A

Succinct Solver solutions

A.1. Succinct Solver solution for the example in Section 3.4.3

```
The Universe:
    (t1, t2, t5, t7, t10, t6, t14, t16, t13, t19, x1, t3, t8',
    x2, t11, t12', x3, t4, t15', x4, t17, t18', t8, x, t12, d, t15, y, t18,
    g, xi1, l, m1)

Relation LAL/2:
    (t6, t19), (t6, t16), (t6, t14), (t5, t13), (t5, t10), (t5, t7),
    (t1, t2),

Relation EQ/2: (m1, m1), (l, l), (xi1, xi1), (g, g), (t18, t18), (y, y),
    (t15, t15), (d, d), (t12, t12), (x, x), (t8, t8), (t18', t18'),
    (t17, t4), (t17, t17), (x4, x3), (x4, x4), (t15', t15'), (t4, t17),
    (t4, t4), (x3, x4), (x3, x3), (t12', t12'), (t11, t3), (t11, t11),
    (x2, x1), (x2, x2), (t8', t8'), (t3, t11), (t3, t3), (x1, x2), (x1, x1),
    (t13, t7), (t13, t10), (t13, t13), (t19, t14), (t19, t16), (t19, t19),
    (t1, t1), (t5, t5), (t6, t6), (t2, t2), (t7, t13), (t7, t7), (t7, t10),
    (t14, t19), (t14, t14), (t14, t16), (t16, t14), (t16, t16), (t16, t19),
    (t10, t7), (t10, t10), (t10, t13),

Relation IsPair_/4:
    (t10, x1, t11, t8'), (t10, x1, t3, t8'), (t10, x2, t3, t12'),
    (t10, x2, t11, t12'), (t16, x3, t17, t15'), (t16, x3, t4, t15'),
    (t16, x4, t4, t18'), (t16, x4, t17, t18'), (t19, x3, t17, t15'),
    (t19, x3, t4, t15'), (t19, x4, t4, t18'), (t19, x4, t17, t18'),
    (t14, x4, t4, t18'), (t14, x4, t17, t18'), (t14, x3, t17, t15'),
    (t14, x3, t4, t15'), (t13, x1, t11, t8'), (t13, x1, t3, t8'),
    (t13, x2, t3, t12'), (t13, x2, t11, t12'), (t7, x2, t3, t12'),
    (t7, x2, t11, t12'), (t7, x1, t11, t8'), (t7, x1, t3, t8'),

Relation IsAbs/5:
    (t18', t18, g, x4, t17), (t15', t15, y, x3, t4), (t12', t12, d, x2, t11),
    (t8', t8, x, x1, t3),

Relation IsOk_/1:
    (t15), (t8),

Relation CanOk_/2:
    (t8, xi1),

Relation FrmlMsg/3:
    (xi1, l, m1),

Relation MsgPai/3:
```

```
    (m1, x, y),

Relation IsOk/1:
    (t18), (t18'), (t15'), (t12), (t12'), (t8'), (t8), (t15),

Relation CanOk/2:
    (t8, xi1),

Relation ICO/4:
    (t8, x1, x, xi1), (t18, x4, g, xi1), (t15, x3, y, xi1), (t15, x2, d, xi1),
    (t12, x2, d, xi1),

Relation IsPair/4:
    (t16, x3, t4, t15'), (t16, x3, t17, t15'), (t16, x4, t4, t18'),
    (t16, x4, t17, t18'), (t19, x3, t4, t15'), (t19, x3, t17, t15'),
    (t19, x4, t4, t18'), (t19, x4, t17, t18'), (t14, x3, t17, t15'),
    (t14, x3, t4, t15'), (t14, x4, t4, t18'), (t14, x4, t17, t18'),
    (t10, x1, t3, t8'), (t10, x1, t11, t8'), (t10, x2, t3, t12'),
    (t10, x2, t11, t12'), (t13, x1, t3, t8'), (t13, x1, t11, t8'),
    (t13, x2, t3, t12'), (t13, x2, t11, t12'), (t7, x1, t11, t8'),
    (t7, x1, t3, t8'), (t7, x2, t3, t12'), (t7, x2, t11, t12'),

Relation IsCha_/2:
    (t1, t2), (t5, t13), (t5, t7), (t5, t10), (t6, t19), (t6, t14),
    (t6, t16),

Relation IsCha/2:
    (t6, t16), (t6, t14), (t6, t19), (t5, t10), (t5, t7), (t5, t13),
    (t1, t2),

Relation IsKey_/2:


Relation IsKey/2:


Relation IsUn/1:


Relation MsgEnc/3:


Relation ACO/3:
    (t12', x2, xi1), (t18', x4, xi1), (t15', x3, xi1),
    (t8', x1, xi1),

Relation AFM/4:
    (x2, xi1, l, m1), (x4, xi1, l, m1), (x3, xi1, l, m1),
    (x1, xi1, l, m1),

Relation Abs/3:
    (x, x1, y), (x, x1, m1),

Relation AMP/4:
```

```
    (x4, m1, d, x4), (x2, m1, x2, y), (x3, m1, d, x3),
    (x1, m1, x1, y),

Relation AME/4:


Relation IFM/5:
    (x1, x, xi1, l, m1), (x3, y, xi1, l, m1), (x4, g, xi1, l, m1),
    (x2, d, xi1, l, m1),

Relation Abs/5:
    (d, x2, x2, d, y), (d, x2, x2, d, m1), (x, x1, x1, x, y),
    (x, x1, x1, x, m1), (g, x4, x4, g, d), (g, x4, x4, g, m1),
    (y, x3, x2, d, d), (y, x3, x2, d, m1), (y, x3, x3, y, d),
    (y, x3, x3, y, m1),

Relation IMP/5:
    (x4, g, m1, d, g), (x3, y, m1, d, y), (x2, d, m1, d, y),
    (x1, x, m1, x, y),

Relation Ins/3:
    (x1, x, y), (x1, x, m1), (x2, d, y), (x2, d, m1), (x3, y, d),
    (x3, y, m1), (x4, g, d), (x4, g, m1),

Relation IsAbs-03412/5:
    (t18', x4, t17, t18, g), (t15', x3, t4, t15, y), (t12', x2, t11, t12, d),
    (t8', x1, t3, t8, x),

Relation LAL-10/2:
    (t13, t5), (t19, t6), (t16, t6), (t14, t6), (t10, t5), (t7, t5), (t2, t1),

Relation IsKey_-10/2:


Relation IsAbs-10234/5:
    (t18, t18', g, x4, t17), (t15, t15', y, x3, t4), (t12, t12', d, x2, t11),
    (t8, t8', x, x1, t3),
```

A.2. Succinct Solver solution for the example in Section 4.3.1

```
The Universe:
    (t_1, t_1_1, t_1_2, t_1_3, t_2, t_2_1, t_2_2, t_2_3, t_3, t_3_1, t_3_2,
    t_3_3, t_4, t_4_1, t_4_2, t_4_3, t_1_11, t_2_12, t_1_12, t_1_13, t_2_11,
    t_2_13, t_3_11, t_3_12, t_3_13, t_4_11, t_4_12, t_4_13, t_1_21, t_5,
    t_1_22, t_1_23, t_2_21, t_2_22, t_2_23, t_3_21, t_3_22, t_3_23, t_4_21,
    t_4_22, t_4_23, t_5_231, t_3_232, t_1_231, t_1_232, t_1_233, t_2_231,
    t_2_232, t_2_233, t_3_231, t_3_233, t_4_231, t_4_232, t_4_233, t_5_232,
    t_5_233, t_5_2331, t_4_2332, t_1_2331, t_1_2332, t_1_2333, t_2_2331,
    t_2_2332, t_2_2333, t_3_2331, t_3_2332, t_3_2333, t_4_2331, t_4_2333,
    t_5_2332, t_5_2333, t_2_31, t_6, t_1_31, t_1_32, t_1_33, t_2_32, t_2_33,
    t_3_31, t_3_32, t_3_33, t_4_31, t_4_32, t_4_33, t_2_331, t_7, t_1_331,
    t_1_332, t_1_333, t_2_332, t_2_333, t_3_331, t_3_332, t_3_333, t_4_331,
    t_4_332, t_4_333, t_6_331, t_6_332, t_6_333)
```

```
Relation ADD3/4:
    (t_6, t_6_331, t_6_332, t_6_333), (t_4_33, t_4_331, t_4_332, t_4_333),
    (t_3_33, t_3_331, t_3_332, t_3_333), (t_2_33, t_2_331, t_2_332, t_2_333),
    (t_1_33, t_1_331, t_1_332, t_1_333), (t_4_3, t_4_31, t_4_32, t_4_33),
    (t_3_3, t_3_31, t_3_32, t_3_33), (t_2_3, t_2_31, t_2_32, t_2_33),
    (t_1_3, t_1_31, t_1_32, t_1_33), (t_5_233, t_5_2331, t_5_2332, t_5_2333),
    (t_4_233, t_4_2331, t_4_2332, t_4_2333),
    (t_3_233, t_3_2331, t_3_2332, t_3_2333),
    (t_2_233, t_2_2331, t_2_2332, t_2_2333),
    (t_1_233, t_1_2331, t_1_2332, t_1_2333),
    (t_5, t_5_231, t_5_232, t_5_233), (t_4_23, t_4_231, t_4_232, t_4_233),
    (t_3_23, t_3_231, t_3_232, t_3_233), (t_2_23, t_2_231, t_2_232, t_2_233),
    (t_1_23, t_1_231, t_1_232, t_1_233), (t_4_2, t_4_21, t_4_22, t_4_23),
    (t_3_2, t_3_21, t_3_22, t_3_23), (t_2_2, t_2_21, t_2_22, t_2_23),
    (t_1_2, t_1_21, t_1_22, t_1_23), (t_4_1, t_4_11, t_4_12, t_4_13),
    (t_3_1, t_3_11, t_3_12, t_3_13), (t_2_1, t_2_11, t_2_12, t_2_13),
    (t_1_1, t_1_11, t_1_12, t_1_13), (t_4, t_4_1, t_4_2, t_4_3),
    (t_3, t_3_1, t_3_2, t_3_3), (t_2, t_2_1, t_2_2, t_2_3),
    (t_1, t_1_1, t_1_2, t_1_3),

Relation OUT/2:
    (t_2, t_4_2332), (t_2, t_3_232), (t_2_1, t_4_2332), (t_2_1, t_3_232),
    (t_2_12, t_4_2332), (t_2_12, t_3_232), (t_1, t_2_12), (t_1_1, t_2_12),
    (t_5, t_4_2332), (t_5, t_3_232), (t_5_233, t_4_2332), (t_5_2331, t_4_2332),
    (t_5_231, t_3_232), (t_1_11, t_2_12),

Relation IN/2:
    (t_2, t_6), (t_2, t_7), (t_1, t_5), (t_1_2, t_5), (t_2_3, t_7), (t_2_3, t_6),
    (t_2_33, t_7), (t_2_331, t_7), (t_2_31, t_6), (t_1_21, t_5),

Relation EQ/2:
    (t_6_333, t_6_333), (t_6_332, t_6_332), (t_6_331, t_6_331),
    (t_4_333, t_4_333), (t_4_332, t_4_332), (t_4_331, t_4_331),
    (t_3_333, t_3_333), (t_3_332, t_3_332), (t_3_331, t_3_331),
    (t_2_333, t_2_333), (t_2_332, t_2_332), (t_1_333, t_1_333),
    (t_1_332, t_1_332), (t_1_331, t_1_331), (t_7, t_4_2332),
    (t_7, t_3_232), (t_7, t_6), (t_7, t_7), (t_2_331, t_2_331),
    (t_4_33, t_4_33), (t_4_32, t_4_32), (t_4_31, t_4_31), (t_3_33, t_3_33),
    (t_3_32, t_3_32), (t_3_31, t_3_31), (t_2_33, t_2_33), (t_2_32, t_2_32),
    (t_1_33, t_1_33), (t_1_32, t_1_32), (t_1_31, t_1_31), (t_6, t_4_2332),
    (t_6, t_3_232), (t_6, t_7), (t_6, t_6), (t_2_31, t_2_31),
    (t_5_2333, t_5_2333), (t_5_2332, t_5_2332),
    (t_4_2333, t_4_2333), (t_4_2331, t_4_2331), (t_3_2333, t_3_2333),
    (t_3_2332, t_3_2332), (t_3_2331, t_3_2331), (t_2_2333, t_2_2333),
    (t_2_2332, t_2_2332), (t_2_2331, t_2_2331), (t_1_2333, t_1_2333),
    (t_1_2332, t_1_2332), (t_1_2331, t_1_2331), (t_4_2332, t_6),
    (t_4_2332, t_3_232), (t_4_2332, t_7), (t_4_2332, t_4_2332),
    (t_5_2331, t_5_2331), (t_5_233, t_5_233), (t_5_232, t_5_232),
    (t_4_233, t_4_233), (t_4_232, t_4_232), (t_4_231, t_4_231),
    (t_3_233, t_3_233), (t_3_231, t_3_231), (t_2_233, t_2_233),
    (t_2_232, t_2_232), (t_2_231, t_2_231), (t_1_233, t_1_233),
    (t_1_232, t_1_232), (t_1_231, t_1_231), (t_3_232, t_4_2332),
    (t_3_232, t_7), (t_3_232, t_6), (t_3_232, t_3_232),
    (t_5_231, t_5_231), (t_4_23, t_4_23), (t_4_22, t_4_22),
```

```
    (t_4_21, t_4_21), (t_3_23, t_3_23), (t_3_22, t_3_22),
    (t_3_21, t_3_21), (t_2_23, t_2_23), (t_2_22, t_2_22), (t_2_21, t_2_21),
    (t_1_23, t_1_23), (t_1_22, t_1_22), (t_5, t_2_12), (t_5, t_5),
    (t_1_21, t_1_21), (t_4_13, t_4_13), (t_4_12, t_4_12), (t_4_11, t_4_11),
    (t_3_13, t_3_13), (t_3_12, t_3_12), (t_3_11, t_3_11), (t_2_13, t_2_13),
    (t_2_11, t_2_11), (t_1_13, t_1_13), (t_1_12, t_1_12), (t_2_12, t_5),
    (t_2_12, t_2_12), (t_1_11, t_1_11), (t_4_3, t_4_3), (t_4_2, t_4_2),
    (t_4_1, t_4_1), (t_4, t_4), (t_3_3, t_3_3), (t_3_2, t_3_2), (t_3_1, t_3_1),
    (t_3, t_3), (t_2_3, t_2_3), (t_2_2, t_2_2), (t_2_1, t_2_1), (t_2, t_2),
    (t_1_3, t_1_3), (t_1_2, t_1_2), (t_1_1, t_1_1), (t_1, t_1),

Relation NOTEMP/1:
    (t_2_12), (t_5), (t_7), (t_3_232), (t_4_2332), (t_6),

Relation UN/1:
    (t_5), (t_2_12), (t_2_1), (t_2_3), (t_2),

Relation ADD2/3:
```

A.3. Succinct Solver solution for the example in Section 4.4.2

```
The Universe:
    (s1, r18, n18, m18, s11, r19, n19, m19, s12, r20, n20, m20, s23, r21, n21,
    m21, s13, r22, n22, m22, s14, r23, n23, m23, s17, r24, n24, m24, r25, n25,
    m25, r26, n26, m26, s15, s16, t1, t1_1, t1_2, t2, t2_1, t2_2, t3, t3_1,
    t3_2, s2, r27, n27, m27, r28, n28, m28, r29, n29, m29, s3, r3, n3, m3,
    t1_11, r1, n1, m1, r2, n2, m2, s4, r4, n4, m4, t4, r5, n5, m5, r6, n6, m6,
    s5, r30, n30, m30, r31, n31, m31, r32, n32, m32, s6, r33, n33, m33, t1_12,
    t1_13, t2_11, t2_12, t2_13, t3_11, t3_12, t3_13, r42, n42, m42, r43, n43,
    m43, r44, n44, m44, r45, n45, m45, s7, r7, n7, m7, t2_131, r8, n8, m8, r9,
    n9, m9, s8, r10, n10, m10, t4_132, r11, n11, m11, r12, n12, m12, s10, r34,
    n34, m34, r35, n35, m35, r36, n36, m36, s9, r37, n37, m37, t1_131, t1_132,
    t1_133, t2_132, t2_133, t3_131, t3_132, t3_133, t4_131, t4_133, r46, n46,
    m46, r47, n47, m47, r48, n48, m48, r49, n49, m49, s19, r13, n13, m13,
    t1_211, r14, n14, m14, r15, n15, m15, s20, r16, n16, m16, t3_212, r17,
    n17, m17, s22, r38, n38, m38, r39, n39, m39, r40, n40, m40, s21, r41, n41,
    m41, t1_21, t1_212, t1_213, t2_21, t2_211, t2_212, t2_213, t3_21, t3_211,
    t3_213, s18, r50, n50, m50, r51, n51, m51, r52, n52, m52, r53, n53, m53)

Relation ASS/4:
    (s18, r50, n50, m50), (s21, r51, n51, m51), (s21, r41, n41, m41),
    (s22, r38, n38, m38), (s20, r52, n52, m52), (s20, r17, n17, m17),
    (s20, r40, n40, m40), (s20, r16, n16, m16), (s19, r53, n53, m53),
    (s19, r15, n15, m15), (s19, r39, n39, m39), (s19, r13, n13, m13),
    (s9, r47, n47, m47), (s9, r37, n37, m37), (s10, r34, n34, m34),
    (s8, r48, n48, m48), (s8, r11, n11, m11), (s8, r36, n36, m36),
    (s8, r10, n10, m10), (s7, r49, n49, m49), (s7, r9, n9, m9),
    (s7, r35, n35, m35), (s7, r7, n7, m7), (s6, r46, n46, m46),
    (s6, r43, n43, m43), (s6, r33, n33, m33), (s5, r30, n30, m30),
    (s4, r44, n44, m44), (s4, r5, n5, m5), (s4, r32, n32, m32), (s4, r4, n4, m4),
    (s3, r45, n45, m45), (s3, r2, n2, m2), (s3, r31, n31, m31), (s3, r3, n3, m3),
    (s2, r42, n42, m42), (s2, r29, n29, m29), (s2, r28, n28, m28),
    (s2, r27, n27, m27), (s17, r24, n24, m24), (s14, r23, n23, m23),
```

```
    (s13, r26, n26, m26), (s13, r22, n22, m22), (s23, r21, n21, m21),
    (s12, r20, n20, m20), (s11, r25, n25, m25), (s11, r19, n19, m19),
    (s1, r18, n18, m18),

Relation LESSEQ/2:
    (m24, m42), (m24, m24), (m24, m28), (m24, m18), (m24, m29), (m24, m30),
    (m24, m27), (m42, m42), (m42, m24), (m42, m28), (m42, m18), (m42, m29),
    (m42, m30), (m42, m27), (m18, m42), (m18, m24), ...

Relation EQ/2:
    (m53, m13), (m53, m39), (m53, m15), (m53, m53), (n53, n3), (n53, n31),
    (n53, n45), (n53, n13), (n53, n39), (n53, t1), (n53, t1_11),
    (n53, t1_211), (n53, t1_12), (n53, t1_21), (n53, t1_13), (n53, t1_212),
    (n53, n2), ...

Relation REP/1:
    (r42), (r24), (r28), (r18), (r29), (r30), (r27),

Relation ASSCHAN/1:
    (s19), (s7), (s3),

Relation FAIL/1:


Relation LIN/1:
    (r45), (r31), (r2), (r44), (r32), (r5), (r33), (r43), (r34), (r11), (r36),
    (r9), (r35), (r38), (r21), (r17), (r40), (r15), (r39), (r53), (r52),
    (r50), (r16), (r13), (r49), (r48), (r46), (r10), (r7), (r4), (r3),

Relation VAL/1:
    (t4_133), (t4_131), (t3), (t3_11), (t3_213), (t3_12), (t3_211), (t3_13),
    (t3_2), (t3_131), (t3_21), (t3_132), (t3_1), (t3_133), (t3_212), (t4_132),
    (t4),

Relation CHAN/1:
    (t2_213), (t2_12), (t2_2), (t2_13), (t2_212), (t2_132), (t2_1), (t2_133),
    (t2_211), (n9), (t2_21), (n49), (t2), (n35), (t2_11), (n7), (n2), (t1_2),
    (t1_131), (n15), (t1_132), (t1_213), (t1_133), (t1_13), (n53), (t1_1),
    (n39), (t1_12), (n13), (t1_212), (n45), (t1), (n31), (t1_21), (n3),
    (t1_211), (t2_131), (t1_11),

Relation LESS/2:
    (m46, t1_12), (m46, t1_13), (m46, t1_212), (m46, n2), (m46, t1_2),
    (m46, t1_131), (m46, n15), (m46, t1_132), (m46, t1_213), (m46, t1_133),
    (m46, t1_211), (m46, n53), (m46, t1_1), (m46, n39), (m46, t1_11), (m46, n13),
    (m46, t1), (m46, n31), (m46, t1_21), (m46, n3), (m46, n45), (m33, t1_12),
    (m33, t1_13), (m33, t1_212), (m33, n2), (m33, t1_2), (m33, t1_131),
    (m33, n15), (m33, t1_132), (m33, t1_213), (m33, t1_133), (m33, t1_211),
    (m33, n53), (m33, t1_1), (m33, n39), (m33, t1_11), (m33, n13), (m33, t1),
    (m33, n31), (m33, t1_21), (m33, n3), (m33, n45), (m34, t1_12), (m34, t1_13),
    (m34, t1_212), (m34, n2), (m34, t1_2), (m34, t1_131), (m34, n15),
    (m34, t1_132), (m34, t1_213), (m34, t1_133), (m34, t1_211), (m34, n53),
    (m34, t1_1), (m34, n39), (m34, t1_11), (m34, n13), (m34, t1), (m34, n31),
    (m34, t1_21), (m34, n3), (m34, n45), (m43, t1_12), (m43, t1_13),
```

```
    (m43, t1_212), (m43, n2), (m43, t1_2), (m43, t1_131), (m43, n15),
    (m43, t1_132), (m43, t1_213), (m43, t1_133), (m43, t1_211), (m43, n53),
    (m43, t1_1), (m43, n39), (m43, t1_11), (m43, n13), (m43, t1), (m43, n31),
    (m43, t1_21), (m43, n3), (m43, n45),

Relation FAIL/2:
```

APPENDIX B

Proofs

B.1. Proofs for Chapter 3

B.1.1. Proof of Theorem 3.2.4.

To prove Theorem 3.2.4 we prove first that if there exists a Γ such that $\Gamma \vdash P$, then $\Gamma_P \vdash P \rightsquigarrow \phi$ such that $\mathscr{L}(\phi)$ is defined, and $\forall n : \tau_n \in \Gamma_P : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$, and then that if $\bigcup_{n \in \mathsf{fn}(P)} n : \tau_n \vdash P \rightsquigarrow \phi$ and ϕ is satisfiable, then there exists a Γ such that $\Gamma \vdash P$ and $\forall n : \tau_n \in \Gamma_P : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$.

If there exists a type environment Γ *such that* $\Gamma \vdash P$, *then* $\Gamma_P \vdash P \rightsquigarrow \phi$ *such that* $\mathscr{L}(\phi)$ *is defined, and* $\forall n : \tau_n \in \Gamma_P : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$. We prove this through induction in the typing rules from Table 3.1.2:

In: By the case we have $P = \underline{M}(\lambda \vec{x})N.P'$ and we use [In-C] to generate constraints, giving us: $\Gamma_P \vdash M \rightsquigarrow \tau_m; \phi_m$, $\Gamma_P \vdash (\lambda \vec{x})N \rightsquigarrow (\vec{\tau} \to \tau_u); \phi_n$, $\Gamma_P, \vec{x} : \vec{\tau} \vdash P' \rightsquigarrow \phi_p$, and $[\![\tau_m \nleftrightarrow \tau_u]\!]$. By Item 1 of Theorem 3.2.4 and the induction hypothesis—as $\Gamma \vdash M : U_s$, $\Gamma \vdash (\lambda \vec{x})N : \vec{T} \to U_o$, $\Gamma, \vec{x} : \vec{T} \vdash P'$—we know that $\mathscr{L}(\phi_m)$ is defined, $\mathscr{L}(\phi_m) \rhd \tau_m = U_s$, $\mathscr{L}(\phi_n)$ is defined, $\mathscr{L}(\phi_n) \rhd (\vec{\tau} \to \tau_u) = (\vec{T} \to U_o)$, $\mathscr{L}(\phi_p)$ is defined, and $\forall n : \tau_n \in \Gamma_P, \vec{x} : \vec{\tau} : n : (\mathscr{L}(\phi_p) \rhd \tau_n) \in \Gamma, \vec{x}, \vec{T}$. Moreover as $U_s \nleftrightarrow U_o$, we know from Item 5 that $[\![\tau_m \nleftrightarrow \tau_u]\!]$ is satisfiable and by the case we know the constraints ϕ_m, ϕ_n, and ϕ_p are not contradictory, and we thus conclude $\mathscr{L}(\phi_p \wedge \phi_n \wedge \phi_m \wedge [\![\tau_m \nleftrightarrow \tau_u]\!])$ is defined and, since $\forall n : \tau_n \in \Gamma_P, \vec{x} : \vec{\tau} : n : (\mathscr{L}(\phi_p) \rhd \tau_n) \in \Gamma, \vec{x}, \vec{T}$ and $\phi_p \wedge \phi_n \wedge \phi_m \wedge [\![\tau_m \nleftrightarrow \tau_u]\!] \Rightarrow \phi_p$, $\forall n : \tau_n \in \Gamma_P : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$.

Out: By the case we have $P = \overline{M}\, N.P'$ and we use [Out-C] to generate constraints, giving us: $\Gamma_P \vdash M \rightsquigarrow \tau_m; \phi_m$, $\Gamma_P \vdash N \rightsquigarrow \tau_n; \phi_n$, $\Gamma_P \vdash P' \rightsquigarrow \phi_p$, and $[\![\tau_m \nleftrightarrow \tau_n]\!]$. By Item 1 of Theorem 3.2.4 and the induction hypothesis—as $\Gamma \vdash M : T_s$, $\Gamma \vdash N : T_o$, $\Gamma \vdash P'$—we know $\mathscr{L}(\phi_m)$ is defined, $\mathscr{L}(\phi_m) \rhd \tau_m = T_s$, $\mathscr{L}(\phi_n)$ is defined, $\mathscr{L}(\phi_n) \rhd \tau_n = T_o$, $\mathscr{L}(\phi_p)$ is defined, and $\forall n : \tau_n \in \Gamma_P : n : (\mathscr{L}(\phi_p) \rhd \tau_n) \in \Gamma$. Moreover as $T_s \nleftrightarrow T_o$, we know from Item 5 that $[\![\tau_m \nleftrightarrow \tau_n]\!]$ is satisfiable and by the case we know the constraints ϕ_m, ϕ_n, and ϕ_p are not contradictory, and we thus conclude $\mathscr{L}(\phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\tau_m \nleftrightarrow \tau_n]\!])$ is defined and, since $\phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\tau_m \nleftrightarrow \tau_n]\!] \Rightarrow \phi_p$ and $\forall n : \tau_n \in \Gamma_P : n : (\mathscr{L}(\phi_p) \rhd \tau_n) \in \Gamma$, $\forall n : \tau_n \in \Gamma_P : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$.

Par: Trivial

Res: Trivial

Rep: Trivial

Ast: Trivial

Case: By the case we have $P = \mathbf{case}\ \sigma_1 : P_1, \ldots, \sigma_k : P_k$ and we use [Case-C] to generate constraints, giving us: $\Gamma_P \vdash \sigma_i \rightsquigarrow \phi_{si}$ and $\Gamma_P \vdash P_i \rightsquigarrow \phi_{pi}$ for $1 \leq i \leq k$. By Item 2 of Theorem 3.2.4 and the induction hypothesis—as $\Gamma \vdash \sigma_i$ and $\Gamma \vdash P_i$ for $1 \leq i \leq k$—we know that for $1 \leq i \leq k$, $\mathscr{L}(\phi_{si})$ and $\mathscr{L}(\phi_{pi})$ are defined, $\forall n : \tau_n \in \Gamma_\sigma : n : (\mathscr{L}(\phi_{si}) \rhd \tau) \in \Gamma$, and $\forall n : \tau_n \in \Gamma_\sigma : n : (\mathscr{L}(\phi_{pi}) \rhd \tau) \in \Gamma$. By the case we know none of the constraints ϕ_{si} and ϕ_{pi} are contradictory and we thus conclude $\mathscr{L}(\bigwedge_{i=1}^{k}(\phi_{si} \wedge \phi_{pi}))$ is defined and $\forall n : \tau_n \in \Gamma_\sigma : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$.

Pat: By the case we have $P = (\lambda \vec{x})M$ and we use [Pat-C] to generate constraints, giving us: $\Gamma_P, \vec{x} : \vec{\tau} \vdash M \rightsquigarrow \tau_m; \phi_m$. By Item 1 of Theorem 3.2.4—as $\Gamma, \vec{x} : \vec{T} \vdash$

$M : U$—we have $\mathscr{L}(\phi_m)$ is defined, $\mathscr{L}(\phi_m) \rhd \tau_m = U$, and $\forall n : \tau_n \in \Gamma_P, \vec{x} : \vec{\tau} : n : (\mathscr{L}(\phi_m) \rhd \tau_n) \in \Gamma, \vec{x}, \vec{T}$, and the result follows.

There exists a Γ *such that* $\Gamma \vdash P$*, if* $\bigcup_{n \in \mathsf{fn}(P)} n : \tau_n \vdash P \rightsquigarrow \phi$ *and* ϕ *is satisfiable.* Firstly we observe that the constraint generation rules and the type checking rules treat the environments identically. By this we mean that whenever e.g. a typing rule add a name to the environment, the constraint generation rule does the same, and vice versa. By letting the type variables assigned to a name in the constraint generation environment correspond the the type assigned to the same name in the type checking environment, it is obvious that the environments are manipulated identically in both derivations. Moreover we observe that the only constraint generated is the constraint $[\![\tau_1 \nleftrightarrow \tau_2]\!]$ in the [In-C] and [Out-C] rules (as terms, conditions, and assertions are undefined). We know from Item 4, that this means that $\mathscr{L}(\phi) \rhd \tau_m \nleftrightarrow \mathscr{L}(\phi) \rhd \tau_n$ and $\mathscr{L}(\phi) \rhd \tau_m \nleftrightarrow \mathscr{L}(\phi) \rhd \tau_u$, as required in the typing rules. We can thus conclude that if we successfully generate/derive constraints, we have the constraints corresponding to the side conditions of [In-T] and [Out-T] and if those are satisfiable we can successfully derive the type check, as the derivation of the two, because of the environments, are identical and the side conditions for the type check are satisfied.

B.1.2. Proof of Lemma 3.4.5.

For any message M**, if** $M \twoheadrightarrow \mu; \psi; \phi$ **then** $\mathscr{L}(\psi \wedge \phi) \lhd \mu = M$**. And for any assertion** $\ell(M)$**, if** $\ell(M) \twoheadrightarrow \xi; \psi; \phi$ **then** $\mathscr{L}(\psi \wedge \phi) \lhd \xi = \ell(M)$**.**

We first prove that for any message M, if $M \twoheadrightarrow \mu; \phi; \psi$ then $\mathscr{L}(\psi \wedge \phi) \lhd \mu = M$ by structural induction on M:

- $M = x$ **for some** $x \in \mathcal{N}$**::** In this case we know that $\psi = \mathsf{NamVar}(x)$ and $\phi = \mathtt{T}$. This means that $\mathsf{NamVar}(x) \in \mathscr{L}(\psi \wedge \phi)$ and therefore $\mathscr{L}(\psi \wedge \phi) \lhd \mu = x$.
- $M =$ **ok:** In this case we know that $\psi = \mathsf{MsgOk}(\mu)$ and $\psi = \mathtt{T}$. This means that $\mathsf{MsgOk}(\mu) \in \mathscr{L}(\psi \wedge \phi)$ and therefore $\mathscr{L}(\psi \wedge \phi) \lhd \mu = \mathsf{ok}$.
- $M =$ **pair**(M_1, M_2)**:** In this case we know that $M_1 \twoheadrightarrow \mu_1; \psi_1; \phi_1$, $M_2 \twoheadrightarrow \mu_2; \psi_2; \phi_2$, $\psi = \mathsf{MsgPai}(\mu, \mu_1, \mu_2)$, and $\phi = \psi_1 \wedge \phi_1 \wedge \psi_2 \wedge \phi_2$.

  From the induction hypothesis we get that, since $\phi = \psi_1 \wedge \phi_1 \wedge \psi_2 \wedge \phi_2$, meaning that $\mathscr{L}(\psi \wedge \phi)$ is also a solution for $\psi_1 \wedge \phi_1$ and $\psi_2 \wedge \phi_2$, $\mathscr{L}(\psi \wedge \phi) \lhd \mu_1 = M_1$ and $\mathscr{L}(\psi \wedge \phi) \lhd \mu_2 = M_2$.

  Since $\mathsf{MsgPai}(\mu, \mu_1, \mu_2) \in \mathscr{L}(\psi \wedge \phi)$ we therefore get $\mathscr{L}(\psi \wedge \phi) \lhd \mu = \mathsf{pair}(M_1, M_2)$.
- $M = \{M_1\}_{M_2}$**:** Similar to above
- $M =$ **fst** M'**:** In this case we know that $M' \twoheadrightarrow \mu'; \psi'; \phi'$, $\psi = \mathsf{MsgFst}(\mu, \mu')$ and $\phi = \psi' \wedge \phi'$.

  From the induction hypothesis we gen that, since $\phi = \psi' \wedge \phi'$, meaning that $\mathscr{L}(\psi \wedge \phi)$ is also a solution for $\psi' \wedge \phi'$, $\mathscr{L}(\psi \wedge \phi) \lhd \mu' = M'$.

  Since $\mathsf{MsgFst}(\mu, \mu') \in \mathscr{L}(\psi \wedge \phi)$, we therefore get $\mathscr{L}(\psi \wedge \phi) \lhd \mu = \mathsf{fst}\ M'$.
- $M =$ **snd** M'**:** Similar to above

We then prove that for any assertion $\ell(M)$, if $\ell(M) \twoheadrightarrow \xi; \psi; \phi$ then $\mathscr{L}(\psi \wedge \phi) \lhd \xi = \ell(M)$:

Since $\ell(M) \twoheadrightarrow \xi; \psi; \phi$, we know from rule [FM-C], that $M \twoheadrightarrow \mu; \psi_m; \phi_m$, $\psi = \mathsf{FrmIMsg}(\xi, \ell, \mu)$, and $\phi = \psi_m \wedge \phi_m$.

As proved above, since $\phi = \psi_m \wedge \phi_m$, meaning that $\mathscr{L}(\psi \wedge \phi)$ is also a solution for $\psi_m \wedge \phi_m$, $\mathscr{L}(\psi \wedge \phi) \lhd \mu = M$.

Since $\mathsf{FrmIMsg}(\xi, \ell, \mu) \in \mathscr{L}(\psi \wedge \phi)$, we therefore get $\mathscr{L}(\psi \wedge \phi) \lhd \xi = \ell(M)$.

B.1.3. Proof of Lemma 3.4.8.

(1) The relations Eq**, Eqa, and Eqi-**6 **are equivalence-relations**

(2) For all constraints ϕ **for which** $\Phi = \mathscr{L}(\phi)$**, type variables** a **and** b**, if** $\mathsf{Eq}(a, b) \in \Phi$ **then** $\Phi \rhd a = \Phi \rhd b$

(3) For all $M \twoheadrightarrow \mu_m; \psi_m; \varphi_m$ **and** $N \twoheadrightarrow \mu_n; \psi_n; \varphi_n$ **if** $\Phi = \mathscr{L}(\psi_{\{m,n\}} \wedge \varphi_{\{m,n\}})$ **then** $\mathsf{Eq}(\mu_m, \mu_n) \in \Phi \Rightarrow \Phi \lhd \mu_m = \Phi \lhd \mu_n$

(4) For all abstract message M **with index** χ_m**, variable** μ_m**, and constraint** ϕ_m **and abstract message** N **with index** χ_n**, variable** μ_n **and constraint** ϕ_n **if** $\Phi = \mathscr{L}(\phi_m \wedge \phi_n)$ **then** $\mathsf{Eqa}(\chi_m, \mu_m, \chi_n, \mu_n) \in \Phi \Rightarrow \Phi\Box(\chi_m, \mu_m) = \Phi\Box(\chi_n, \mu_n)$

(5) For all instantiated message M **with index** χ_m, μ'_m**, variable** μ_m**, and constraint** ϕ_m **and message** N **with variable** μ_n **and constraint** ϕ_n **if** $\Phi = \mathscr{L}(\phi_m \wedge \phi_n)$ **then** $\mathsf{Eqi}(\mu_n, \chi_m, \mu'_m, \mu_m) \in \Phi \Rightarrow \Phi\Diamond(\chi_m, \mu'_m, \mu_m) = \Phi \lhd \mu_n$

(6) For all instantiated message M **with index** χ_m, μ'_m**, variable** μ_m**, and constraint** ϕ_m **and instantiated message** N **with index** χ_n, μ'_n**, variable** μ_n**, and constraint** ϕ_n **if** $\Phi = \mathscr{L}(\phi_m \wedge \phi_n)$ **then** $\mathsf{Eqi}(\chi_m, \mu'_m, \mu_m, \chi_n, \mu'_n, \mu_n) \in \Phi \Rightarrow \Phi\Diamond(\chi_m, \mu'_m, \mu_m) = \Phi\Diamond(\chi_n, \mu'_n, \mu_n)$

We prove each of the cases individually.

(1) We consider each of the three relations:

Eq: Follows directly from row-group 1 of Table 3.4.13

Eqi-6: We consider each of the five axioms of row-group 4 in Table 3.4.14 individually. We first consider the case where $x = m$ as this case is immediately disallowed by the axiom. The reasoning is simple: if $x = m$ then surely the message should have been a and not m, since the constraint is capturing the message m for which we have instantiated x with a and thus the message would have been x, a, a instead of x, a, m. We now proceed to prove the three required properties for all valid selections of x, a, and m by induction in definition of Eqi where the ordering of the cases correspond to the order in which the axioms appear in Table 3.4.14. We can do this, as the axioms can be read as derivation where $P \Rightarrow Q$ denotes the derivation rule $\frac{P}{Q}$.

(a) We prove the three required properties

$\mathsf{Eqi}(x, a, m, x, a, m)$ for all x, a, and m: For all the cases where we have $x \neq m$ the result follows by the Eq case: since Eq is an equivalence relation, then surely $\mathsf{Eq}(m, m)$, $\mathsf{Eq}(x, x)$, and $\mathsf{Eq}(a, a)$ for all m and a and any valid x, giving us $\mathsf{Eqi}(x, a, m, x, a, m)$ as expected

$\mathsf{Eqi}(x, a, m, x', a', m') \Rightarrow \mathsf{Eqi}(x', a', m', x, a, m)$: Similar to the case above: since Eq is an equivalence relation we know that if $\mathsf{Eq}(a, a')$ then $\mathsf{Eq}(a', a)$ for any a and a'

$\mathsf{Eqi}(x, a, m, x', a', m') \wedge \mathsf{Eqi}(x', a', m', x'', a'', m'') \Rightarrow \mathsf{Eqi}(x, a, m, x'', a'', m'')$: Similar to the case above: since Eq is an equivalence relation we know that if $\mathsf{Eq}(a, a')$ and $\mathsf{Eq}(a', a'')$ then $\mathsf{Eq}(a, a'')$ for any a, a', and a''

(b) We prove the three required properties

$\mathsf{Eqi}(x, a, m, x, a, m)$ for all x, a, and m: Given an instantiated message $\mathsf{InsMsgPai}(x, a, m, m_1, m_2)$ we know, by the induction hypothesis, that $\mathsf{Eqi}(x, a, m_1, x, a, m_1)$ and $\mathsf{Eqi}(x, a, m_2, x, a, m_2)$. Moreover by the fact that Eq is an equivalence relation, we know that $\mathsf{Eq}(x, x)$, $\mathsf{Eq}(a, a)$, $\mathsf{Eq}(m, m)$, $\mathsf{Eq}(m_1, m_1)$, and $\mathsf{Eq}(m_2, m_2)$. This, by the axiom, gives us $\mathsf{Eqi}(x, a, m, x, a, m)$ and the result follows

$\mathsf{Eqi}(x, a, m, x', a', m') \Rightarrow \mathsf{Eqi}(x', a', m', x, a, m)$: Similar to the case above: since Eq is an equivalence relation we know that if $\mathsf{Eq}(a, a')$ then $\mathsf{Eq}(a', a)$ for any a and a'

$\mathsf{Eqi}(x, a, m, x', a', m') \wedge \mathsf{Eqi}(x', a', m', x'', a'', m'') \Rightarrow \mathsf{Eqi}(x, a, m, x'', a'', m'')$: Similar to the case above: since Eq is an equivalence relation we know that if $\mathsf{Eq}(a, a')$ and $\mathsf{Eq}(a', a'')$ then $\mathsf{Eq}(a, a'')$ for any a, a', and a''

(c) Similar to the case above

(d) Similar to the case above
(e) Similar to the case above

Eqa: Similar to the case above

(2) We assume we have some ϕ for which $\mathscr{L}(\phi) = \Phi$ such that for some type variables τ_1 and τ_2 we have $\mathsf{Eq}(\tau_1, \tau_2) \in \Phi$. We know proceed to prove that $\Phi \rhd \tau_1 = \Phi \rhd \tau_2$ by structural induction in $\Phi \rhd \tau_1$.

Un: Assume $\Phi \rhd \tau_1 = \mathsf{Un}$. This implies, by definition of $\rhd$, that we must have some $\mathsf{IsUn}(\tau_1)$ constraint in Φ, and by row-group 2 of Table 3.4.13 we have $\mathsf{IsUn}(\tau_2) \in \Phi$. By row-group 9 of the same table, we clearly have that there can be no other type assignment of τ_2 in Φ as all transitions from $\mathsf{IsX?}$ to IsX requires the absence of a $\mathsf{IsUn}(\tau_2)$ constraint, which we have just argued exists.

$\mathsf{Ok}(S)$: Assume $\Phi \rhd \tau_1 = \mathsf{Ok}(S)$. In a similar way to the case above can we deduce that there must exist some $\mathsf{IsOk}(\tau_2) \in \Phi$. We now need to argue that for every $\mathsf{CanOk}(\tau_1, \xi)$, $\mathsf{InsCanOk}(\tau_1, \chi, \mu, \xi)$, and $\mathsf{AbsCanOk}(\tau_1, \chi\xi)$ there exists a $\mathsf{CanOk}(\tau_2, \xi')$ for which $\mathsf{Eq}(\xi, xi')$, $\mathsf{InsCanOk}(\tau_2, \chi', \mu', \xi')$ for which $\mathsf{Eqi}(\chi, \mu, \xi, \chi'\mu', \xi')$, and $\mathsf{AbsCanOk}(\tau_2, \chi', \xi')$ for which $\mathsf{Eqa}(\chi, \xi, \chi', \xi')$. In order to do this, we investigate where we introduces new Eq-relations:

(a) If $\tau_1 = \tau_2$ i.e. they are the same variable
(b) If we have $\mathsf{IsCha}(\tau, \tau_1) \wedge \mathsf{IsCha}(\tau, \tau_2)$
(c) If we have $\mathsf{IsKey}(\tau, \tau_1) \wedge \mathsf{IsKey}(\tau, \tau_2)$
(d) If we have $\mathsf{IsPair}(\tau, \chi, \tau_1, \tau') \wedge \mathsf{IsPair}(\tau, \chi, \tau_2, \tau')$
(e) If we have $\mathsf{IsPair}(\tau, \chi, \tau', \tau_1) \wedge \mathsf{IsPair}(\tau, \chi, \tau', \tau_2)$

For (1) the result follows trivially. For (2) we can apply the two last axioms of row-group 10, and we know that S only contains the effects for which both τ_1 and τ_2 already have $\mathsf{CanOk?}$s and $\mathsf{InsCanOk?}$s that satisfy $\mathsf{Eq}(\xi, \xi')$ or $\mathsf{Eqi}(\chi, \mu, \xi, \chi', \mu', \xi')$ and the result follows. For (3) we can deduce $S = \emptyset$ as the only place we transform $\mathsf{CanOk?}$, $\mathsf{InsCanOk?}$, and $\mathsf{AbsCanOk?}$ to their definitive versions is in row-group 10 of Table 3.4.13, and none of these axioms are applicable in this case. We thus conclude the result must hold, since there are no CanOk, $\mathsf{InsCanOk}$, or $\mathsf{AbsCanOk}$ for τ_1. For (4) we can use a similar argumentation as for (3). For (5) we can apply the two first axioms of row-group 10, and we know that S only contains the effects for which both τ_1 and τ_2 already have $\mathsf{AbsCanOk?}$s that satisfy $\mathsf{Eqa}(\chi, \xi, \chi', \xi')$ and the result follows.

$\mathsf{Ch}(T)$: Assume $\Phi \rhd \tau_1 = \mathsf{Ch}(T)$. This implies, by definition of $\rhd$, we must have some $\mathsf{IsCha}(\tau_1, \tau_1')$ constraint in Φ for which $\Phi \rhd \tau_1' = T$. By row-group 3 of Table 3.4.13 and the assumption that $\mathsf{Eq}(\tau_1, \tau_2)$ we know, since $\mathsf{Eq}(\tau_1', \tau_1')$ by row-group 1, there must exists some $\mathsf{IsCha}(\tau_2, \tau_1')$ constraint in Φ. In addition, by row-group 9, we know there cannot exists any other IsX-constraint for τ_2, implying that $\Phi \rhd \tau_2 = \mathsf{Ch}(\Phi \rhd \tau_1') = \mathsf{Ch}(T) = \Phi \rhd \tau_1$

$\mathsf{Key}(T)$: Similar to the case above

$\mathsf{Pair}(\chi : T, U)$: Similar to the case above

(3) By Lemma 3.4.5 we know that if $M \twoheadrightarrow \mu; \psi; \varphi$ then $\mathscr{L}(\psi \wedge \varphi) \lhd \mu = M$. This implies that $\Phi \lhd \mu_m = M$ and $\Phi \lhd \mu_n = N$. We now proceed to show that if $\mathsf{Eq}(\mu_m, \mu_n)$ then $M = N$ by induction on M seen in row-group 1 and 11 of Table 3.4.13. The first case encompass all the axioms from row-group 1.

$M \in \mathcal{N}$: From the definition of $\lhd$ we know that this means that $\mathsf{NamVar}(\mu_m) \in \mathscr{L}(\psi \wedge \varphi)$. This means that the only way $\mathsf{Eq}(\mu_m, \mu_n)$ could have been

generated is by using one of the following axioms:

$$\forall a.\mathsf{Eq}(a,a)$$
$$\forall a:\forall b:\mathsf{Eq}(a,b)\Rightarrow\mathsf{Eq}(b,a)$$
$$\forall a:\forall b:\forall c:(\mathsf{Eq}(a,b)\wedge\mathsf{Eq}(b,c))\Rightarrow\mathsf{Eq}(a,c)$$
$$\forall t:\forall x:\forall u:\forall x_t:\forall x_u:\forall t_f:\forall t_s:\forall u_f:\forall u_s:(\mathsf{IsPair}(t,x_t,t_f,t_s)\wedge\mathsf{IsPair}(u,x_u,u_f,u_s)\wedge\mathsf{Eq}(t,u))\Rightarrow(\mathsf{Eq}(t_f,u_f)\wedge\mathsf{Eq}(x_t,x_u)\wedge\mathsf{Eq}(t_s,u_s))$$

If the first axiom was used, it is obvious that $M=N$, and if the second or third axioms were used, it would require other Eq prefixes to have been generated previously, which would have to be generated using one of the other two axioms. If the fourth axiom was used then we know that whenever we generate $\mathsf{IsPair?}(\tau_1,x,\tau_2,\tau_3)$ we also generate $\mathsf{PairVar}(x)$. This means that $\mathsf{PairVar}(x_t)\in\Phi$ and $\mathsf{PairVar}(x_u)\in\Phi$, and therefore either $\Phi\rhd\mu_m=\mu_m=\Phi\rhd\mu_n=\mu_n$ or $\Phi\rhd\mu_m=\chi=\Phi\rhd\mu_n$.

$M=\mathbf{pair}(M_1,M_2)$**:** Here we know from the definition of $\lhd$, that $\mathsf{MsgPai}(\mu_m,\mu_m^1,\mu_m^2)\in\mathscr{L}(\psi\wedge\varphi)$. This mean that $\mathsf{Eq}(\mu_m,\mu_n)$ must have been generated using the axiom

$$\forall m:\forall m_1:\forall m_2:\forall n:\forall n_1:\forall n_2:(\mathsf{MsgPai}(m,m_1,m_2)\wedge\mathsf{MsgPai}(n,n_1,n_2)\wedge\mathsf{Eq}(m_1,n_1)\wedge\mathsf{Eq}(m_2,n_2))\Rightarrow\mathsf{Eq}(m,n)$$

In this case we have $\mathsf{MsgPai}(\mu_n,\mu_n^1,\mu_n^2)$, $\mathsf{Eq}(\mu_m,\mu_n)$, $\mathsf{Eq}(\mu_m^1,\mu_n^1)$, and $\mathsf{Eq}(\mu_m^2,\mu_n^2)$. By the induction hypothesis we have $\Phi\lhd\mu_n^1=\Phi\lhd\mu_m^1=M_1$ and $\Phi\lhd\mu_n^2=\Phi\lhd\mu_m^2=M_2$. By definition of $\lhd$ we thus have $\Phi\lhd\mu_n=\mathsf{pair}(M_1,M_2)$.

$M=\{M_1\}_{M_2}$**:** Similar to the case above

$M=$ **ok:** Similar to the case above

$M=$ **fst** M'**:** Similar to the case above

$M=$ **snd** M'**:** Similar to the case above

(4) We prove this case by induction on $\Phi\Box(\chi_m,\mu_m)$.

$\Phi\Box(\chi_m,\mu_m)\in\mathcal{N}$**:** In this case we know from the definition of $\Box$, that have $\mathsf{NamVar}(m)\in\Phi$. This means that $\mathsf{Eqa}(\chi_m,\mu_m,\chi_n,\mu_n)$ must have been generated using the axiom $(\mathsf{NamVar}(m)\wedge\mathsf{NamVar}(m')\wedge\mathsf{Eq}(m,m')\wedge\mathsf{NamVar}(x)\wedge\mathsf{NamVar}(x')\wedge\mathsf{Eq}(x,x'))\Rightarrow\mathsf{Eqa}(x,m,x',m')$. This means that $\mathsf{NamVar}(\mu_n)$, $\mathsf{Eq}(\mu_m,\mu_n)$, $\mathsf{NamVar}(\chi_m)$, $\mathsf{NamVar}(\chi_n)$, and $\mathsf{Eq}(\chi_m,\chi_n)$. Since $\mathsf{Eq}(\mu_m,\mu_n)\in\Phi$, we know that $\Phi\lhd\mu_m=\Phi\lhd\mu_n$, as in they are the exact same name.

$\Phi\Box(\chi_m,\mu_m)=\mathbf{pair}(M_1,M_2)$**:** Here we know from the definition of $\Box$, that there exist μ_m^1 and μ_m^2 such that $\mathsf{AbsMsgPai}(\chi_m,\mu_m,\mu_m^1,\mu_m^2)$, $\Phi\Box(\chi_m,\mu_m^1)=M_1$, and $\Phi\Box(\chi_m,\mu_m^2)=M_2$. This means that $\mathsf{Eqa}(\chi_m,\mu_m,\chi_n,\mu_n)$ must have been generated using the axiom

$$(\mathsf{Eqa}(x_1,m_1,x_1',m_1')\wedge\mathsf{Eqa}(x_2,m_2,x_2',m_2')\wedge\mathsf{Eq}(x_1,x_2)\wedge\mathsf{Eq}(x_1',x_2')\wedge\mathsf{AbsMsgPai}(x,m,m_1,m_2)\wedge\mathsf{Eq}(x,x_1)\wedge\mathsf{AbsMsgPai}(x',m',m_1',m_2')\wedge\mathsf{Eq}(x',x_1'))\Rightarrow\mathsf{Eqa}(x,m,x',m')$$

In this case we have $\mathsf{AbsMsgPai}(\chi_n,\mu_n,\mu_n^1,\mu_n^2)$, $\mathsf{Eqa}(\chi_m,\mu_m^1,\chi_n,\mu_n^1)$, $\mathsf{Eqa}(\chi_m,\mu_m^2,\chi_n,\mu_n^2)$, and $\mathsf{Eqa}(\chi_m,\mu_m,\chi_n,\mu_n)$. By the induction hypothesis we have $\Phi\Box(\chi_n,\mu_n^1)=\Phi\Box(\chi_m,\mu_m^1)=M_1$ and $\Phi\Box(\chi_n,\mu_n^2)=\Phi\Box(\chi_m,\mu_m^2)=M_2$. By definition of $\Box$ and $\mathsf{AbsMsgPai}(\chi_n,\mu_n,\mu_n^1,\mu_n^2)$ we have $\Phi\Box(\chi_n,\mu_n)=\mathsf{pair}(M_1,M_2)$.

$\Phi\Box(\chi_m,\mu_m)=\{M_1\}_{M_2}$**:** Similar to the case above

$\Phi\Box(\chi_m,\mu_m)=$ **fst** M'**:** Similar to the case above

$\Phi\Box(\chi_m,\mu_m)=$ **snd** M'**:** Similar to the case above

(5) We prove this case by induction on $\Phi\Diamond(\chi_m,\mu_m',\mu_m)$.

$\Phi\Diamond(\chi_m,\mu_m',\mu_m)\in\mathcal{N}$**:** In this case we know from the definition of $\Diamond$, that $\mathsf{NamVar}(\mu_m)\in\Phi$. This means that the only way $\mathsf{Eqi}(\mu_n,\chi_m,\mu_m',\mu_m)$ could have been generated is by using the axiom $(\mathsf{NamVar}(n)\wedge\mathsf{NamVar}(n')\wedge\mathsf{Eq}(n,n')\wedge\neg\mathsf{Eq}(n',x))\Rightarrow\mathsf{Eqi}(n,x,m,n')$. In this case we have $\mathsf{NamVar}(\mu_n)$, $\mathsf{Eq}(\mu_n,\mu_m)$, and $\neg\mathsf{Eq}(\mu_m,\chi_m)$. As mentioned previously, the latter of the

constraints is to prevent nonsense instantiated messages like χ, μ, χ, which should obviously be χ, μ, μ. By the three former constraints we can deduce that $\mu_m = \mu_n$, and thus $\Phi \lhd \mu_n = \mu_n = \mu_m = \Phi\Diamond(\chi_m, \mu'_m, \mu_m)$.

$\Phi\Diamond(\chi_m, \mu'_m, \mu_m) = \mathbf{pair}(M_1, M_2)$**:** Here we know from the definition of $\Diamond$, that there exist μ^1_m and μ^2_m such that $\mathsf{InsMsgPai}(\chi_m, \mu'_m, \mu_m, \mu^1_m, \mu^2_m)$, $\Phi\Diamond(\chi_m, \mu'_m, \mu^1_m) = M_1$, and, $\Phi\Diamond(\chi_m, \mu'_m, \mu^2_m) = M_2$. This means that $\mathsf{Eqi}(\mu_n, \chi_m, \mu'_m, \mu_m)$ must have been generated using the axiom

$$(\mathsf{MsgPai}(m'_1, m'_2, m'_3) \wedge \mathsf{InsMsgPai}(m_1, m_2, m_3, m_4, m_5) \wedge \mathsf{Eqi}(m'_2, m_1, m_2, m_4) \wedge \mathsf{Eqi}(m'_3, m_1, m_2, m_5)) \Rightarrow \mathsf{Eqi}(m'_1, m_1, m_2, m_3)$$

In this case we have $\mathsf{MsgPai}(\mu_n, \mu^1_n, mu^2_n)$, $\mathsf{Eqi}(\mu^1_n, \chi_m, \mu'_m, \mu^1_m)$, $\mathsf{Eqi}(\mu^2_n, \chi_m, \mu'_m, \mu^2_m)$, and $\mathsf{Eqi}(\mu_n, \chi_m, \mu'_m, \mu_m)$. By the induction hypothesis we have $\Phi \lhd \mu^1_n = \Phi\Diamond(\chi_m, \mu'_m, \mu^1_m) = M_1$ and $\Phi \lhd \mu^2_n = \Phi\Diamond(\chi_m, \mu'_m, \mu^2_m) = M_2$. By definition of $\lhd$ and by $\mathsf{MsgPai}(\mu_n, \mu^1_n, \mu^2_n)$ we can conclude $\Phi \lhd \mu_n = \mathsf{pair}(M_1, M_2)$.

$\Phi\Diamond(\chi_m, \mu'_m, \mu_m) = \mathbf{fst}\ M$**:** Similar to the case above

$\Phi\Diamond(\chi_m, \mu'_m, \mu_m) = snd\ M$**:** Similar to the case above

$\Phi\Diamond(\chi_m, \mu'_m, \mu_m) = \{M_1\}_{M_2}$**:** Similar to the case above

(6) Similar to the two previous cases

B.1.4. Proof of Item 4 of Theorem 3.2.4 for correspondence assertions.

Let τ_1 and τ_2 be type variables and ϕ be a constraint. If there exists a solution to $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!])$, then $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!]) \rhd \tau_1 \leftarrowtriangle \mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!]) \rhd \tau_2$.

Due to the definition of $\leftarrowtriangle$, we need to show that either $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!]) \rhd \tau_1 = \mathsf{Ch}(\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!])) \rhd \tau_2)$ or $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!]) \rhd \tau_1 = \mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!]) \rhd \tau_2 = \mathsf{Un}$.

Since $[\![\tau_1 \leftarrowtriangle \tau_2]\!] = \mathsf{IsCha?}(\tau_1, \tau_2)$, we look at the axioms involving $\mathsf{IsCha?}(\tau_1, \tau_2)$. The axioms:

$$\forall t : \forall u : (\mathsf{IsOk?}(t) \wedge \mathsf{IsCha?}(t, u)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u))$$
$$\forall t : \forall u_1 : \forall u_2 : (\mathsf{IsCha?}(t, u_1) \wedge \mathsf{IsKey?}(t, u_2)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u_1) \wedge \mathsf{IsUn}(u_2))$$
$$\forall t : \forall x : \forall u : \forall t_f : \forall t_s : (\mathsf{IsCha?}(t, u) \wedge \mathsf{IsPair?}(t, x, t_f, t_s)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u) \wedge \mathsf{IsUn}(t_f) \wedge \mathsf{IsUn}(t_s))$$
$$\forall t_1 : \forall t_2 : (\mathsf{IsUn}(t) \wedge \mathsf{IsCha?}(t_1, t_2)) \Rightarrow \mathsf{IsUn}(t_2)$$
$$\forall t : \forall x : \forall u_1 : \forall u_2 : \forall t_f : \forall t_s : (\neg\mathsf{IsOk?}(t) \wedge \mathsf{IsCha?}(t, u_1) \wedge \neg\mathsf{IsKey?}(t, u_2) \wedge \neg\mathsf{IsPair?}(t, x, t_f, t_s) \wedge \neg\mathsf{IsUn}(t)) \Rightarrow \mathsf{IsCha}(t, u_1)$$

mean that if $\mathsf{IsCha?}(\tau_1, \tau_2)$ then either $\mathsf{IsCha}(\tau_1, \tau_2)$ or $\mathsf{IsUn}(\tau_1)$ and $\mathsf{IsUn}(\tau_2)$.

In the first case $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!]) \rhd \tau_1 = \mathsf{Ch}(\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!])) \rhd \tau_2)$, and in the second case $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!]) \rhd \tau_1 = \mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!]) \rhd \tau_2 = \mathsf{Un}$, proving the lemma.

B.1.5. Proof of Item 5 of Theorem 3.2.4 for correspondence assertions.

For any constraint generated by a process, ϕ, and any type variables, τ_1 and τ_2, if $\mathscr{L}(\phi) \rhd \tau_1 \leftarrowtriangle \mathscr{L}(\phi) \rhd \tau_2$ then $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!])$ exists.

The only way to generate Fail predicates is through a $((\bigwedge_{x:\tau\in\Gamma}(\forall\xi' : \forall\xi'' : \forall x : \forall\mu :$
$((\neg\mathsf{CanOk}(\tau, \xi') \vee \neg\mathsf{Eq}(\xi, \xi')) \wedge$
$(\neg\mathsf{InsCanOk}(\tau, x, \mu, \xi'') \vee \neg\mathsf{Eqi}(\xi, x, \mu, \xi'')))))) \Rightarrow$
$\mathsf{FAIL}(\xi))$ constraint generated by [End-C]. The only way this constraint would cause $\mathscr{L}(\phi \wedge [\![\tau_1 \leftarrowtriangle \tau_2]\!])$ to include a FAIL predicate when $\mathscr{L}(\phi)$ does not is if $\mathsf{IsCha?}(\tau_1, \tau_2)$ prevented an $\mathsf{IsOk}(\tau)$ and therefore also a $\mathsf{CanOk}(\tau, \xi')$ or $\mathsf{InsCanOk}(\tau, x, \mu, \xi'')$ predicate from being generated because of the axiom $\forall t : \forall x : \forall u_1 : \forall u_2 : \forall t_f : \forall t_s : (\mathsf{IsOk?}(t) \wedge \neg\mathsf{IsCha?}(t, u_1) \wedge \neg\mathsf{IsKey?}(t, u_2) \wedge \neg\mathsf{IsPair?}(t, x, t_f, t_s) \wedge \neg\mathsf{IsUn}(t)) \Rightarrow \mathsf{IsOk}(t)$. But that would require that $\tau = \tau_1$ and $\mathsf{IsOk}(\tau_1) \in \mathscr{L}(\phi)$, which in turn would mean that $\mathscr{L}(\phi) \rhd \tau_1 = \mathsf{Ok}(S)$, making it impossible that $\mathscr{L}(\phi) \rhd \tau_1 \leftarrowtriangle \mathscr{L}(\phi) \rhd \tau_2$.

B.1.6. Proof of Lemma 3.4.6.

For all τ and ϕ where $\mathsf{IsAbs}(\tau', \tau, a, x, \tau'') \in \mathscr{L}(\phi)$ and $\mathscr{L}(\phi) \rhd \tau = \mathsf{Ok}(S)$ we have $\mathscr{L}(\phi) \rhd' \tau' = \mathsf{Ok}(R)$ such that $R = S[\not{x}_a]$

For each $\ell(M)$ in S we have two cases: either a $\mathsf{CanOk}(\tau, \xi)$ or an $\mathsf{InsCanOk}(\tau, x, a, \xi)$ for some x, a, and ξ, using either $\lhd$ or $\Diamond$ have given rise to $\ell(M)$. These two cases represent the case where $\ell(M)$ is directly in the environment or the case where we have some $x : \mathsf{Ok}(S')$ in the environment where $\ell(M) \in S'$. The two cases are very similar, and only differ in the aspect that we use the "index" x, a for the $\mathsf{InsCanOk}$-case. We thus only proceed to prove the case without the "index", but it can be directly translated to the other case by adding the x, a-index and by prepending all relations with the Ins-prefix.

By the case we have $\ell(M) \in S$ is created by a $\mathsf{CanOk}(\tau, \xi)$ and its corresponding $\mathsf{FrmlMsg}(\xi, \ell, \mu)$ constraint such that $\mathscr{L}(\phi) \lhd \xi = \ell(M)$. In addition we have the constraint $\mathsf{IsAbs}(\tau', \tau, x, a, \tau'')$. For simplicity we assume $\mathscr{L}(\phi) \lhd m = M$ for any m, e.g. $\mathscr{L}(\phi) \lhd a = A$ and so forth. By the axioms of row-group 1 of Table 3.4.14 we have two cases: either $\mathsf{Eq}(\mu, a)$ or $\neg\mathsf{Eq}(\mu, a)$:

- If $\mathsf{Eq}(\mu, a)$ we have $\ell(M) = \ell(A)$ by Lemma 3.4.8 and since we are to replace all occurrences of a with x we simply get $\mathsf{AbsFrmlMsg}(x, \xi, \ell, x)$ which by definition of $\Box$, since $\mathsf{PairVar}(x) \in \mathscr{L}(\phi)$, gives us $\mathscr{L}(\phi)\Box(x, \xi) = \ell(\chi)$ and thus $\ell(\chi) \in R$ by $\rhd'$ as expected.
- If $\neg\mathsf{Eq}(\mu, a)$ we have $\ell(M) \neq \ell(A)$ and we get $\mathsf{AbsFrmlMsg}(x, \xi, \ell, \mu) \wedge \mathsf{Abs}(a, x, \mu)$. We now have to prove that $\mathsf{Abs}(a, x, \mu)$ represent the recursive substitution of a for x in the message μ, which would lead to $\mathscr{L}(\phi)\Box(x, \xi) = \ell(M[\not{x}_a])$ as expected.

  We now proceed to prove that $\mathsf{Abs}(a, x, \mu)$ indeed behaves as the recursive substitution $[\not{x}_a]$ on the message M. For this we have five cases corresponding to the five types of messages, namely: $\mathsf{MsgOk}(\mu)$, $\mathsf{MsgFst}(\mu_1, \mu_2)$, $\mathsf{MsgSnd}(\mu_1, \mu_2)$, $\mathsf{MsgEnc}(\mu_1, \mu_2, \mu_3)$, and $\mathsf{MsgPai}(\mu_1, \mu_2, \mu_3)$. For each of those cases we either have two or four sub-cases. As each of the five cases proceed in a similar way, we will only present the case for $\mathsf{MsgPai}(\mu_1, \mu_2, \mu_3)$ as seen below. The proof is done by induction on the length of the constraint-chain.

  By the case we have $\mathsf{IsAbs}(a, x, \mu)$ and $\mathsf{MsgPai}(\mu, \mu_1, \mu_2)$ such that $\mathscr{L}(\phi) \lhd \mu = \mathsf{pair}(M_1, M_2)$. We now proceed to each of the four sub-cases:

  $\mathsf{Eq}(a, \mu_1)$ and $\mathsf{Eq}(a, \mu_2)$: For this case, by the axioms in row-group 2 of Table 3.4.14, we get $\mathsf{AbsMsgPai}(x, \mu, x, x)$ which by $\Box$, since $\mathsf{PairVar}(x) \in \mathscr{L}(\phi)$, gives us $\mathscr{L}(\phi)\Box(x, \mu) = \mathsf{pair}(\chi, \chi)$ and clearly $\mathsf{pair}(\chi, \chi) = \mathsf{pair}(A, A)[\not{x}_A]$ and the result follows.

  $\mathsf{Eq}(a, \mu_1)$ and $\neg\mathsf{Eq}(a, \mu_2)$: For this case, by the axioms in row-group 2 of Table 3.4.14, we get $\mathsf{AbsMsgPai}(x, \mu, x, \mu_2) \wedge \mathsf{Abs}(a, x, \mu_2)$. By the induction hypothesis we know we have some $\mathsf{Abs}X(x, \mu_2, \dots)$ such that $\mathscr{L}(\phi)\Box(x, \mu_2) = M_2[\not{x}_A]$. Thus by definition of $\Box$ we know $\mathscr{L}(\phi)\Box(x, \mu) = \mathsf{pair}(\chi, M_2[\not{x}_A])$ and clearly $\mathsf{pair}(\chi, M_2[\not{x}_A]) = \mathsf{pair}(A, M_2)[\not{x}_A]$ and the result follows.

  $\neg\mathsf{Eq}(a, \mu_1)$ and $\mathsf{Eq}(a, \mu_2)$: Similar to the case above.

  $\neg\mathsf{Eq}(a, \mu_1)$ and $\neg\mathsf{Eq}(a, \mu_2)$: For this case, by the axioms in row-group 2 of Table 3.4.14, we get $\mathsf{AbsMsgPai}(x, \mu, \mu_1, \mu_2) \wedge \mathsf{Abs}(a, x, \mu_1) \wedge \mathsf{Abs}(a, x, \mu_2)$. By the induction hypothesis we know we have some $\mathsf{Abs}X(x, \mu_1, \dots)$ such that $\mathscr{L}(\phi)\Box(x, \mu_1) = M_1[\not{x}_A]$ and $\mathsf{Abs}X(x, \mu_2, \dots)$ such that $\mathscr{L}(\phi)\Box(x, \mu_2) = M_2[\not{x}_A]$. Thus by definition of $\Box$ we know $\mathscr{L}(\phi)\Box(x, \mu) = \mathsf{pair}(M_1[\not{x}_A], M_2[\not{x}_A])$ and clearly $\mathsf{pair}(M_1[\not{x}_A], M_2[\not{x}_A]) = \mathsf{pair}(M_1, M_2)[\not{x}_A]$ and the result follows.

B.1.7. Proof of Lemma 3.4.7.

For all τ' and ϕ where $\mathsf{IsAbs}(\tau', \tau, a, x, \tau'') \in \mathscr{L}(\phi)$ and $\mathscr{L}(\phi) \rhd' \tau' = \mathsf{Ok}(S)$ we have $\mathscr{L}(\phi) \rhd \tau = \mathsf{Ok}(R)$ such that $S[{}^{a}/_{\chi}] = R$

For each $\ell(M)$ in S we must have an $\mathsf{AbsCanOk}(\tau', x, \xi)$ and $\mathsf{AbsFrmlMsg}(x, \xi, \ell, \mu)$ for some x, μ, and ξ for which $\mathscr{L}(\phi)\Box(x, \xi) = \ell(M)$. In addition we have the constraint $\mathsf{IsAbs}(\tau', \tau, x, a, \tau'')$. For simplicity we assume $\mathscr{L}(\phi) \lhd m = M$ for any m, e.g. $\mathscr{L}(\phi) \lhd a = A$ and so forth. By the axioms of row-group 1 of Table 3.4.14 we have two cases: either $\mathsf{Eq}(\mu, x)$ or $\neg\mathsf{Eq}(\mu, x)$:

- If $\mathsf{Eq}(\mu, x)$ we have $\ell(M) = \ell(x)$ by Lemma 3.4.8 and since we are to replace all occurrences of x with A we simply get $\mathsf{InsCanOk}(t, x, a, \xi) \wedge \mathsf{InsFrmlMsg}(x, \xi, \ell, a) \wedge \mathsf{Ins}(x, a, a)$. Assuming—we prove this below—that $\mathsf{Ins}(x, a, a)$ represents the recursive substitution $[{}^{a}/_{\chi}]$, we will get, by definition of $\Diamond$, $\mathscr{L}(\phi)\Diamond(x, a, \xi) = \ell(A)$ and thus $\ell(A) \in R$ by $\rhd$ as expected.
- If $\neg\mathsf{Eq}(\mu, x)$ we have $\ell(M) \neq \ell(x)$ and we get $\mathsf{InsCanOk}(t, x, a, \xi) \wedge \mathsf{InsFrmlMsg}(x, \xi, \ell, \mu) \wedge \mathsf{Ins}(x, a, \mu)$. Assuming—we prove this below—that $\mathsf{Ins}(x, a, \mu)$ represents the recursive substitution $[{}^{a}/_{\chi}]$, we will get, by definition of $\Diamond$, $\mathscr{L}(\phi)\Diamond(x, a, \xi) = \ell(M[{}^{A}/_{\chi}])$ and thus $\ell(M[{}^{A}/_{\chi}]) \in R$ by $\rhd$ as expected.

  We now proceed to prove that $\mathsf{Ins}(x, a, \mu)$ indeed behaves as the recursive substitution $[{}^{a}/_{\chi}]$ on the message M. For this we have five cases corresponding to the five types of messages, namely: $\mathsf{AbsMsgOk}(x, \mu)$, $\mathsf{AbsMsgFst}(x, \mu_1, \mu_2)$, $\mathsf{AbsMsgSnd}(x, \mu_1, \mu_2)$, $\mathsf{AbsMsgEnc}(x, \mu_1, \mu_2, \mu_3)$, and $\mathsf{AbsMsgPai}(x, \mu_1, \mu_2, \mu_3)$. For each of those cases we either have two or four sub-cases. As each of the five cases proceed in a similar way, we will only present the case for $\mathsf{AbsMsgPai}(x, \mu_1, \mu_2, \mu_3)$ as seen below. The proof is done by induction on the length of the constraint-chain.

  By the case we have $\mathsf{Ins}(x, a, \mu)$ and $\mathsf{AbsMsgPai}(x, \mu, \mu_1, \mu_2)$ such that $\mathscr{L}(\phi)\Box(x, \mu) = \mathsf{pair}(M_1, M_2)$. We now proceed to each of the four sub-cases:

  $\mathsf{Eq}(x, \mu_1)$ and $\mathsf{Eq}(x, \mu_2)$: For this case, by the axioms in row-group 2 of Table 3.4.15, we get $\mathsf{InsMsgPai}(x, a, \mu, a, a) \wedge \mathsf{Ins}(x, a, a)$ which by $\Diamond$ gives us $\mathscr{L}(\phi)\Diamond(x, a, \mu) = \mathsf{pair}(A, A)$ and clearly $\mathsf{pair}(A, A) = \mathsf{pair}(\chi, \chi)[{}^{A}/_{\chi}]$ and the result follows.

  $\mathsf{Eq}(x, \mu_1)$ and $\neg\mathsf{Eq}(x, \mu_2)$: For this case, by the axioms in row-group 2 of Table 3.4.15, we get $\mathsf{InsMsgPai}(x, a, \mu, a, \mu_2) \wedge \mathsf{Ins}(x, a, a) \wedge \mathsf{Ins}(x, a, \mu_2)$. By the induction hypothesis we know we have some $\mathsf{Ins}X(x, a, \mu_2, \dots)$ such that $\mathscr{L}(\phi)\Diamond(x, a, \mu_2) = M_2[{}^{A}/_{\chi}]$. Thus by definition of $\Diamond$ we know $\mathscr{L}(\phi)\Diamond(x, a, \mu) = \mathsf{pair}(A, M_2[{}^{A}/_{\chi}])$ and clearly $\mathsf{pair}(A, M_2[{}^{A}/_{\chi}]) = \mathsf{pair}(\chi, M_2)[{}^{A}/_{\chi}]$ and the result follows.

  $\neg\mathsf{Eq}(x, \mu_1)$ and $\mathsf{Eq}(x, \mu_2)$: Similar to the case above.

  $\neg\mathsf{Eq}(x, \mu_1)$ and $\neg\mathsf{Eq}(x, \mu_2)$: For this case, by the axioms in row-group 2 of Table 3.4.15, we get $\mathsf{InsMsgPai}(x, a, \mu, \mu_1, \mu_2) \wedge \mathsf{Ins}(x, a, a) \wedge \mathsf{Ins}(x, a, \mu_1) \wedge \mathsf{Ins}(x, a, \mu_2)$. By the induction hypothesis we know we have some $\mathsf{Ins}X(x, a, \mu_1, \dots)$ such that $\mathscr{L}(\phi)\Diamond(x, a, \mu_1) = M_1[{}^{A}/_{\chi}]$ and $\mathsf{Ins}X(x, a, \mu_2, \dots)$ such that $\mathscr{L}(\phi)\Diamond(x, a, \mu_2) = M_2[{}^{A}/_{\chi}]$. Thus by definition of $\Diamond$ we know $\mathscr{L}(\phi)\Diamond(x, a, \mu) = \mathsf{pair}(M_1[{}^{A}/_{\chi}], M_2[{}^{A}/_{\chi}])$ and clearly $\mathsf{pair}(M_1[{}^{A}/_{\chi}], M_2[{}^{A}/_{\chi}]) = \mathsf{pair}(M_1, M_2)[{}^{A}/_{\chi}]$ and the result follows.

B.1.8. Proof of Items 1 to 3 of Theorem 3.2.4 for correspondence assertions.

We prove these three together, since they are very similar criteria for terms, conditions and assertions. We first prove the implication one way for all three criteria, and then the other way for all three.

If there exists a type environment Γ and type T such that $\Gamma \vdash M : T$, then $\Gamma_M \vdash M \rightsquigarrow \tau; \phi$ and $\mathscr{L}(\phi)$ is defined, $\mathscr{L}(\phi) \rhd \tau = T$, and $\forall n : \tau_n \in \Gamma_M : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$

If there exists a type environment Γ and condition σ, such that $\Gamma \vdash \sigma$, then $\Gamma_\sigma \vdash \sigma \rightsquigarrow \phi$ and $\mathscr{L}(\phi)$ is defined and $\forall n : \tau_n \in \Gamma_\sigma : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$

And if there exists a type environment Γ and condition Ψ, such that $\Gamma \vdash \Psi$, then $\Gamma_\Psi \vdash \sigma \rightsquigarrow \phi$ and $\mathscr{L}(\phi)$ is defined and $\forall n : \tau_n \in \Gamma_\Psi : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$

We make the following two observations: Firstly, *Obs. 1*, if for any two constraints ϕ_1 and ϕ_2, and for some type inference environment Γ_p we have $\forall (x : \tau) \in \Gamma_p : (x : \mathscr{L}(\phi_1) \rhd \tau) \in \Gamma$ and $\forall (x : \tau) \in \Gamma_p : (x : \mathscr{L}(\phi_2) \rhd \tau) \in \Gamma$ for some environment Γ, then $\mathscr{L}(\phi_1 \wedge \phi_2)$ must be defined and $\forall \tau \in \mathsf{ran}(\Gamma_p) : \mathscr{L}(\phi_1) \rhd \tau = \mathscr{L}(\phi_2) \rhd \tau = \mathscr{L}(\phi_1 \wedge \phi_2) \rhd \tau$. We can show this by investigation the possible cases:

- $\mathscr{L}(\phi_1) \rhd \tau = T$ **and** $\mathscr{L}(\phi_1) \rhd \tau = T$**:** Both constraints agree on the type assignment of τ and the result follows
- $\mathscr{L}(\phi_1) \rhd \tau = U$ **for any** U **and** $\mathscr{L}(\phi_1) \rhd \tau = T$**:** We can simply let $U = T$ and the result follows
- $\mathscr{L}(\phi_1) \rhd \tau = U$ **for any** U **and** $\mathscr{L}(\phi_1) \rhd \tau = T$ **for any** T**:** As U and T can be any type, we chose one such that $U = T$
- $\mathscr{L}(\phi_1) \rhd \tau = U$ **and** $\mathscr{L}(\phi_1) \rhd \tau = T$ **where** $U \neq T$**:** Not possible, as this violate the assumption of $\forall (x : \tau) \in \Gamma_p : (x : \mathscr{L}(\phi_1) \rhd \tau) \in \Gamma$ and $(x : \tau) \in \Gamma_p : (x : \mathscr{L}(\phi_2) \rhd \tau) \in \Gamma$.

Secondly, *Obs. 2*, if $\mathscr{L}(\phi) \rhd \tau = T$ for some τ and T, then by the axioms of Table 3.4.13 we know:

- $T = U$**:** $\mathscr{L}(\phi \wedge \mathsf{IsUn}(\tau)) \rhd \tau = \mathsf{Un}$
- $T = \mathsf{Un}$**:** $\mathscr{L}(\phi \wedge \mathsf{IsX?}(\tau, \dots)) \rhd \tau = \mathsf{Un}$
- $T = \mathsf{Ok}(S)$**:** $\mathscr{L}(\phi \wedge \mathsf{IsOk?}(\tau)) \rhd \tau = T$
- $T = \mathsf{Key}(U)$**:** If $\mathscr{L}(\phi) \rhd \tau' = U$ then $\mathscr{L}(\phi \wedge \mathsf{IsKey?}(\tau, \tau')) \rhd \tau = T$
- $T = \mathsf{Ch}(U)$**:** If $\mathscr{L}(\phi) \rhd \tau' = U$ then $\mathscr{L}(\phi \wedge \mathsf{IsCha?}(\tau, \tau')) \rhd \tau = T$
- $T = \mathsf{Pair}(x : U_1, U_2)$**:** If $\mathscr{L}(\phi) \rhd \tau_1 = U_1$ and $\mathscr{L}(\phi) \rhd \tau_2 = U_2$ then $\mathscr{L}(\phi \wedge \mathsf{IsPair?}(\tau, x, \tau_1, \tau_2)) \rhd \tau = T$

Finally, since $\Gamma \vdash M$, $\Gamma \vdash \sigma$, or $\Gamma \vdash \Psi$ by the case, we will let Γ_c be the type inference environment defined as $\bigcup_{(x:T) \in \Gamma} x : \tau_x$ where every τ_x is unique, and prove the cases for $\Gamma' = \Gamma_c$. We use this to prove the theorem by induction in M, σ, and Ψ. We also structure each case into three sub-cases corresponding to the three statement that must be proven.

Nam: The result follows directly from the definition of Γ_c and $\rhd$.

Enc: We consider each of the two relevant typing rules

Not Un: For this case we know from the induction hypothesis that $\Gamma_c \vdash M \rightsquigarrow \tau_m; \phi_m$, $\mathscr{L}(\phi_m)$ is defined, $\mathscr{L}(\phi_m) \rhd \tau_m = T$, $\forall (x : \tau) \in \Gamma_c : (x : \mathscr{L}(\phi_m) \rhd \tau) \in \Gamma$, $\Gamma_c \vdash N \rightsquigarrow \tau_n; \phi_n$, $\mathscr{L}(\phi_n)$ is defined, $\mathscr{L}(\phi_n) \rhd \tau_n = \mathsf{Key}(T)$, and $\forall (x : \tau) \in \Gamma_c : (x : \mathscr{L}(\phi_n) \rhd \tau) \in \Gamma$. We now prove each of the three statements:

(1) We first conclude $\mathscr{L}(\phi_m \wedge \phi_n)$ is defined. This follows directly from *Obs. 1*. We now proceed to include each of the generated constraints, and argue that for each conjunction the resulting constraint still has a well-defined solution.
For $\phi_m \wedge \phi_n \wedge \mathsf{IsKey?}(\tau_n, \tau_m)$ we can conclude that since $\mathscr{L}(\phi_n \wedge \phi_m) \rhd \tau_n = \mathsf{Key}(T)$ then either we have $\mathsf{IsKey}(\tau_n, \tau'_m)$ in $\phi_n \wedge \phi_m$ for which $\mathscr{L}(\phi_n \wedge \phi_m) \rhd \tau'_m = T$ or we have no $\mathsf{IsX}(\tau_n, \dots)$ in $\phi_n \wedge \phi_m$. In the former case, by the axioms of row-group 9 of Table 3.4.13, we deduce we must have some $\mathsf{IsKey?}(\tau_n, \tau'_m)$ constraint in $\phi_n \wedge \phi_m$ for which $\mathscr{L}(\phi_n \wedge \phi_m) \rhd \tau'_m = T$. Thus by the axioms in row-group 4 of the same table we know this implies $\mathsf{Eq}(\tau_m, \tau'_m)$. However as $\mathscr{L}(\phi_n \wedge \phi_m) \rhd \tau_m = T$ and $\mathscr{L}(\phi_n \wedge \phi_m) \rhd \tau'_m = T$ we can conclude

the constraint $\mathsf{Eq}(\tau_m, \tau'_m)$ is satisfied and thus since $\tau'_m = \tau_m$, and by extension $T = T$, we can conclude that $\mathscr{L}(\phi_m \wedge \phi_n \wedge \mathsf{IsKey?}(\tau_n, \tau_m))$ must be defined. In the latter case, where we have no $\mathsf{IsX}(\tau_n, \dots)$ constraint in $\phi_n \wedge \phi_m$ we can conclude that $\mathscr{L}(\phi_m \wedge \phi_n \wedge \mathsf{IsKey?}(\tau_n, \tau_m))$ must be defined by the axioms of row-group 9 of Table 3.4.13 such that $\mathscr{L}(\phi_m \wedge \phi_n \wedge \mathsf{IsKey}(\tau_n, \tau_m)) \rhd \tau_n = \mathsf{Key}(\mathscr{L}(\phi_m \wedge \phi_n \wedge \mathsf{IsKey}(\tau_n, \tau_m)) \rhd \tau_m) = \mathsf{Key}(T)$.
For $\phi_m \wedge \phi_n \wedge \mathsf{IsKey?}(\tau_n, \tau_m) \wedge \mathsf{IsUn}(\tau)$ we trivially conclude $\mathscr{L}(\phi_m \wedge \phi_n \wedge \mathsf{IsKey?}(\tau_n, \tau_m) \wedge \mathsf{IsUn}(\tau))$ is defined, since $\tau \notin \mathsf{names}(\phi_n \wedge \phi_m)$ implying $\mathscr{L}(\phi_m \wedge \phi_n \wedge \mathsf{IsKey?}(\tau_n, \tau_m)) \rhd \tau = U$ for any U, thus allowing $\mathscr{L}(\phi_m \wedge \phi_n \wedge \mathsf{IsKey?}(\tau_n, \tau_m) \wedge \mathsf{IsUn}(\tau)) \rhd \tau = \mathsf{Un}$.

(2) Follows directly from the above, the constraint $\mathsf{IsUn}(\tau)$, and *Obs. 2*

(3) Since $\mathscr{L}(\phi_m) \rhd \tau_m = T$ and $\mathscr{L}(\phi_n) \rhd \tau_n = \mathsf{Key}(T)$, we know by *Obs. 1* and *Obs. 2* that $\mathscr{L}(\phi_m \wedge \phi_n \wedge \mathsf{IsKey?}(\tau_n, \tau_m) \wedge \mathsf{IsUn}(\tau)) \rhd \tau_n = \mathsf{Key}(T)$, $\mathscr{L}(\phi_m \wedge \phi_n \wedge \mathsf{IsKey?}(\tau_n, \tau_m) \wedge \mathsf{IsUn}(\tau)) \rhd \tau_m = T$, and $\mathscr{L}(\phi_m \wedge \phi_n \wedge \mathsf{IsKey?}(\tau_n, \tau_m) \wedge \mathsf{IsUn}(\tau)) \rhd \tau = \mathsf{Un}$. The remaining type variables follows from *Obs. 1*

Un: Similar to the case above

Pai: **Not Un:** For this case we know from the induction hypothesis that $\Gamma_c \vdash M \rightsquigarrow \tau_m; \phi_m$, $\mathscr{L}(\phi_m)$ is defined, $\mathscr{L}(\phi_m) \rhd \tau_m = T$, $\forall (x : \tau) \in \Gamma_c : (x : \mathscr{L}(\phi_m) \rhd \tau) \in \Gamma$, $\Gamma_c \vdash N \rightsquigarrow \tau_n; \phi_n$, $\mathscr{L}(\phi_n)$ is defined, $\mathscr{L}(\phi_n) \rhd \tau_n = U(M)$, and $\forall (x : \tau) \in \Gamma_c : (x : \mathscr{L}(\phi_n) \rhd \tau) \in \Gamma$. We also know from Lemma 3.4.5 that $\mathscr{L}(\psi_m \wedge \varphi_m) \lhd \mu = M$. We have limited $U(M)$ to being either $\mathsf{Ok}(S)$ or Un. We also know from Lemma 3.4.6 that if $U(M) = \mathsf{Ok}(S)$ then $\mathscr{L}(\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n \wedge \mathsf{NamVar}(x) \wedge \mathsf{PairVar}(x) \wedge \mathsf{IsPair?}(\tau, x, \tau_m, \tau'_n) \wedge \mathsf{IsAbs?}(\tau'_n, \tau_n, \mu, x, \tau_m)) \rhd \tau'_n = \mathsf{Ok}(S[\not{\chi}_M]) = U(\chi)$. And the axiom $\forall t : \forall x : \forall t_f : \forall t'_s : \forall t_s : \forall a : \forall t''_s : (\mathsf{IsPair}(t, x, t_f, t'_s) \wedge \mathsf{IsAbs}(t'_s, t_s, a, x, t''_s) \wedge \mathsf{IsUn}(t_s)) \Rightarrow \mathsf{IsUn}(t'_s)$ says that if $U(M) = \mathsf{Un}$ then $\mathscr{L}(\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n \wedge \mathsf{NamVar}(x) \wedge \mathsf{PairVar}(x) \wedge \mathsf{IsPair?}(\tau, x, \tau_m, \tau'_n) \wedge \mathsf{IsAbs?}(\tau'_n, \tau_n, \mu, x, \tau_m)) \rhd \tau'_n = \mathsf{Un}$

(1) From *Obs. 1* we get that $\mathscr{L}(\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n)$ is defined. Since x is fresh for $\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n$, adding $\mathsf{NamVar}(x)$ and $\mathsf{PairVar}(x)$ cannot prevent a solution from being found. Since τ and τ'_n are fresh for $\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n \wedge \mathsf{NamVar}(x) \wedge \mathsf{PairVar}(x)$ there is no reason we cannot create a solution such that $\mathscr{L}(\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n \wedge \mathsf{NamVar}(x) \wedge \mathsf{PairVar}(x)) \rhd \tau = \mathsf{Pair}(x : \mathscr{L}\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n \wedge \mathsf{NamVar}(x)) \rhd \tau_m, \mathscr{L}(\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n \wedge \mathsf{NamVar}(x) \wedge \mathsf{PairVar}(x)) \rhd \tau'_n$ and $\mathscr{L}(\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n \wedge \mathsf{NamVar}(x) \wedge \mathsf{PairVar}(x)) \rhd \tau'_n = \mathscr{L}(\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n \wedge \mathsf{NamVar}(x) \wedge \mathsf{PairVar}(x)) \rhd \tau_n[\not{\chi}_{\mathscr{L}(\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n \wedge \mathsf{NamVar}(x) \wedge \mathsf{PairVar}(x)) \rhd \mu}]$ making this a solution $\mathscr{L}(\phi_m \wedge \psi_m \wedge \varphi_m \wedge \phi_n \wedge \mathsf{NamVar}(x) \wedge \mathsf{PairVar}(x) \wedge \mathsf{IsPair?}(\tau, x, \tau_m, \tau'_n) \wedge \mathsf{IsAbs?}(\tau'_n, \tau_n, \mu, x, \tau_m))$.

(2) Since $\mathsf{IsPair?}(\tau, x, \tau_m, \tau'_n) \in \phi$ and τ is fresh, $\mathscr{L}(\phi) \rhd \tau = \mathsf{Pair}(x : \mathscr{L}(\phi) \rhd \tau_m, \mathscr{L}(\phi) \rhd \tau'_n)$, and as shown above, $\mathscr{L}(\phi) \rhd \tau_m = T$ and $\mathscr{L}(\phi) \rhd \tau'_n = U(\chi)$

(3) Follows directly from *Obs. 1* and the fact that τ is fresh

Un: Similar to the case above

Ok: We consider each of the two relevant typing rules

Not Un: For this case we know $\Gamma_c \vdash \diamond$, and since $\Psi_i \twoheadrightarrow \xi_i; \psi_i; \varphi_i$, we get from Item 4 that $\mathscr{L}(\psi_i \wedge \varphi_i) \lhd \xi_i = \Psi_i$ for each i.

(1) Since ψ_i and φ_i are message encodings, and have no influence on the satisfiability of the other constraints and since generating more $\mathsf{CanOk?}(\tau, \xi')$ and $\mathsf{InsCanOk}(\tau, x, \mu, \xi')$ prefixes cannot possibly lead to $\mathsf{FAIL}(\xi)$ prefixes, we can conclude by *Obs. 1* and the fact that τ is fresh for $\bigwedge_i(\psi_i \wedge \varphi_i)$, that $\mathscr{L}(\mathsf{IsOk?}(\tau) \wedge \bigwedge_i(\mathsf{CanOk?}(\tau, \xi_i) \wedge \psi_i \wedge$

$\phi_i) \wedge \bigwedge_{\tau' \in \mathsf{rng}(\Gamma)}(\forall \xi' : (\mathsf{IsOk}(\tau') \wedge \mathsf{CanOk}(\tau', \xi')) \Rightarrow \mathsf{CanOk?}(\tau, \xi')) \wedge \bigwedge_{\tau' \in \mathsf{rng}(\Gamma)}(\forall \xi' : \forall x : \forall \mu : (\mathsf{IsOk}(\tau') \wedge \mathsf{InsCanOk}(\tau', x, \mu, \xi') \wedge \mathsf{IsOk}(\tau)) \Rightarrow \mathsf{InsCanOk}(\tau, x, \mu, \xi')))$ is defined

(2) Since τ is a fresh type variable, and since the constraint generation rules for effects do not generate any IsX? constraints, we know that $\mathsf{IsOk?}(\tau)$, $\mathsf{CanOk?}(\tau, \xi_i)$, and $\mathsf{InsCanOk}(\tau, x, \mu, \xi')$, by the axioms $\forall t : \forall u_1 : \forall u_2 : \forall \mu : \forall t_f : \forall t_s : (\mathsf{IsOk?}(t) \wedge \neg\mathsf{IsCha?}(t, u_1) \wedge \neg\mathsf{IsKey?}(t, u_2) \wedge \neg\mathsf{IsPair?}(t, \mu, t_f, t_s) \wedge \neg\mathsf{IsUn}(t)) \Rightarrow \mathsf{IsOk}(t)$ and $\forall t : \forall \xi : (\mathsf{IsOk}(t) \wedge \mathsf{CanOk?}(t, \xi)) \Rightarrow \mathsf{CanOk}(t, \xi)$, will give rise to $\mathsf{IsOk}(\tau)$, $\mathsf{CanOk}(\tau, \xi_i)$, and $\mathsf{InsCanOk}(\tau, x, \mu, \xi')$ in $\mathscr{L}(\bigwedge_i(\mathsf{CanOk?}(\tau, \xi_i) \wedge \phi_i \wedge \psi_i \wedge \varphi_i) \wedge \mathsf{IsOk?}(\tau))$, and thus the derivation $\mathscr{L}(\bigwedge_i(\mathsf{CanOk?}(\tau, \xi_i) \wedge \phi_i \wedge \psi_i \wedge \varphi_i) \wedge \mathsf{IsOk?}(\tau)) \rhd \tau = \mathsf{Ok}(\bigcup_i \ell_i(M_i))$ where $S \subseteq \bigcup_i \ell_i(M_i)$.

(3) Follows directly from *Obs. 1* and the fact that τ is fresh

Un: Similar to the case above

Fst: We consider each of the two relevant typing rules

Not Un: For this case we know $\Gamma_c \vdash M \rightsquigarrow \tau_m \phi_m$, $\mathscr{L}(\phi_m)$ is defined, $\mathscr{L}(\phi_m) \rhd \tau_m = \mathsf{Pair}(T, U)$, and $\forall (x : \tau) \in \Gamma_c : (x : \mathscr{L}(\phi_m) \rhd \tau) \in \Gamma$.

(1) similar to case **Pai**

(2) By the definition of $\rhd$, as seen in Table 3.4.16, we know that $\mathscr{L}(\phi_m) \rhd \tau_m = \mathsf{Pair}(T, U)$ is derived from either (1) no IsX-constraint exists in $\mathscr{L}(\phi_m)$ for τ_m or because there exists a $\mathsf{IsPair}(\tau_m, \mu_m, \tau_m^1, \tau_m^2)$ constraint in $\mathscr{L}(\phi_m)$. In the case where $\mathsf{IsPair}(\tau_m, \mu_m, \tau_m^1, \tau_m^2) \in \mathscr{L}(\phi_m)$ we get $\mathsf{Eq}(\tau, \tau_1^m)$ by the axioms of table Table 3.4.13, implying $\mathscr{L}(\phi_m \wedge \mathsf{IsPair?}(\tau_m, \mu, \tau, \tau_2)) \rhd \tau = T$, and for the case with no $\mathsf{IsPair}(\dots)$-constraint, by definition of $\rhd$, we can trivially make the derivation $\mathscr{L}(\phi_m \wedge \mathsf{IsPair?}(\tau_m, \mu, \tau, \tau_2)) \rhd \tau = T$.

(3) Follows directly from the fact τ and τ_2 are fresh, *Obs. 1*, and *Obs. 2*

Un: Similar to the case above

Snd: As we have two rules for this case, we consider both:

Not Un: We know that $\Gamma_c \vdash M \rightsquigarrow \tau_m; \phi_m$, $\mathscr{L}(\phi_m)$ is defined, $\mathscr{L}(\phi_m) \rhd \tau_m = \mathsf{pair}(\chi : T, T'(\chi))$, $M_1 \twoheadrightarrow \mu; \psi; \varphi$, and $\forall (x : \tau) \in \Gamma_c : (x : \mathscr{L}(\phi_m) \rhd \tau) \in \Gamma$. Since $\mathscr{L}(\phi_m) \rhd \tau_m = \mathsf{pair}(\chi : T, T'(\chi))$, we know that there exists some τ_m^2, τ_m^3 such that $\mathsf{IsPair}(\tau_m, x, \tau_m^2, \tau_m^3)$. This implies $\mathscr{L}(\phi_m \wedge \psi \wedge \varphi \wedge \mathsf{IsPair?}(\tau_m, \mu, \tau', \tau))$ is defined. By the axioms in Table 3.4.13, in particular row-group 5, we can deduce $\mathsf{Eq}(\tau', \tau_m^2)$, and since $\mathscr{L}(\phi_m) \rhd \tau_m^2 = U$ by Lemma 3.4.7 and $\forall t : \forall x : \forall t_f : \forall t_s' : \forall t_s : \forall a : \forall t_s'' : (\mathsf{IsPair}(t, x, t_f, t_s') \wedge \mathsf{IsAbs}(t_s', t_s, a, x, t_s'') \wedge \mathsf{IsUn}(t_s)) \Rightarrow \mathsf{IsUn}(t_s')$ we can conclude that $\mathscr{L}(\phi) \rhd \tau = \mathscr{L}(\phi) \rhd \tau'[{}^{\mathsf{snd}\ M}/_{\chi}]$, and so by the induction hypothesis we can conclude $\mathscr{L}(\phi_m \wedge \psi \wedge \varphi \wedge \mathsf{IsPair?}(\tau_m, \mu, \tau', \tau)) \rhd \tau = T'(\mathsf{snd}\ M)$ and the result follows.

Un: similar to above

Eq: Trivial

Neq: Trivial

Seq: For this case we have $\Gamma_c \vdash M \rightsquigarrow \tau_m; \phi_m$, $\mathscr{L}(\phi_m)$ is defined, $\mathscr{L}(\phi_m) \rhd \tau_m = T$, $\forall (x : \tau) \in \Gamma_c : (x : \mathscr{L}(\phi_m) \rhd \tau) \in \Gamma$, $\Gamma_c \vdash N \rightsquigarrow \tau_n; \phi_n$, $\mathscr{L}(\phi_n)$ is defined, $\mathscr{L}(\phi_n) \rhd \tau_n = T$, and $\forall (x : \tau) \in \Gamma_c : (x : \mathscr{L}(\phi_m) \rhd \tau) \in \Gamma$.

(1) Follows directly from *Obs. 1*, $\mathscr{L}(\phi_m) \rhd \tau_m = T$, and $\mathscr{L}(\phi_n) \rhd \tau_n = T$

(2) Follows directly from the above case and *Obs. 1*

Eeq: Trivial

One: Trivial

Dcr: Similar to the case for [Enc-C]

Bgn: Trivial

End: For this case must consider two sub-cases as defined by [End-C]:

$\ell(M) \in \Gamma_c$**:** For this case we know $\ell(M) \in \Gamma_c$.

(1) Since $\Gamma \vdash \diamond$ we know that $\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma)$ implying $\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma_c)$ and the result follows
(2) Since $[\![\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma_c)]\!]$ is the only constraint, then by definition of $\rhd$ we have $\mathscr{L}(\phi) \rhd \tau = T$ for any T implying $\forall n : \tau_n \in \Gamma_c : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$

Otherwise: For this case we know $\ell(M) \notin \Gamma_c$.

(1) Since $\Gamma \vdash \diamond$ we know that $\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma)$ implying $\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma_c)$. Assuming the process initially was well-formed, we know $\Gamma = \Gamma_1, x : \mathsf{Ok}(S), \Gamma_2$ for which $\ell(M) \in S$, we must have $x : \tau_x \in \Gamma_c$ for which $\mathscr{L}(\phi + \epsilon) \rhd \tau_x = \mathsf{Ok}(S)$. We let ϵ represent the constraint generated further down the constraint generation tree. In this derivation we utilise $\mathsf{CanOk}(\tau_x, \xi)$ and $\mathsf{InsCanOk}(\tau, \chi, \mu, \xi)$ constraints for which there exists one such that either $\mathscr{L}(\phi+\epsilon) \lhd \xi = \ell(M)$ or $\mathscr{L}(\phi+\epsilon) \Diamond (\chi, \mu, \xi) = \ell(M)$. In both cases the constraint generated by [End-C] does not imply $\mathsf{Fail}(\xi)$.
(2) Since $[\![\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma_c)]\!]$ is the only constraint, then by definition of $\rhd$ we have $\mathscr{L}(\phi) \rhd \tau = T$ for any T implying $\forall n : \tau_n \in \Gamma_c : n : (\mathscr{L}(\phi) \rhd \tau_n) \in \Gamma$

If $\Gamma \vdash M \rightsquigarrow \tau; \phi$ **such that** $\mathscr{L}(\phi)$ **is defined and** $\Gamma' = \mathsf{assertions}(\Gamma) \cup \{x : \mathscr{L}(\phi) \rhd \tau \mid x : \tau \in \Gamma\}$ **then** $\Gamma' \vdash M : \mathscr{L}(\phi) \rhd \tau$

If $\Gamma \vdash \sigma \rightsquigarrow \phi$ **such that** $\mathscr{L}(\phi)$ **is defined and** $\Gamma' = \mathsf{assertions}(\Gamma) \cup \{x : \mathscr{L}(\phi) \rhd \tau \mid x : \tau \in \Gamma\}$ **then** $\Gamma' \vdash \sigma$

And if $\Gamma \vdash \Psi \rightsquigarrow \phi$ **such that** $\mathscr{L}(\phi)$ **is defined and** $\Gamma' = \mathsf{assertions}(\Gamma) \cup \{x : \mathscr{L}(\phi) \rhd \tau \mid x : \tau \in \Gamma\}$ **then** $\Gamma' \vdash \Psi$

We prove this through induction in the constraint generation rules for messages, conditions, and assertions:

[Nam-C]: Here we have that $M = a$ and $\phi = \mathtt{T}$. From the rule we get $\Gamma \vdash \diamond$ and $\Gamma(a) = \tau$. It obviously follows from this that $\Gamma' \vdash \diamond$ and $\Gamma'(a) = \mathscr{L}(\mathtt{T}) \rhd \tau$, and therefore, according to [Nam], $\Gamma' \vdash a : (\mathscr{L}(\mathtt{T}) \rhd \tau)$ for any assignment of $\mathscr{L}(\mathtt{T}) \rhd \tau$.

[Enc-C]: Here we have that $M = \{M'\}_N$, $\Gamma \vdash M' \rightsquigarrow \tau_{M'}; \phi_{M'}$, $\Gamma \vdash N \rightsquigarrow \tau_N; \phi_N$, and $\phi = \phi'_M \wedge \phi_N \wedge \mathsf{IsUn}(\tau) \wedge \mathsf{IsKey?}(\tau_N, \tau_{M'})$. From the induction hypothesis we get that, since $\phi \Rightarrow \phi_{M'}$ and $\phi \Rightarrow \phi_N$, meaning that $\mathscr{L}(\phi)$ is also a solution to $\phi_{M'}$ and ϕ_N, and thus $\Gamma' \vdash M' : (\mathscr{L}(\phi) \rhd \tau_{M'})$ and $\Gamma' \vdash N : (\mathscr{L}(\phi) \rhd \tau_N)$. We now consider the two remaining constraints:

- Since $\phi \Rightarrow \mathsf{IsUn}(\tau)$ we know $\mathscr{L}(\phi) \rhd \tau = \mathsf{Un}$, as is required in both [Enc] and [Enc-Un].
- Since $\phi \Rightarrow \mathsf{IsKey?}(\tau_N, \tau_{M'})$, we need to consider the following axioms:

$$\forall t : \forall u : (\mathsf{IsOk?}(t) \wedge \mathsf{IsKey?}(t, u)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u))$$

$$\forall t : \forall u_1 : \forall u_2 : (\mathsf{IsCha?}(t, u_1) \wedge \mathsf{IsKey?}(t, u_2)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u_1) \wedge \mathsf{IsUn}(u_2))$$

$$\forall t : \forall u : \forall \mu : \forall t_f : \forall t_s : (\mathsf{IsKey?}(t, u) \wedge \mathsf{IsPair?}(t, x, t_f, t_s)) \Rightarrow$$
$$(\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u) \wedge \mathsf{IsUn}(t_f) \wedge \mathsf{IsUn}(t_s))$$

$$\forall t : \forall x : \forall u_1 : \forall u_2 : \forall t_f : \forall t_s : (\neg\mathsf{IsOk?}(t) \wedge \neg\mathsf{IsCha?}(t, u_1) \wedge \mathsf{IsKey?}(t, u_2) \wedge$$
$$\neg\mathsf{IsPair?}(t, x, t_f, t_s) \wedge \neg\mathsf{IsUn}(t)) \Rightarrow \mathsf{IsKey}(t, u_2)$$

Together they mean that either $\mathscr{L}(\phi) \rhd \tau_{M'} = \mathscr{L}(\phi) \rhd \tau_N = \mathsf{Un}$ or $\mathscr{L}(\phi) \rhd \tau_N = \mathsf{Key}(\mathscr{L}(\phi) \rhd \tau_{M'})$ corresponding to [Enc-Un] and [Enc] respectively.

[Pai-C]: Here we have that $M = \mathsf{pair}(M_1, M_2)$, $\Gamma \vdash M_1 \rightsquigarrow \tau_1; \phi_1$, $M_1 \twoheadrightarrow \mu; \psi_1, \varphi_1$, $\Gamma \vdash M_2 \rightsquigarrow \tau_2; \phi_2$. Moreover we generate $\mathsf{NamVar}(x)$, $\mathsf{PairVar}(x)$, $\mathsf{IsPair?}(\tau, x, \tau_1, \tau'_2)$, and $\mathsf{IsAbs?}(\tau'_2, \tau_2, \mu, x, \tau_1)$.

By the induction hypothesis we have that, since $\phi \Rightarrow \phi_1$ and $\phi \Rightarrow \phi_2$, meaning that $\mathscr{L}(\phi)$ is also a solution to ϕ_1 and ϕ_2, and thus $\Gamma' \vdash M_1 : (\mathscr{L}(\phi) \rhd \tau_1)$ and $\Gamma' \vdash M_2 : (\mathscr{L}(\phi) \rhd \tau_2)$.

Since $\phi \Rightarrow \mathsf{IsPair?}(\tau, \mu, \tau_1, \tau_2)$ we look at the following axioms:

$\forall t : \forall x : \forall t_f : \forall t_s : (\mathsf{IsOk?}(t) \wedge \mathsf{IsPair?}(t, x, t_f, t_s)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(t_f) \wedge \mathsf{IsUn}(t_s))$
$\forall t : \forall u : \forall x : \forall t_f : \forall t_s : (\mathsf{IsCha?}(t, u) \wedge \mathsf{IsPair?}(t, x, t_f, t_s)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u) \wedge$
$\mathsf{IsUn}(t_f) \wedge \mathsf{IsUn}(t_s))$
$\forall t : \forall u : \forall x : \forall t_f : \forall t_s : (\mathsf{IsKey?}(t, u) \wedge \mathsf{IsPair?}(t, x, t_f, t_s)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u) \wedge$
$\mathsf{IsUn}(t_f) \wedge \mathsf{IsUn}(t_s))$
$\forall t : \forall x : \forall u_1 : \forall u_2 : \forall t_f : \forall t_s : (\neg\mathsf{IsOk?}(t) \wedge \neg\mathsf{IsCha?}(t, u_1) \wedge \neg\mathsf{IsKey?}(t, u_2) \wedge$
$\mathsf{IsPair?}(t, x, t_f, t_s) \wedge \mathsf{IsAbs?}(t_s, t'_s, a, x, t_f) \wedge \neg\mathsf{IsUn}(t)) \Rightarrow$
$(\mathsf{IsPair}(t, x, t_f, t_s) \wedge \mathsf{IsAbs}(t_s, t'_s, a, x, t_f))$

Together these constraints imply that either $\mathscr{L}(\phi) \rhd \tau = \mathscr{L}(\phi) \rhd \tau_1 = \mathscr{L}(\phi) \rhd \tau_2 = \mathsf{Un}$ or $\mathscr{L}(\phi) \rhd \tau = \mathsf{Pair}(\chi : \mathscr{L}(\phi) \rhd \tau_1, \mathscr{L}(\phi) \rhd \tau'_2)$, corresponding to rule [Pai-Un] and [Pai] respectively. By row-group 6 from Table 3.4.13 and by Lemma 3.4.6 we know that $\mathscr{L}(\phi) \rhd \tau'_2 = \mathscr{L}(\phi) \rhd \tau_2[\not{\chi}_M]$, and the conditions from [Pai] are satisfied.

[Ok-C]: Here we have that $M = \mathsf{ok}$, $\Gamma \vdash \diamond$, for all $\Psi_i \in \Gamma : \ \Psi_i \twoheadrightarrow \xi_i : \psi_i, \phi_i$, and $\phi = \mathsf{IsOk?}(\tau) \wedge \mathsf{IsOkTerm}(\tau) \wedge \bigwedge_i(\mathsf{CanOk?}(\tau, \xi_i) \wedge \psi_i \wedge \phi_i) \wedge \bigwedge_{\tau' \in \mathsf{rng}(\Gamma)}(\forall \xi' :$ $(\mathsf{IsOk}(\tau') \wedge \mathsf{CanOk}(\tau', \xi')) \Rightarrow \mathsf{CanOk?}(\tau, \xi')) \wedge \bigwedge_{\tau' \in \mathsf{rng}(\Gamma)}(\forall \xi' : \forall x : \forall \mu : (\mathsf{IsOk}(\tau') \wedge$ $\mathsf{InsCanOk}(\tau', x, \mu, \xi') \wedge \mathsf{IsOk}(\tau)) \Rightarrow \mathsf{InsCanOk?}(\tau, x, \mu, \xi'))$.

Since, $\Gamma \vdash \diamond$, obviously $\Gamma' \vdash \diamond$, as required in both [Ok] and [Ok-Un]. By row-group 7 and 9 of Table 3.4.13 we know that either $\mathscr{L}(\phi) \rhd \tau = \mathsf{Un}$ or $\mathscr{L}(\phi) \rhd \tau = \mathsf{Ok}(S)$ for some S, corresponding to either [Ok-Un] or [Ok]. By ϕ we know that we generate a $\mathsf{CanOk?}(\tau, \xi)$ or $\mathsf{InsCanOk?}(\tau, x, \mu, \xi)$ for all possible effects that might be in S. By row-group 10 of Table 3.4.13 we know that we only convert $\mathsf{CanOk?}(\tau, \xi)$ and $\mathsf{InsCanOk?}(\tau, x, \mu, \xi)$ to the definitive versions if all possible senders of a certain channel agree of the effect ξ. Thus if we have $\mathsf{CanOk}(\tau, \xi)$ or $\mathsf{InsCanOk}(\tau, x, \mu, \xi)$ we know that it is safe to include $\mathscr{L}(\phi) \lhd \xi$ or $\mathscr{L}(\phi) \Diamond (x, \mu, \xi)$ to S, and thus $\Gamma' \vdash M : (\mathscr{L}(\phi) \rhd \tau)$.

[Fst-C]: Here we have that $M = \mathsf{fst}\ M$, $\Gamma \vdash M \rightsquigarrow \tau_M; \phi_M$, and $\phi = \phi_M \wedge \mathsf{IsPair?}(\tau_M, x, \tau, \tau'_2) \wedge \mathsf{NamVar}(x) \wedge \mathsf{IsAbs?}(\tau'_2, \tau_2, \mu, x, \tau) \wedge \mathsf{PairVar}(x)$.

By the induction hypothesis we have that, since $\phi \Rightarrow \phi_M$ meaning that $\mathscr{L}(\phi)$ is also a solution to ϕ_M, and thus $\Gamma' \vdash M : (\mathscr{L}(\phi) \rhd \tau_M)$.

Since $\phi \Rightarrow \mathsf{IsPair?}(\tau_M, x, \tau, \tau_2)$ we look at the following axioms:

$\forall t : \forall x : \forall t_f : \forall t_s : (\mathsf{IsOk?}(t) \wedge \mathsf{IsPair?}(t, x, t_f, t_s)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(t_f) \wedge \mathsf{IsUn}(t_s))$
$\forall t : \forall u : \forall x : \forall t_f : \forall t_s : (\mathsf{IsCha?}(t, u) \wedge \mathsf{IsPair?}(t, x, t_f, t_s)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u) \wedge$
$\mathsf{IsUn}(t_f) \wedge \mathsf{IsUn}(t_s))$
$\forall t : \forall u : \forall x : \forall t_f : \forall t_s : (\mathsf{IsKey?}(t, u) \wedge \mathsf{IsPair?}(t, x, t_f, t_s)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u) \wedge$
$\mathsf{IsUn}(t_f) \wedge \mathsf{IsUn}(t_s))$
$\forall t : \forall x : \forall u_1 : \forall u_2 : \forall t_f : \forall t_s : (\neg\mathsf{IsOk?}(t) \wedge \neg\mathsf{IsCha?}(t, u_1) \wedge \neg\mathsf{IsKey?}(t, u_2) \wedge$
$\mathsf{IsPair?}(t, x, t_f, t_s) \wedge \mathsf{IsAbs?}(t_s, t'_s, a, x, t_f) \wedge \neg\mathsf{IsUn}(t)) \Rightarrow$
$(\mathsf{IsPair}(t, x, t_f, t_s) \wedge \mathsf{IsAbs}(t_s, t'_s, a, x, t_f))$

Together they mean that either $\mathscr{L}(\phi) \rhd \tau_M = \mathscr{L}(\phi) \rhd \tau = \mathscr{L}(\phi) \rhd \tau_2 = \mathsf{Un}$ or $\mathscr{L}(\phi) \rhd \tau_M = \mathsf{Pair}(\chi : \mathscr{L}(\phi) \rhd \tau, \mathscr{L}(\phi) \rhd \tau'_2)$, corresponding to rule [Fst-Un] and [Fst] respectively.

[Snd-C]: Here we have that $M = \mathsf{snd}\ N$, $\Gamma \vdash N \rightsquigarrow \tau_n; \phi_n$, $\mathsf{fst}\ N \twoheadrightarrow \mu; \psi_1; \phi_1$, and $\phi = \phi_m \wedge \psi_1 \wedge \phi_1 \wedge \mathsf{IsPair?}(\tau_n, x, \tau_1, \tau') \wedge \mathsf{IsAbs?}(\tau', \tau, \mu, x, \tau_1) \wedge \mathsf{NamVar}(x) \wedge \mathsf{PairVar}(x)$.

By the induction hypothesis, since $\phi \Rightarrow \phi_m$ and $\mathscr{L}(\phi)$ is also a solution to ϕ_m, and thus $\Gamma' \vdash N : (\mathscr{L}(\phi) \rhd \tau_n)$. By row-group 7 and 9 of Table 3.4.13 we know that either $\mathscr{L}(\phi) \rhd \tau_n = \mathsf{Un}$ or $\mathscr{L}(\phi) \rhd \tau_n = \mathsf{Pair}(\chi : \mathscr{L}(\phi) \rhd \tau_1, \mathscr{L}(\phi) \rhd \tau')$. If $\mathscr{L}(\phi) \rhd \tau_n = \mathsf{Un}$ then by the same axioms we have $\mathscr{L}(\phi) \rhd \tau = \mathsf{Un}$ corresponding to [Snd-Un]. If however $\mathscr{L}(\phi) \rhd \tau_n = \mathsf{Pair}(\chi : \mathscr{L}(\phi) \rhd \tau_1, \mathscr{L}(\phi) \rhd' \tau')$, then we know $N = \mathsf{pair}(N_1, N_2)$—by row-group 9—and by a similar argumentation as in

the case for [Pai-C], we know $\mathscr{L}(\phi) \rhd \tau' = \mathscr{L}(\phi) \rhd \tau[{}^{\chi}/_{N_1}]$. By the restrictions made to processes, and by row-group 6 we can deduce either $\mathscr{L}(\phi) \rhd' \tau' = \mathsf{Un}$ or $\mathscr{L}(\phi) \rhd' \tau' = \mathsf{Ok}(S)$ for some S. In the former case, namely Un, the result follows immediately as $\mathsf{Un} = \mathsf{Un}[{}^{a}/_{b}]$ for any a and b, corresponding to [Snd-Un]. For the case of $\mathsf{Ok}(S)$ we refer to the axioms in row-group 1 of Table 3.4.14, which by Lemma 3.4.7 and Lemma 3.4.5 ensures that $\mathscr{L}(\phi) \rhd \tau = \mathscr{L}(\phi) \rhd' \tau[{}^{\mathsf{fst}\ N}/_{\chi}]$ which corresponds to [Snd].

[Eq-C]: Trivial

[Neq-C]: Trivial

[Seq-C]: Here we have that $\sigma = M \text{ as } N$, $\Gamma \vdash M \rightsquigarrow \tau_m; \phi_m$, $\Gamma \vdash N \rightsquigarrow \tau_n; \phi_n$, and $\phi = \phi_m \wedge \phi_n \wedge \mathsf{Eq}(\tau_m, \tau_n)$.

By Item 1 of of Theorem 3.2.4, since $\phi \Rightarrow \phi_m$ and $\phi \Rightarrow \phi_n$ and $\mathscr{L}(\phi)$ is also a solution for both of them, we thus have $\Gamma' \vdash M : (\mathscr{L}(\phi) \rhd \tau_m)$ and $\Gamma' \vdash N : (\mathscr{L}(\phi) \rhd \tau_n)$. According to Lemma 3.4.8, since $\phi \Rightarrow \mathsf{Eq}(\tau_m, \tau_n)$, we know $\mathscr{L}(\phi) \rhd \tau_m = \mathscr{L}(\phi) \rhd \tau_n$, and the result follows.

[Eeq-C]: Trivial

[One-C]: Trivial

[Dcr-C]: Here we have that $\Psi = `x = \mathsf{dec}(M_1, M_2)'$, $\Gamma \vdash x \rightsquigarrow \tau_x; \phi_x$, $\Gamma \vdash M_1 \rightsquigarrow \tau_{m1}; \phi_{m1}$, $\Gamma \vdash M_2 \rightsquigarrow \tau_{m2}; \phi_{m2}$, and $\phi = \mathsf{IsKey?}(\tau_{m2}, \tau_x) \wedge \mathsf{IsUn}(\tau_{m1}) \wedge \phi_x \wedge \phi_{m1} \wedge \phi_{m2}$.

By Item 1 of Theorem 3.2.4, since $\phi \Rightarrow \phi_x$, $\phi \Rightarrow \phi_{m1}$, and $\phi \Rightarrow \phi_{m2}$ and $\mathscr{L}(\phi)$ is a solution to those constraints, $\Gamma' \vdash x : (\mathscr{L}(\phi) \rhd \tau_x)$, $\Gamma' \vdash M_1 : (\mathscr{L}(\phi) \rhd \tau_{m1})$, and $\Gamma' \vdash M_2 : (\mathscr{L}(\phi) \rhd \tau_{m2})$. We now consider the two generated constraints:

- Since $\phi \Rightarrow \mathsf{IsUn}(\tau_{m1})$, we have $\mathscr{L}(\phi) \rhd \tau_{m1} = \mathsf{Un}$ as expected.
- Since $\phi \Rightarrow \mathsf{IsKey?}(\tau_{m2}, \tau_x)$ we look at the following axioms:

$$\forall t : \forall u : (\mathsf{IsOk?}(t) \wedge \mathsf{IsKey?}(t, u)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u))$$
$$\forall t : \forall u_1 : \forall u_2 : (\mathsf{IsCha?}(t, u_1) \wedge \mathsf{IsKey?}(t, u_2)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u_1) \wedge \mathsf{IsUn}(u_2))$$
$$\forall t : \forall u : \forall \mu : \forall t_f : \forall t_s : (\mathsf{IsKey?}(t, u) \wedge \mathsf{IsPair?}(t, \mu, t_f, t_s)) \Rightarrow (\mathsf{IsUn}(t) \wedge \mathsf{IsUn}(u) \wedge \mathsf{IsUn}(t_f) \wedge \mathsf{IsUn}(t_s))$$
$$\forall t : \forall x : \forall u_1 : \forall u_2 : \forall t_f : \forall t_s : (\neg\mathsf{IsOk?}(t) \wedge \neg\mathsf{IsCha?}(t, u_1) \wedge \mathsf{IsKey?}(t, u_2) \wedge \neg\mathsf{IsPair?}(t, x, t_f, t_s) \wedge \neg\mathsf{IsUn}(t)) \Rightarrow \mathsf{IsKey}(t, u_2)$$

Together they mean that either $\mathscr{L}(\phi) \rhd \tau_{m2} = \mathscr{L}(\phi) \rhd \tau_x = \mathsf{Un}$ or $\mathscr{L}(\phi) \rhd \tau_{m2} = \mathsf{Key}(\mathscr{L}(\phi) \rhd \tau_x)$, corresponding to [Dcr-2] and [Dcr-1] respectively.

[Bgn-C]: Trivial

[End-C]: Here we have $\Psi = \mathsf{end}\ \ell(M)$ and $\ell(M) \twoheadrightarrow \xi; \psi; \varphi$. By the case we have two sub-cases.

Effect in environment: For this case we have $\phi = \psi \wedge \varphi \wedge [\![\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma)]\!]$ and $\ell(M) \in \Gamma$. Since $\phi \Rightarrow [\![\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma)]\!]$ we know $\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma)$ thus satisfying the conditions of [End-1].

Effect not in environment: For this case we have $\phi = \psi \wedge \varphi \wedge [\![\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma)]\!] \wedge ((\bigwedge_{x:\tau \in \Gamma} (\forall \xi' : \forall \xi'' : \forall x : \forall \mu : ((\neg\mathsf{CanOk}(\tau, \xi') \vee \neg\mathsf{Eq}(\xi, \xi')) \wedge (\neg\mathsf{InsCanOk}(\tau, x, \mu, \xi'') \vee \neg\mathsf{Eqi}(\xi, x, \mu, \xi'')))))) \Rightarrow \mathsf{FAIL}(\xi))$. Clearly this constraint cannot be satisfied on its own. However, since we only consider robustly safe processes, we know that if this constraint is satisfied, then there exists some constraints ϵ which are generated for the remainder of the processes, such that we have either $\mathsf{CanOk}(\tau, \xi') \in \mathscr{L}(\phi \wedge \epsilon)$ or $\mathsf{InsCanOk}(\tau, \chi, \mu, \xi') \in \mathscr{L}(\phi \wedge \epsilon)$. Moreover by ϕ we know that we have $\mathsf{Eq}(\xi, \xi')$ or $\mathsf{Eqi}(\xi, \chi, \mu, \xi')$ respectively. By Lemma 3.4.8 we know this implies $\mathscr{L}(\phi \wedge \epsilon) \lhd \xi' = \mathscr{L}(\phi \wedge \epsilon) \lhd \xi$ or $\mathscr{L}(\phi \wedge \epsilon) \Diamond (\chi, \mu, \xi') = \mathscr{L}(\phi \wedge \epsilon) \lhd \xi$. Since $\phi \Rightarrow \psi \wedge \varphi$ and by Lemma 3.4.5 we know $\mathscr{L}(\phi \wedge \epsilon) \lhd \xi = \ell(M)$. This implies that $\mathscr{L}(\phi \wedge \epsilon) \rhd \tau = \mathsf{Ok}(S)$ for which $\ell(M) \in S$. Finally, since $\phi \Rightarrow [\![\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma)]\!]$ we know $\mathsf{fn}(M) \subseteq \mathsf{dom}(\Gamma)$ thus satisfying the conditions of [End-2].

B.2. Proofs for Chapter 4

B.2.1. Proof of Theorem 4.2.4.

For any process P, there exists a type environment Γ and an assertion Ψ such that $\Gamma, \Psi \vdash P$, if and only if $\Gamma_P \vdash P \rightsquigarrow \psi; \phi$ such that $\mathscr{L}(\phi)$ is defined, $\mathscr{L}(\phi) \rhd \psi = \Psi$, and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P$ and $\mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_P(x)$

We prove this by first proving that if $\Gamma, \Psi \vdash P$, then $\Gamma_P \vdash P \rightsquigarrow \psi; \phi$ such that $\mathscr{L}(\phi)$ is defined, $\mathscr{L}(\phi) \rhd \psi = \Psi$, and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P$ and $\mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_P(x)$. We prove this by structural induction in P:

[T-In]: In this case we know that $P = \underline{M}(\lambda\vec{x})N..P'$ and we use [C-In] to generate constraints, giving us $\Gamma_M \vdash M \rightsquigarrow \tau_m; \psi_m; \phi_m$,$\Gamma_N \vdash (\lambda\vec{x})N \rightsquigarrow (\vec{\tau} \to \tau_n); \psi_n; \phi_n$, $\Gamma_{P'}, \vec{x} : \vec{\tau} \vdash P' \rightsquigarrow \psi_p; \phi_p$, and
$phi =$
$phi_m \wedge$
$phi_n \wedge$
$phi_p \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftrightarrow^{\rho^-} \tau_n]\!] \wedge \bigwedge_{\substack{\Gamma(x)=\tau_1 \\ \Gamma_M(x)=\tau_2 \\ \Gamma_N(x)=\tau_3 \\ \Gamma_{P'}(x)=\tau_4}} [\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!] \wedge$
$[\![\mathsf{if}\ \mathcal{C}(\underline{M}(\lambda\vec{x})N..P', \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]$.

From the induction hypothesis and the first criterion we get that, since $\Gamma_1 + \vec{x} : \vec{T}, \Psi_1 \vdash P$, $\Gamma_2, \Psi_2 \vdash_{\min} (\lambda\vec{x})N : \vec{T} \to U_o$, $\Gamma_3, \Psi_3 \vdash_{\min} M : U_s$, $\Gamma_M \vdash M \rightsquigarrow \tau_m; \psi_m; \phi_m$, $\Gamma_N \vdash (\lambda\vec{x})N \rightsquigarrow (\vec{\tau} \to \tau_n); \psi_n; \phi_n$, $\Gamma_{P'}, \vec{x} : \vec{\tau} \vdash P' \rightsquigarrow \psi_p; \phi_p$, we know that $\mathscr{L}(\phi_p)$ is defined, $\mathscr{L}(\phi_p)\psi \rhd \Psi_1$, $\mathsf{dom}(\Gamma_1 + \vec{x} : \vec{T}) = \{x \mid x : \tau_x \in \Gamma_{P'}, \vec{x} : \vec{\tau}$ and $\mathscr{L}(\phi_p) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma_1 + \vec{x} : \vec{T})$, $\Gamma_1 + \vec{x} : \vec{T}(x) = \mathscr{L}(\phi_p) \rhd \Gamma_{P'}, \vec{x} : \vec{\tau}(x)$, $\mathscr{L}(\phi_n)$ is defined, $\mathscr{L}(\phi_n) \rhd \tau_n = U_o$, for all i $\mathscr{L}(\phi_n) \rhd \tau_i = T_i$ $\mathscr{L}(\phi_n) \rhd \psi_n = \Psi_2$, $\mathsf{dom}(\Gamma_2) = \{x \mid x : \tau_x \in \Gamma_N$ and $\mathscr{L}(\phi_n) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma_2)$, $\Gamma_2(x) = \mathscr{L}(\phi_n) \rhd \Gamma_N(x)$, and $\mathscr{L}(\phi_m)$ is defined, $\mathscr{L}(\phi_m) \rhd \tau_m = U_s$, $\mathscr{L}(\phi_m) \rhd \psi_m = \Psi_3$, $\mathsf{dom}(\Gamma_3) = \{x \mid x : \tau_x \in \Gamma_M$ and $\mathscr{L}(\phi_m) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma_3)$, $\Gamma_3(x) = \mathscr{L}(\phi_m) \rhd \Gamma_M(x)$.

We now prove that a solution $\mathscr{L}(\phi)$ exists. Aside from $t\vec{a}u$, ϕ_m, ϕ_n, and ϕ_p do not discuss the same variables at all, and, since $\mathscr{L}(\phi_m)$ exists, $\mathscr{L}(\phi_n)$ exists, $\mathscr{L}(\phi_p)$ exists, and for all τ_i, $\mathscr{L}(\phi_n) \rhd \tau_i = T_i = \mathscr{L}(\phi_p) \rhd \tau_i$, $\mathscr{L}(\phi_m \wedge \phi_n \wedge \phi_p)$ must exist. Since $\mathscr{L}(\phi_p)\psi \rhd \Psi_1$, $\mathscr{L}(\phi_n) \rhd \psi_n = \Psi_2$, $\mathscr{L}(\phi_m) \rhd \psi_m = \Psi_3$, and ψ' is fresh variable not occurring in ϕ_m, ϕ_n, or ϕ_p, there is no reason we cannot have $\mathscr{L}(\phi_m \wedge \phi_n \wedge \phi_p) \rhd \phi' = \Psi$,which means that, by criterion 11, since $\Psi = \Psi_1 \otimes \Psi_2 \otimes \Psi_3$, $\mathscr{L}(\phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!])$ exists. Since $\mathscr{L}(\phi_m) \rhd \tau_m = U_s$ and $\mathscr{L}(\phi_n) \rhd \tau_n = U_o$ and $U_s \leftrightarrow^{\rho^-} U_o$ we also know that $\mathscr{L}(\phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftrightarrow^{\rho^-} \tau_n]\!])$ exists. Since $\mathsf{dom}(\Gamma_1 + \vec{x} : \vec{T}) = \{x \mid x : \tau_x \in \Gamma_{P'}, \vec{x} : \vec{\tau}$ and $\mathscr{L}(\phi_p) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma_1 + \vec{x} : \vec{T})$, $\Gamma_1 + \vec{x} : \vec{T}(x) = \mathscr{L}(\phi_p) \rhd \Gamma_{P'}, \vec{x} : \vec{\tau}(x)$, we know that

$$\mathscr{L}(\phi_p) \rhd \Gamma_{P'}(x) = \begin{cases} \Gamma_1(x) & if\ x \in \mathsf{dom}(\Gamma_1) \\ \varepsilon & otherwise \end{cases}$$

Since $\mathsf{dom}(\Gamma_2) = \{x \mid x : \tau_x \in \Gamma_N$ and $\mathscr{L}(\phi_n) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma_2)$, $\Gamma_2(x) = \mathscr{L}(\phi_n) \rhd \Gamma_N(x)$

$$\mathscr{L}(\phi_n) \rhd \Gamma_N(x) = \begin{cases} \Gamma_2(x) & if\ x \in \mathsf{dom}(\Gamma_2) \\ \varepsilon & otherwise \end{cases}$$

And since $\mathsf{dom}(\Gamma_3) = \{x \mid x : \tau_x \in \Gamma_M \text{ and } \mathscr{L}(\phi_m) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma_3)$, $\Gamma_3(x) = \mathscr{L}(\phi_m) \rhd \Gamma_M(x)$,

$$\mathscr{L}(\phi_m) \rhd \Gamma_M(x) = \begin{cases} \Gamma_3(x) & if\ x \in \mathsf{dom}(\Gamma_3) \\ \varepsilon & otherwise \end{cases}$$

Since all the variables in the range of Γ_P are fresh for $\phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftarrow\!\!\rho^- \tau_n]\!]$, there is nothing preventing us from constructing a solution $\mathscr{L}(\phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftarrow\!\!\rho^- \tau_n]\!])$ such that for all $x : \tau_x \in \Gamma_P$, $\mathscr{L}(\phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftarrow\!\!\rho^- \tau_n]\!]) \rhd \tau_x = \Gamma(x)$. Since $\Gamma = \Gamma_1 + \Gamma_2 + \Gamma_3$ and $\varepsilon + T = T$ for all T, we then get from criterion 9, that $\mathscr{L}(\phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftarrow\!\!\rho^- \tau_n]\!] \wedge \bigwedge_{\substack{\Gamma(x)=\tau_1 \\ \Gamma_M(x)=\tau_2 \\ \Gamma_N(x)=\tau_3 \\ \Gamma_{P'}(x)=\tau_4}} [\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!])$ exists. Finally, since ψ is fresh for $\phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftarrow\!\!\rho^- \tau_n]\!] \wedge \bigwedge_{\substack{\Gamma(x)=\tau_1 \\ \Gamma_M(x)=\tau_2 \\ \Gamma_N(x)=\tau_3 \\ \Gamma_{P'}(x)=\tau_4}} [\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!]$, there is nothing preventing us from creating a solution such that $\mathscr{L}(\phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftarrow\!\!\rho^- \tau_n]\!] \wedge \bigwedge_{\substack{\Gamma(x)=\tau_1 \\ \Gamma_M(x)=\tau_2 \\ \Gamma_N(x)=\tau_3 \\ \Gamma_{P'}(x)=\tau_4}} [\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!]) \rhd \psi = \mathscr{L}(\phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftarrow\!\!\rho^- \tau_n]\!] \wedge \bigwedge_{\substack{\Gamma(x)=\tau_1 \\ \Gamma_M(x)=\tau_2 \\ \Gamma_N(x)=\tau_3 \\ \Gamma_{P'}(x)=\tau_4}} [\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!]) \rhd \psi'$, meaning that $\mathscr{L}(\phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftarrow\!\!\rho^- \tau_n]\!] \wedge \bigwedge_{\substack{\Gamma(x)=\tau_1 \\ \Gamma_M(x)=\tau_2 \\ \Gamma_N(x)=\tau_3 \\ \Gamma_{P'}(x)=\tau_4}} [\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!] \wedge [\![\mathsf{if}\ \mathcal{C}(\underline{M}(\lambda\vec{x})N..P', \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!])$ is defined.

It should be obvious that in the above solution, $\mathscr{L}(\phi) \rhd \psi = \Psi$, and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_P(x)$

[T-Out]: Similar to above

[T-Par]: In this case we know that $P = P_1 \mid P_2$ and we use [C-Par] to generate constraints, giving us $\Gamma_1', \Gamma_1'' \vdash P_1 \rightsquigarrow \psi_1; \phi_1$, $\Gamma_2', \Gamma_2'' \vdash P_2 \rightsquigarrow \psi_2; \phi_2$, $\mathcal{F}'(P_1) = \langle \Gamma_1'', \psi^1, \phi^1 \rangle$, $\mathcal{F}'(P_2) = \langle \Gamma_2'', \psi^2, \phi^2 \rangle$, and $\phi = \phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!] \wedge [\![\psi'^2 \leq \psi^2]\!] \wedge [\![\psi_2 = \psi_2' \otimes \psi'^1]\!] \wedge [\![\psi'^1 \leq \psi'^1]\!] \wedge \bigwedge_{\substack{\Gamma_P(x)=\tau_1 \\ \Gamma_1'(x)=\tau_2 \\ \Gamma_2'(x)=\tau_3}} [\![\tau_1 = \tau_2 + \tau_3]\!] \wedge [\![\mathsf{if}\ \mathcal{C}(P_1 \mid P_2, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]$.

From the induction hypothesis we get that, since $\Gamma_1 + \Gamma_{P_2}, \Psi_1 \otimes \Psi'_{P_2} \vdash P_1$, $\Gamma_2 + \Gamma_{P_1}, \Psi_2 \otimes \Psi'_{P_1} \vdash P_2$, $\Gamma_1', \Gamma_1'' \vdash P_1 \rightsquigarrow \psi_1; \phi_1$, and $\Gamma_2', \Gamma_2'' \vdash P_2 \rightsquigarrow \psi_2; \phi_2$, we know that $\mathscr{L}(\phi_1)$ is defined, $\mathscr{L}(\phi_1) \rhd \psi_1 = \Psi_1 \otimes \Psi'_{P_2}$, and $\mathsf{dom}(\Gamma_1 + \Gamma_{P_2}) = \{x \mid x : \tau_x \in \Gamma_1', \Gamma_1'' \text{ and } \mathscr{L}(\phi_1) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma_1 + \Gamma_{P_2})$, $\Gamma_1 + \Gamma_{P_2}(x) = \mathscr{L}(\phi_1) \rhd \Gamma_1', \Gamma_1''(x)$, and $\mathscr{L}(\phi_2)$ is defined, $\mathscr{L}(\phi_2) \rhd \psi_2 = \Psi_2 \otimes \Psi'_{P_1}$, and $\mathsf{dom}(\Gamma_2 + \Gamma_{P_1}) = \{x \mid x : \tau_x \in \Gamma_2', \Gamma_2'' \text{ and } \mathscr{L}(\phi_2) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma_2 + \Gamma_{P_1})$, $\Gamma_2 + \Gamma_{P_1}(x) = \mathscr{L}(\phi_2) \rhd \Gamma_2', \Gamma_2''(x)$.

We now prove that a solution $\mathscr{L}(\phi)$ exists. The only variables discussed by both ϕ_1 and ϕ_2 are the ones in $\mathsf{ran}(\Gamma_1'') \cup \mathsf{ran}(\Gamma_2'')$. Since We know that $\Gamma_2 + \Gamma_{P_1} \vdash P_2$ we know that for any $x : T \in \Gamma_{P_1}$ the same $x : T$ has been introduced in the derivation of $\Gamma_1 + \Gamma_{P_2}, \Psi_1 \otimes \Psi'_{P_2}$. This means we know from the induction

hypothesis, that since there must exist $\Gamma_{1'}$ and $\Psi_{1'}$ such that $\Gamma_{1'}, \Psi_{1'} \vdash (\nu x : T)P_1'$ and $\Gamma_{1''} \vdash (\nu x : \tau)P_1' \rightsquigarrow \psi_{1'}; \phi_{1'}$ $\mathscr{L}(\phi_{1'}) \rhd \tau = T$ and since x is new cannot have been mentioned earlier in P_1, $\mathscr{L}(\phi_1) \rhd \tau = T$. In addition, since for all $x \in \mathsf{dom}(\Gamma_2 + \Gamma_{P_1})$, $\Gamma_2 + \Gamma_{P_1}(x) = \mathscr{L}(\phi_2) \rhd \Gamma_2', \Gamma_2''(x)$, and $\Gamma_{P_1}(x) = T$, we have $\mathscr{L}(\phi_1) \rhd \tau = \mathscr{L}(\phi_2) \rhd \tau$. This means that there exists a solution $\mathscr{L}(\phi_1 \wedge \phi_2)$, and since all the names of ϕ^1 and ϕ^2 are fresh for $\phi_1 \wedge \phi_2$ and each other, $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2)$ exists. Since ψ', ψ_1', and ψ_2' are fresh for $\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2$ we can say that $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2) \rhd \psi' = \Psi$, $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2) \rhd \psi_1' = \Psi_1$, and $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2) \rhd \psi_2' = \Psi_2$. Then we know from criterion 11 that $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!])$ exists. Since ψ'^2 is fresh for $\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!]$ we can say that $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!]) \rhd \psi'^2 = \Psi'_{P_2}$ and since $\mathscr{L}(\phi_1) \rhd \psi_1 = \Psi_1 \otimes \Psi_{P_2}$ and ψ_1 is not mentioned in $\phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!]$, $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!]) \rhd \psi_1 = \Psi_1 \otimes \Psi_{P_2}$. We therefore get from criterion 11, that $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!])$ exists. Since we know from the definition of $\mathcal{F}'$ that $\mathscr{L}(\phi^2) \rhd \psi^2 = \Psi_{P_2}$ and ψ^2 is fresh for $\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!]$ we know that $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!]) \rhd \psi^2 = \Psi_{P_2}$, and we therefore get from criterion 13 that $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!] \wedge [\![\psi'^2 \leq \psi^2]\!])$ exists. From similar arguments, we know that $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!] \wedge [\![\psi'^2 \leq \psi^2]\!] \wedge [\![\psi_2 = \psi_2' \otimes \psi'^1]\!] \wedge [\![\psi'^1 \leq \psi'^1]\!])$ exists. Since $\mathsf{dom}(\Gamma_1 + \Gamma_{P_2}) = \{x \mid x : \tau_x \in \Gamma_1', \Gamma_1''$ and $\mathscr{L}(\phi_1) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma_1 + \Gamma_{P_2})$, $\Gamma_1 + \Gamma_{P_2}(x) = \mathscr{L}(\phi_1) \rhd \Gamma_1', \Gamma_1''(x)$ we know that

$$\mathscr{L}(\phi_1)\Gamma_1', \Gamma_1''(x) = \begin{cases} \Gamma_1 + \Gamma_{P_2}(x) & if\ x \in \mathsf{dom}(\Gamma_1 + \Gamma_{P_2}) \\ \varepsilon & otherwise \end{cases}$$

And Since $\mathsf{dom}(\Gamma_2 + \Gamma_{P_1}) = \{x \mid x : \tau_x \in \Gamma_2', \Gamma_2''$ and $\mathscr{L}(\phi_2) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma_2 + \Gamma_{P_1})$, $\Gamma_2 + \Gamma_{P_1}(x) = \mathscr{L}(\phi_2) \rhd \Gamma_2', \Gamma_2''(x)$ we know that

$$\mathscr{L}(\phi_2)\Gamma_2', \Gamma_2''(x) = \begin{cases} \Gamma_2 + \Gamma_{P_1}(x) & if\ x \in \mathsf{dom}(\Gamma_2 + \Gamma_{P_1}) \\ \varepsilon & otherwise \end{cases}$$

Since all the variables in $\mathsf{ran}(\Gamma_P)$ are fresh for $\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!] \wedge [\![\psi'^2 \leq \psi^2]\!] \wedge [\![\psi_2 = \psi_2' \otimes \psi'^1]\!] \wedge [\![\psi'^1 \leq \psi'^1]\!]$ there is no reason we cannot construct a solution $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!] \wedge [\![\psi'^2 \leq \psi^2]\!] \wedge [\![\psi_2 = \psi_2' \otimes \psi'^1]\!] \wedge [\![\psi'^1 \leq \psi'^1]\!])$ such that for all $x : \tau_x \in \Gamma_P$, $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!] \wedge [\![\psi'^2 \leq \psi^2]\!] \wedge [\![\psi_2 = \psi_2' \otimes \psi'^1]\!] \wedge [\![\psi'^1 \leq \psi'^1]\!]) \rhd \tau_x = \Gamma(x)$. Since $\Gamma = \Gamma_1 + \Gamma_2$ and $T + \varepsilon = T$, we get from criterion 9 that $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!] \wedge [\![\psi'^2 \leq \psi^2]\!] \wedge [\![\psi_2 = \psi_2' \otimes \psi'^1]\!] \wedge [\![\psi'^1 \leq \psi'^1]\!] \wedge \bigwedge_{\substack{\Gamma_P(x)=\tau_1 \\ \Gamma_1'(x)=\tau_2 \\ \Gamma_2'(x)=\tau_3}} [\![\tau_1 = \tau_2 + \tau_3]\!])$ exists. Finally, since ψ is fresh for $\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!] \wedge [\![\psi'^2 \leq \psi^2]\!] \wedge [\![\psi_2 = \psi_2' \otimes \psi'^1]\!] \wedge [\![\psi'^1 \leq \psi'^1]\!] \wedge \bigwedge_{\substack{\Gamma_P(x)=\tau_1 \\ \Gamma_1'(x)=\tau_2 \\ \Gamma_2'(x)=\tau_3}} [\![\tau_1 = \tau_2 + \tau_3]\!]$ we can say that $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!] \wedge [\![\psi'^2 \leq \psi^2]\!] \wedge [\![\psi_2 = \psi_2' \otimes \psi'^1]\!] \wedge [\![\psi'^1 \leq \psi'^1]\!] \wedge \bigwedge_{\substack{\Gamma_P(x)=\tau_1 \\ \Gamma_1'(x)=\tau_2 \\ \Gamma_2'(x)=\tau_3}} [\![\tau_1 = \tau_2 + \tau_3]\!]) \rhd \phi = \Psi$, in which case we know from criterion 19 that $\mathscr{L}(\phi_1 \wedge \phi_2 \wedge \phi^1 \wedge \phi^2 \wedge [\![\psi' = \psi_1' \otimes \psi_2']\!] \wedge [\![\psi_1 = \psi_1' \otimes \psi'^2]\!] \wedge$

$[\![\psi'^2 \leq \psi^2]\!] \wedge [\![\psi_2 = \psi_2' \otimes \psi'^1]\!] \wedge [\![\psi'^1 \leq \psi'^1]\!] \wedge \bigwedge_{\substack{\Gamma_P(x)=\tau_1 \\ \Gamma_1'(x)=\tau_2 \\ \Gamma_2'(x)=\tau_3}} [\![\tau_1 = \tau_2 + \tau_3]\!] \wedge [\![\mathsf{if}\ \mathcal{C}(P_1 \mid P_2, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]$) exists.

It should be obvious that in the above solution $\mathscr{L}(\phi)\psi \rhd \Psi$, and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P$ and $\mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_P(x)$.

[T-Rep]: In this case we know that $P = *P'$, and we use [C-Rep] to generate constraints, giving us $\Gamma_P \vdash P' \rightsquigarrow \psi'; \phi_p$ and $\phi = \phi_p \wedge [\![\psi' = \psi' \otimes \psi']\!] \wedge \bigwedge_{\tau \in \mathsf{ran}(\Gamma_P)} [\![\tau = \tau + \tau]\!] \wedge [\![\mathsf{if}\ \mathcal{C}(*P', \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]$.

From the induction hypothesis, since $\Gamma, \Psi \vdash P'$ and $\Gamma_P \vdash P' \rightsquigarrow \psi'; \phi_p$, we know that $\mathscr{L}(\phi_p)$ is defined, $\mathscr{L}(\phi_p) \rhd \psi' = \Psi$, and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P$ and $\mathscr{L}(\phi_p) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi_p) \rhd \Gamma_P(x)$.

We now prove that $\mathscr{L}(\phi)$ exists. We already know from the induction hypothesis that $\mathscr{L}(\phi_p)$ exists, and since $\mathscr{L}(\phi_p) \rhd \psi' = \Psi$ and Ψ is idempotent, obviously $\mathscr{L}(\phi_p \wedge [\![\psi' = \psi' \otimes \psi']\!]) = \mathscr{L}(\phi_p)$. Since $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P$ and $\mathscr{L}(\phi_p) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi_p) \rhd \Gamma_P(x)$, we know that

$$\mathscr{L}(\phi_p) \rhd \Gamma_P(x) = \begin{cases} \Gamma(x) & if\ x \in \mathsf{dom}(\Gamma) \\ \varepsilon & otherwise \end{cases}$$

We also know that Γ is unlimited, and $\varepsilon + \varepsilon = \varepsilon$. This means that $\forall \tau \in \mathsf{ran}(\Gamma_P) : \tau = \tau + \tau$, and therefore $\mathscr{L}(\phi_p \wedge [\![\psi' = \psi' \otimes \psi']\!] \wedge \bigwedge_{\tau \in \mathsf{ran}(\Gamma_P)} [\![\tau = \tau + \tau]\!]) = \mathscr{L}(\phi_p)$. Finally, since ψ is fresh for $\phi_p \wedge [\![\psi' = \psi' \otimes \psi']\!] \wedge \bigwedge_{\tau \in \mathsf{ran}(\Gamma_P)} [\![\tau = \tau + \tau]\!]$ we can easily construct a solution $\mathscr{L}(\phi_p \wedge [\![\psi' = \psi' \otimes \psi']\!] \wedge \bigwedge_{\tau \in \mathsf{ran}(\Gamma_P)} [\![\tau = \tau + \tau]\!])$ such that $\mathscr{L}(\phi_p \wedge [\![\psi' = \psi' \otimes \psi']\!] \wedge \bigwedge_{\tau \in \mathsf{ran}(\Gamma_P)} [\![\tau = \tau + \tau]\!]) \rhd \psi = \mathscr{L}(\phi_p \wedge [\![\psi' = \psi' \otimes \psi']\!] \wedge \bigwedge_{\tau \in \mathsf{ran}(\Gamma_P)} [\![\tau = \tau + \tau]\!]) \rhd \psi' = \Psi$, and therefore, by criterion 9, we get that $\mathscr{L}(\phi_p \wedge [\![\psi' = \psi' \otimes \psi']\!] \wedge \bigwedge_{\tau \in \mathsf{ran}(\Gamma_P)} [\![\tau = \tau + \tau]\!] \wedge [\![\mathsf{if}\ \mathcal{C}(*P', \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!])$ exists.

It should be obvious that in the above solution $\mathscr{L}(\phi) \rhd \psi = \Psi$, and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P$ and $\mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_P(x)$.

[T-Res]: In this case $P = (\nu x : T)P'$, we use τ to represent T in the constraint generation, and we use [C-Res] to generate constraints, giving us $\Gamma_P, x : \tau \vdash P' \rightsquigarrow \psi_p; \phi_p$ and $\phi = \phi_p \wedge [\![\psi' = \psi_p \div x]\!] \wedge [\![\mathsf{if}\ \mathcal{C}((\nu x : \tau)P, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]$.

From the induction hypothesis, since $\Gamma + x : T, \Psi \vdash P$ and $\Gamma_P, x : \tau \vdash P' \rightsquigarrow \psi_p; \phi_p$, we know that $\mathscr{L}(\phi_p)$ is defined, $\mathscr{L}(\phi_p)\psi_p \rhd \Psi$, and $\mathsf{dom}(\Gamma + x : T) = \{x' \mid x' : \tau_x \in \Gamma_P, x : \tau$ and $\mathscr{L}(\phi_p) \rhd \tau_x \neq \varepsilon\}$ and for all $x' \in \mathsf{dom}(\Gamma + x : T)$, $\Gamma + x : T(x') = \mathscr{L}(\phi_p) \rhd \Gamma_P, x : \tau(x')$.

We will now prove that a solution $\mathscr{L}(\phi)$ exists. We know from above that $\mathscr{L}(\phi_p)$ exists. Since ψ' is fresh for ϕ_p there is no reason we cannot create a solution $\mathscr{L}(\phi_p)$ such that $\mathscr{L}(\phi_p) \rhd \psi' = \mathscr{L}(\phi_p) \rhd \psi_p \div x$, meaning that according to criterion 17 $\mathscr{L}(\phi_p \wedge [\![\psi' = \psi_p \div x]\!])$ exists. Since ψ is fresh for $\phi_p \wedge [\![\psi' = \psi_p \div x]\!]$, we can create a solution $\mathscr{L}(\phi_p \wedge [\![\psi' = \psi_p \div x]\!])$ such that $\mathscr{L}(\phi_p \wedge [\![\psi' = \psi_p \div x]\!]) \rhd \psi = \mathscr{L}(\phi_p \wedge [\![\psi' = \psi_p \div x]\!]) \rhd \psi'$, meaning that according to criterion 19, $\mathscr{L}(\phi_p \wedge [\![\psi' = \psi_p \div x]\!] \wedge [\![\mathsf{if}\ \mathcal{C}((\nu x : \tau)P, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!])$ exists.

It should be obvious that in the above solution $\mathscr{L}(\phi) \rhd \psi = \Psi$, and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P$ and $\mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_P(x)$.

[T-Ass]: In this case $P = (\!|\Psi'|\!)$ and we use [C-Ass] to generate constraints, giving us $\Gamma_P \vdash \Psi' \rightsquigarrow \psi_a; \phi_a$ and $\phi = \phi_a \wedge [\![\mathsf{if}\ \mathcal{C}((\!|\Psi|\!), \psi_a)\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi_a)\ \mathsf{else}\ \psi = \psi_a]\!]$.

From criterion 3, since $\Gamma, \Psi \vdash \Psi'$ and $\Gamma_P \vdash \Psi' \rightsquigarrow \psi_a; \phi_a$, we know that $\mathscr{L}(\phi_a)$ is defined, $\mathscr{L}(\phi_a) \rhd \psi_a = \Psi$ and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P \text{ and } \mathscr{L}(\phi_a) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi_a) \rhd \Gamma_P(x)$.

We now prove that $\mathscr{L}(\phi)$ exists. We already know from above that $\mathscr{L}(\phi_a)$ exists. And since ψ is fresh fer ϕ_a, we can construct a solution $\mathscr{L}(\phi_a)$ such that $\mathscr{L}(\phi_a) \rhd \psi = \mathscr{L}(\phi_a) \rhd \psi_a$. This means that, according to criterion 19, $\mathscr{L}(\phi_a \wedge [\![\mathsf{if}\ \mathcal{C}((\!|\Psi|\!), \psi_a)\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi_a)\ \mathsf{else}\ \psi = \psi_a]\!])$ exists.

It should be obvious that in the above solution $\mathscr{L}(\phi) \rhd \psi = \Psi$, and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_P(x)$.

[T-Cas]: In this case $P = \mathbf{case}\ \sigma_1 : P_1, \ldots, \sigma_k : P_k$ and we use [C-Cas] to generate constraints, giving us $\Gamma_P \vdash \sigma_i \rightsquigarrow \psi_{si}; \phi_{si}$ and $\Gamma_P \vdash P_i \rightsquigarrow \psi_{pi}; \phi_{pi}$ for all i such that $1 \leq i \leq k$ and $\phi = \bigwedge_{1 \leq i \leq k} (\phi_{si} \wedge \phi_{pi}) \wedge [\![\psi_{s1} = \cdots = \psi_{sk} = \psi_{p1} = \cdots = \psi_{pk} = \psi']\!] \wedge [\![\mathsf{if}\ \mathcal{C}(\mathbf{case}\ \sigma_1 : P_1, \ldots, \sigma_k : P_k, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]$.

From the induction hypothesis and criterion 2, since $\Gamma, \Psi \vdash \sigma_i$, $\Gamma, \Psi \vdash P_i$ $1 \leq i \leq k$, $\Gamma_P \vdash \sigma_i \rightsquigarrow \psi_{si}; \phi_{si}$, and $\Gamma_P \vdash P_i \rightsquigarrow \psi_{pi}; \phi_{pi}$ for all i such that $1 \leq i \leq k$, we get that $\mathscr{L}(\phi_{si})$ is defined, $\mathscr{L}(\phi_{si}) \rhd \psi_{si} = \Psi$ and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P \text{ and } \mathscr{L}(\phi_{si}) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi_{si}) \rhd \Gamma_P(x)$ and $\mathscr{L}(\phi_{pi})$ is defined, $\mathscr{L}(\phi_{pi}) \rhd \psi_{pi} = \Psi$ and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P \text{ and } \mathscr{L}(\phi_{pi}) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi_{pi}) \rhd \Gamma_P(x)$ for all i such that $1 \leq i \leq k$.

We now prove that there exists a solution $\mathscr{L}(\phi)$. We know that $\mathscr{L}(\phi_{si})$ and $\mathscr{L}(\phi_{pi})$ exist for all i such that $1 \leq i \leq k$. The only variables shared by these is $\mathsf{ran}(\Gamma_P)$. But because $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P \text{ and } \mathscr{L}(\phi_{si}) \rhd \tau_x \neq \varepsilon\}$, for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi_{si}) \rhd \Gamma_P(x)$, $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P \text{ and } \mathscr{L}(\phi_{pi}) \rhd \tau_x \neq \varepsilon\}$, and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi_{pi}) \rhd \Gamma_P(x)$ for all i such that $1 \leq i \leq k$, we know that

$$\mathscr{L}(\phi_{si}) \rhd \Gamma_P(x) = \begin{cases} \Gamma(x) & if\ x \in \mathsf{dom}(\Gamma) \\ \varepsilon & otherwise \end{cases}$$

and

$$\mathscr{L}(\phi_{pi}) \rhd \Gamma_P(x) = \begin{cases} \Gamma(x) & if\ x \in \mathsf{dom}(\Gamma) \\ \varepsilon & otherwise \end{cases}$$

This means that for all x in $\mathsf{dom}(\Gamma_P)$, $\mathscr{L}(\phi_{s1}) \rhd \Gamma_P(x) = \cdots = \mathscr{L}(\phi_{sk}) \rhd \Gamma_P(x) = \mathscr{L}(\phi_{p1}) \rhd \Gamma_P(x) = \cdots = \mathscr{L}(\phi_{pk}) \rhd \Gamma_P(x)$. Therefore there must exist a solution $\mathscr{L}(\bigwedge_{1 \leq i \leq k} (\phi_{si} \wedge \phi_{pi}))$. Since $\mathscr{L}(\phi_{s1}) \rhd \psi_{s1} = \cdots = \mathscr{L}(\phi_{sk}) \rhd \psi_{sk} = \mathscr{L}(\phi_{p1}) \rhd \psi_{p1} = \cdots = \mathscr{L}(\phi_{pk}) \rhd \psi_{pk}$ there must, according to criterion 15, exist a solution $\mathscr{L}(\bigwedge_{1 \leq i \leq k} (\phi_{si} \wedge \phi_{pi}) \wedge [\![\psi_{s1} = \cdots = \psi_{sk} = \psi_{p1} = \cdots = \psi_{pk} = \psi']\!])$. Finally, since ψ is fresh for $\bigwedge_{1 \leq i \leq k} (\phi_{si} \wedge \phi_{pi}) \wedge [\![\psi_{s1} = \cdots = \psi_{sk} = \psi_{p1} = \cdots = \psi_{pk} = \psi']\!]$, there is not reason we cannot create a solution $\mathscr{L}(\bigwedge_{1 \leq i \leq k} (\phi_{si} \wedge \phi_{pi}) \wedge [\![\psi_{s1} = \cdots = \psi_{sk} = \psi_{p1} = \cdots = \psi_{pk} = \psi']\!])$ such that $\mathscr{L}(\bigwedge_{1 \leq i \leq k} (\phi_{si} \wedge \phi_{pi}) \wedge [\![\psi_{s1} = \cdots = \psi_{sk} = \psi_{p1} = \cdots = \psi_{pk} = \psi']\!]) \rhd \psi = \mathscr{L}(\bigwedge_{1 \leq i \leq k} (\phi_{si} \wedge \phi_{pi}) \wedge [\![\psi_{s1} = \cdots = \psi_{sk} = \psi_{p1} = \cdots = \psi_{pk} = \psi']\!]) \rhd \psi'$, meaning that according to criterion 19 $\mathscr{L}(\bigwedge_{1 \leq i \leq k} (\phi_{si} \wedge \phi_{pi}) \wedge [\![\psi_{s1} = \cdots = \psi_{sk} = \psi_{p1} = \cdots = \psi_{pk} = \psi']\!] \wedge [\![\mathsf{if}\ \mathcal{C}(\mathbf{case}\ \sigma_1 : P_1, \ldots, \sigma_k : P_k, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!])$ exists.

It should be obvious that in the above solution $\mathscr{L}(\phi) \rhd \psi = \Psi$, and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_P(x)$.

[T-Wea]: In this case we know that there must be another typing rule used on P above [T-Weak], and we use the corresponding constraint generation rule to generate constraints. But instead of setting $\mathscr{L}(\phi) \rhd \psi = \mathscr{L}(\phi) \rhd \psi'$ like in the previous cases, this time $\mathscr{L}(\phi) \rhd \psi = \mathscr{L}(\phi) \rhd \psi' \otimes \Psi_2$. Since ψ is never mentioned in any part of ϕ other than $[\![\mathsf{if}\ \mathcal{C}(\mathbf{case}\ \sigma_1 : P_1, \ldots, \sigma_k : P_k, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]$ and $\mathcal{C}(P, \Psi_1)$, this is a solution $\mathscr{L}(\psi)$, and it should be obvious that in this solution $\mathscr{L}(\phi) \rhd \psi = \Psi$, and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_P(x)$.

We then prove that if $\Gamma_P \vdash P \rightsquigarrow \psi; \phi$ such that $\mathscr{L}(\phi)$ is defined, $\mathscr{L}(\phi) \rhd \psi = \Psi$, and $\mathsf{dom}(\Gamma) = \{x \mid x : \tau_x \in \Gamma_P \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$ and for all $x \in \mathsf{dom}(\Gamma)$, $\Gamma(x) = \mathscr{L}(\phi) \rhd \Gamma_P(x)$, then $\Gamma, \Psi \vdash P$. We do this by induction in the constraint generation rules of Table 4.2.1:

[C-In]: In this case we know that $P = \underline{M}(\lambda\vec{x})N.P'$, $\Gamma_M \vdash M \rightsquigarrow \tau_m; \psi_m; \phi_m$, $\Gamma_N \vdash (\lambda\vec{x})N \rightsquigarrow (\vec{\tau} \to \tau_n); \psi_n; \phi_n$, $\Gamma'_P, \vec{x} : \vec{\tau} \vdash P \rightsquigarrow \psi_p; \phi_p$, and $\phi = \phi_m \wedge \phi_n \wedge \phi_p \wedge [\![\psi' = \psi_m \otimes \psi_n \otimes \psi_p]\!] \wedge [\![\tau_m \leftarrow\!\!\rho^{-} \tau_n]\!] \bigwedge_{\substack{\Gamma(x)=\tau_1 \\ \Gamma_M(x)=\tau_2 \\ \Gamma_N(x)=\tau_3 \\ \Gamma_P(x)=\tau_4}} [\![\tau_1 = \tau_2 + \tau_3 + \tau_4]\!] \wedge$ $[\![\mathsf{if}\ \mathcal{C}(\underline{M}(\lambda\vec{x})N..P, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]$.

From the induction hypothesis and criterion 1 we get that, since $\mathscr{L}(\phi)$ is also a solution to ϕ_m, ϕ_n, and ϕ_p, we have a Γ'_M, T_M, and Ψ_M such that $\mathscr{L}(\phi) \rhd \tau_m = T_M$, $\mathscr{L}(\phi) \rhd \psi_m = \Psi_m$, $\mathsf{dom}(\Gamma'_M) = \{x \mid x : \tau_x \in \Gamma_M \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$, for all $x \in \mathsf{dom}(\Gamma'_M)$, $\Gamma'_M(x) = \mathscr{L}(\phi) \rhd \Gamma_M(x)$, and $\Gamma'_M, \Psi_M \vdash M : T$, we have a Γ'_N, $\vec{T}$, T_N and Ψ_N such that for all τ_i in $\vec{\tau}$ $\mathscr{L}(\phi) \rhd \tau_i = T_i$, $\mathscr{L}(\phi) \rhd \tau_n = T_N$, $\mathscr{L}(\phi) \rhd \psi_m = \Psi_N$, $\mathsf{dom}(\Gamma'_N) = \{x \mid x : \tau_x \in \Gamma_N \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$, for all $x \in \mathsf{dom}(\Gamma'_N)$, $\Gamma'_N(x) = \mathscr{L}(\phi) \rhd \Gamma_N(x)$, and $\Gamma'_N, \Psi_N \vdash (\lambda\vec{x})N : \vec{T} \to T_N$, and we have $\Gamma'_{P'}$ and $\Psi_{P'}$ such that $\mathscr{L}(\phi) \rhd \psi_p = \Psi_{P'}$, $\mathsf{dom}(\Gamma'_{P'}) = \{x \mid x : \tau_x \in \Gamma_{P'}, \vec{x} : \vec{T} \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$, for all $x \in \mathsf{dom}(\Gamma'_{P'})$, $\Gamma'_{P'}(x) = \mathscr{L}(\phi) \rhd \Gamma_{P'}, \vec{x} : \vec{T}(x)$, and $\Gamma'_{P'}, \Psi'_P \vdash P'$.

We know from criterion 4 that $T_M \leftarrow\!\!\rho^{-} T_N$, from criterion 10 that $\mathscr{L}(\phi) \rhd \psi' = \Psi_M \otimes \Psi_N \otimes \Psi_{P'}$, and from criterion 8 that for all $x \in \mathsf{dom}(\Gamma_P)$, $\mathscr{L}(\phi) \rhd \Gamma_P(x) = \mathscr{L}(\phi) \rhd \Gamma_M(x) \otimes \mathscr{L}(\phi) \rhd \Gamma_N(x) \otimes \mathscr{L}(\phi) \rhd \Gamma_{P'}(x)$. This means that, according to [C-In], $\Gamma, \mathscr{L}(\phi) \rhd \psi' \vdash P$ and depending on whether $\mathscr{L}(\phi) \rhd \psi' = \mathscr{L}(\phi) \rhd \psi$ or $\mathcal{C}(P, \mathscr{L}(\phi) \rhd \psi')$ and $\mathscr{L}(\phi) \rhd \psi = \mathscr{L}(\phi) \rhd \psi' \otimes \Psi_1$ we either use [T-Weak] or do not. From this we can conclude that $\Gamma, \Psi \vdash P$.

[C-Out]: Similar to the case above

[C-Par]: In this case we know that $P = P' \mid Q$, $\Gamma_{P'}, \Gamma^Q \vdash P' \rightsquigarrow \psi_p; \phi_p$, $\Gamma_Q, \Gamma^{P'} \vdash Q \rightsquigarrow \psi_q; \phi_q$, $\mathcal{F}'(P') = \Gamma^{P'}, \psi^p, \phi^p$, $\mathcal{F}'(Q) = \Gamma^Q, \psi^q, \phi^q$, and $\phi = \phi_p \wedge \phi_q \wedge \phi^p \wedge \phi^q \wedge [\![\psi' = \psi'_p \otimes \psi'_q]\!] \wedge [\![\psi_p = \psi'_p \otimes \psi^{q'}]\!] \wedge [\![\psi^{q'} \leq \psi^q]\!] \wedge [\![\psi_q = \psi'_q \otimes \psi^{p'}]\!] \wedge [\![\psi^{p'} \leq \psi^p]\!] \bigwedge_{\substack{\Gamma_P(x)=\tau_1 \\ \Gamma_{P'}(x)=\tau_2 \\ \Gamma_Q(x)=\tau_3}} [\![\tau_1 = \tau_2 + \tau_3]\!] \wedge [\![\mathsf{if}\ \mathcal{C}(P \mid Q, \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]$.

From the induction hypothesis we know that, since $\mathscr{L}(\phi)$ is also a solution to ϕ_p and ϕ_q, we have a $\Gamma'_{P'}$, $\Psi_{P'}$, Γ'_Q, and Ψ_Q such that $\mathscr{L}(\phi) \rhd \psi_p = \Psi_P$, and $\mathsf{dom}(\Gamma'_{P'}) = \{x \mid x : \tau_x \in \Gamma_{P'}, \Gamma^Q \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$, for all $x \in \mathsf{dom}(\Gamma'_{P'})$, $\Gamma'_{P'}(x) = \mathscr{L}(\phi) \rhd \Gamma_{P'}, \Gamma^Q(x)$, and $\Gamma, \Psi \vdash P$ and $\mathscr{L}(\phi) \rhd \psi_q = \Psi_Q$, and $\mathsf{dom}(\Gamma'_Q) = \{x \mid x : \tau_x \in \Gamma_Q, \Gamma^{P'} \text{ and } \mathscr{L}(\phi) \rhd \tau_x \neq \varepsilon\}$, for all $x \in \mathsf{dom}(\Gamma'_Q)$, $\Gamma'_Q(x) = \mathscr{L}(\phi) \rhd \Gamma_Q, \Gamma^{P'}(x)$, and $\Gamma, \Psi \vdash P$. We now know from criterion 8, that for all x in $\mathsf{dom}(\Gamma_P)$, $\mathscr{L}(\phi) \rhd \Gamma_P(x) = \mathscr{L}(\phi) \rhd \Gamma_{P'}(x) \otimes \mathscr{L}(\phi) \rhd \Gamma_Q(x)$, from criterion 10 that

$\mathscr{L}(\phi) \rhd \psi' = \mathscr{L}(\phi) \rhd \psi'_p \otimes \mathscr{L}(\phi) \rhd \psi'_q$, $\mathscr{L}(\phi) \rhd \psi_p = \mathscr{L}(\phi) \rhd \psi'_p \otimes \mathscr{L}(\phi) \rhd \psi^{q\prime}$, and $\mathscr{L}(\phi) \rhd \psi_q = \mathscr{L}(\phi) \rhd \psi'_q \otimes \mathscr{L}(\phi) \rhd \psi^{p\prime}$, and from criterion 12 that $\mathscr{L}(\phi) \rhd \psi^{q\prime} \leq \mathscr{L}(\phi) \rhd \psi^q$ and $\mathscr{L}(\phi) \rhd \psi^{p\prime} \leq \mathscr{L}(\phi) \rhd \psi^p$. This means that $\Gamma, \mathscr{L}(\phi) \rhd \psi' \vdash P$ and depending on whether $\mathscr{L}(\phi) \rhd \psi' = \mathscr{L}(\phi) \rhd \psi$ or $\mathcal{C}(P, \mathscr{L}(\phi) \rhd \psi')$ and $\mathscr{L}(\phi) \rhd \psi = \mathscr{L}(\phi) \rhd \psi' \otimes \Psi_1$ we either use [T-Weak] or do not. From this we can conclude that $\Gamma, \Psi \vdash P$.

[C-Rep]: In this case we know that $P = *P'$, $\Gamma_{P'} \vdash P' \rightsquigarrow \psi_p; \phi_p$, and $\phi = \phi_p \wedge [\![\psi' = \psi' \otimes \psi']\!] \wedge \bigwedge_{\tau \in \mathsf{ran}(\Gamma_{P'})} [\![\tau = \tau + \tau]\!] \wedge [\![\mathsf{if}\ \mathcal{C}(*P', \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]$.

From the induction hypothesis we know that since $\mathscr{L}(\phi)$ is also a solution to ϕ_p, we have a $\Gamma'_{P'}$ and $\Psi'_{P'}$ such that $\mathscr{L}(\phi) \rhd \psi_p = \Psi'_{P'}$, $\mathsf{dom}(\Gamma'_{P'}) = \{x \mid x : \tau_x \in \Gamma_{P'} \text{ and } \mathscr{L}(\phi)\tau_x \neq \varepsilon\}$, for all $x \in \mathsf{dom}(\Gamma'_{P'})$ we have $\Gamma'_{P'}(x) = \mathscr{L}(\phi) \rhd \tau_x$, and finally $\Gamma'_{P'}, \Psi'_{P'} \vdash P'$. Moreover we know $\Gamma_{P'} = \Gamma'_{P'}$ by the case. By criterion 8 we know that for all $x \in \mathsf{dom}(\Gamma_{P'})$ we have $\mathscr{L}(\phi) \rhd \Gamma_{P'}(x) = \mathscr{L}(\phi) \rhd \Gamma_{P'}(x) + \mathscr{L}(\phi) \rhd \Gamma_{P'}(x)$ implying that $\Gamma_{P'}$ is unlimited. Moreover by criterion 10 we can conclude that $\mathscr{L}(\phi) \rhd \psi' = \mathscr{L}(\phi) \rhd \psi' \otimes \mathscr{L}(\phi) \rhd \psi'$ implying that ψ' is idempotent. This, according to [T-Rep] implies that $\Gamma, \mathscr{L}(\phi) \rhd \psi' \vdash *P'$ and depending on whether $\mathscr{L}(\phi) \rhd \psi' = \mathscr{L}(\phi) \rhd \psi$ or $\mathcal{C}(P, \mathscr{L}(\phi) \rhd \psi')$ and $\mathscr{L}(\phi) \rhd \psi = \mathscr{L}(\phi) \rhd \psi' \otimes \Psi_1$ we either use [T-Wea] or do not. From this we can conclude that $\Gamma, \Psi \vdash P$.

[C-Res]: In this case we know that $P = (\nu x : \tau)P'$, $\Gamma_{P'}, x : \tau \vdash P' \rightsquigarrow \psi_p; \phi_p$, and $\phi = \phi_p \wedge [\![\psi' = \psi_p \div x]\!] \wedge [\![\mathsf{if}\ \mathcal{C}((\nu x : \tau)P', \psi')\ \mathsf{then}\ \mathsf{Weak}(\psi, \psi')\ \mathsf{else}\ \psi = \psi']\!]$.

From the induction hypothesis we know that since $\mathscr{L}(\phi)$ is also a solution to ϕ_p, we have a $\Gamma'_{P'}$ and $\Psi'_{P'}$ such that $\mathscr{L}(\phi) \rhd \psi_p = \Psi'_{P'}$, $\mathsf{dom}(\Gamma'_{P'}) = \{y \mid y : \tau_y \in \Gamma_{P'}, x : \tau \text{ and } \mathscr{L}(\phi)\tau_y \neq \varepsilon\}$, for all $y \in \mathsf{dom}(\Gamma'_{P'})$ we have $\Gamma'_{P'}(y) = \mathscr{L}(\phi) \rhd \tau_y$, and finally $\Gamma'_{P'}, \Psi'_{P'} \vdash P'$. Moreover we know $\Gamma_{P'}, x : \tau = \Gamma'_{P'}$ by the case. By criterion 16 we know that $\mathscr{L}(\phi) \rhd \psi = \mathscr{L}(\phi) \rhd \psi_p \div x$. This, according to [T-Res] implies that $\Gamma, x : \tau \mathscr{L}(\phi) \rhd \psi' \vdash (\nu x : \tau)P'$ and depending on whether $\mathscr{L}(\phi) \rhd \psi' = \mathscr{L}(\phi) \rhd \psi$ or $\mathcal{C}(P, \mathscr{L}(\phi) \rhd \psi')$ and $\mathscr{L}(\phi) \rhd \psi = \mathscr{L}(\phi) \rhd \psi' \otimes \Psi_1$ we either use [T-Wea] or do not. From this we can conclude that $\Gamma, \Psi \vdash P$.

[C-Ass]: Similar to the case above

[C-Cas]: Similar to the case above

B.2.2. Proof of Theorem 4.4.4.

If there exists a Γ and Ψ such that $\Gamma, \Psi \vdash P$ then P terminates

To prove this we take inspiration from [3] and start by defining the weight of a process in an environment, $wt(\Gamma, P)$. This refers to the vector $\langle n_1, n_2, \ldots, n_k \rangle$ where k is the maximum n such that there exists an x such that $x : \mathsf{Ch}^n \in \Gamma$ and for each $1 \leq i \leq k$ there are n_i outputs on channels with type Ch^i in$P$ that are not guarded by a replication. We say that $\langle n_{11}, n_{21}, \ldots, n_{k1} \rangle < \langle n_{12}, n_{22}, \ldots, n_{k2} \rangle$ if there exists an $i \leq k$ such that $n_{i1} < n_{i2}$ and for all $i < j \leq k$ $n_{j1} = n_{j2}$. $wt(\Gamma, P)$ is formally defined in Definition B.2.1, where we use $\vec{0}$ to denote the vector in which all $n_i = 0$ and $\vec{0}_i$ to denote the vector in which $n_i = 1$ and all other $n_j = 0$.

Definition B.2.1 (Weight (wt)). Given a Process P and an environment Γ we define $wt(\Gamma, P)$ in the following way:

$$
\begin{array}{ll}
wt(\Gamma, (\!|1|\!)) = \vec{0} & wt(\Gamma, \overline{a}\, x.P) = wt(\Gamma, P) + \vec{0}_n \text{ where } \Gamma(a) = \mathsf{Ch}^n \\
wt(\Gamma, *P) = \vec{0} & wt(\Gamma, P \mid Q) = wt(\Gamma, P) + wt(\Gamma, Q) \\
wt(\Gamma, \underline{a}(\lambda \vec{x})x.P) = wt(\Gamma, P) & wt(\Gamma, (\nu a : T)P) = wt(\Gamma + a : T, P) \\
 & wt(\Gamma, \mathbf{case}\ \sigma_1 : P_1, \ldots, \sigma_k : P_k) = \max(wt(\Gamma, P_1), \ldots, wt(\Gamma, P_k))
\end{array}
$$

We then show that if $\Psi \rhd P \xrightarrow{\tau} P'$ and $\Gamma, \Psi' \vdash P$ then $wt(\Gamma, P) > wt(\Gamma, P')$. Since replication, according to the definition of assertion compatibility, must be directly followed by an input, after which all in- and output must have a lower level than that input, we know that any for replicated process starting with an input on a channel with level i, the

remainder of the process must have a lower weight than $\vec{0}_i$ and we therefore have that if $P_1 \xrightarrow{ax} P_2, \Gamma_1, \Psi_1 \vdash P_1$, and $\Gamma_1(a) = \mathsf{Ch}^n$ then $wt(\Gamma_1, P_2) < wt(\Gamma_1, P_1) + \vec{0}_n$. And obviously if $P_1 \xrightarrow{\overline{a}(\nu\vec{b}:\vec{T})x} P_2$, $\Gamma_1, \Psi_1 \vdash P_1$, and $\Gamma_1(a) = \mathsf{Ch}^n$ then $wt(\Gamma_1, P_2) \leq wt(\Gamma_1, P_1) - \vec{0}_n$. Therefore if $\Psi \rhd P \xrightarrow{\tau} P'$ and $\Gamma, \Psi' \vdash P$ then $wt(\Gamma, P) > wt(\Gamma, P')$.

We now prove that if there exists a Γ and Ψ such that $\Gamma, \Psi \vdash P$ then P terminates through induction in $wt(\Gamma, P)$:

- $wt(\Gamma, P) = \vec{0}$**::** In this case the process has no output capabilities, and has therefore terminated.
- $wt(\Gamma, P) > \vec{0}$**::** As we observed above, for all P' such that $\Psi' \rhd P \xrightarrow{\tau} P'$, $wt(\Gamma, P) > wt(\Gamma, P')$. And since for this type system all Γ are unlimited, we know that there exists a Ψ'' such that $\Gamma, \Psi'' \vdash P'$, and we use the induction hypothesis to determine that P' terminates, and therefore P terminates.