A Sound and Complete Projection for Global Types

Dawit Tirore
IT University of Copenhagen, Denmark

Jesper Bengtson
IT University of Copenhagen, Denmark

Marco Carbone
IT University of Copenhagen, Denmark

Abstract

Multiparty session types is a typing discipline used to write specifications, known as global types, for branching and recursive message-passing systems. A necessary operation on global types is projection to abstractions of local behaviour, called local types. Typically, this is a computable partial function that given a global type and a role erases all details irrelevant to this role.

Computable projection functions in the literature are either unsound or too restrictive when dealing with recursion and branching. Recent work has taken a more general approach to projection defining it as a coinductive, but not computable, relation. Our work defines a new computable projection function that is sound and complete with respect to its coinductive counterpart and, hence, equally expressive. All results have been mechanised in the Coq proof assistant.

2012 ACM Subject Classification Theory of computation → Type theory; Computing methodologies → Distributed programming languages; Theory of computation → Program verification

Keywords and phrases Session types, Mechanisation, Projection, Coq

Digital Object Identifier 10.4230/LIPIcs.ITP.2023.28

Supplementary Material *Software (Coq Source Code)*: `https://github.com/Tirore96/projection` archived at `swh:1:dir:c4f38245d064fc65ed000d62ebd15826ef8da337`

Funding This work was supported by DFF-Research Project 1 Grant no. 1026-00385A, from The Danish Council for Independent Research for the Natural Sciences (FNU).

1 Introduction

Session types are types for abstracting the behaviour of communicating processes. First proposed by Honda et al. [15] for binary protocols, they specify the sequence of possible actions processes need to follow when sending and receiving messages over a channel. Session types provide a clear language for describing protocols that are guaranteed to not deadlock or contain communication errors, e.g., never receive an integer when expecting a boolean. A decade after their conception, Honda et al. [16] proposed a generalisation, called *multiparty session types*, that specifies how an arbitrary but fixed number of processes should interact with each other. Multiparty session types are based on the concept of *global types* which provide a global description of the multiparty protocol being abstracted. Recently, multiparty session types have gained interest from several communities, resulting in their integration into several mainstream programming languages [2].

Multiparty session types follow a precise approach to designing and implementing communicating processes: from global types that specify the protocols, we can automatically generate *local types*, the local specifications of the behaviour of each *role* in the protocol; then, each local type specification is (type) checked against the local code being written by the programmer. The automatic generation of local types from global types, called *projection*, is key for relating global types to implementations. Given a role, projection is an operation that erases the parts of the global type irrelevant for the role. When projection is defined the output is a local type specifying the behaviour of this role. As an example, let us consider a global type where Carl can ask Dave to either go Left or Right over some channel k:

$$\mathsf{Carl} \to \mathsf{Dave} : k \left\{ \begin{array}{l} \mathtt{Left} : \ \mathsf{Carl} \to \mathsf{Dave} : k'\langle \mathtt{Int} \rangle.\mathsf{Alice} \to \mathsf{Bob} : k''\langle \mathtt{Int} \rangle.\mathsf{end} \\ \mathtt{Right} : \ \mathsf{Carl} \to \mathsf{Dave} : k'\langle \mathtt{String} \rangle.\mathsf{Alice} \to \mathsf{Bob} : k''\langle \mathtt{Int} \rangle.\mathsf{end} \end{array} \right\}$$

Above, if Carl chooses Left, he will also send an integer (Int) over some other channel k'; otherwise, he will send a string (String). No matter what branch Carl chooses, all roles must collectively follow the description of that branch.

Nested in both branches, there is a communication over k'' of an integer Int between Alice and Bob. The projections of Carl and Alice are:

$$\mathsf{Carl} : \quad k \oplus \left\{ \begin{array}{l} \mathtt{Left} : !k'\langle \mathtt{Int} \rangle.\mathsf{end} \\ \mathtt{Right} : !k'\langle \mathtt{String} \rangle.\mathsf{end} \end{array} \right\} \qquad\qquad \mathsf{Alice} : \quad !k''\langle \mathtt{Int} \rangle.\ \mathsf{end}$$

Above, Carl makes a choice (denoted by $\oplus$), and then outputs on channel k' either something of type Int or something of type String. Alice is instead sending over channel k''. An important observation is that, since neither Alice nor Bob are informed of the choice made by Carl, their behaviour should be independent from Carl's choice. In fact, a *restriction* that projection usually imposes is that all those roles not participating to a branching communication behave the same on all branches.

In order to be able to express repetitive behaviour, global types (and local types) are usually equipped with recursion, expressed as *μ-types* [22]. For example, consider

$$\mu\mathbf{t}.\ \mathsf{Alice} \to \mathsf{Bob} : k\langle \mathtt{String} \rangle.\ \mu\mathbf{t}'.\ \mathsf{Carl} \to \mathsf{Dave} : k' \left\{ \begin{array}{l} \mathtt{Left} : \ \mathbf{t} \\ \mathtt{Right} : \ \mathsf{Alice} \to \mathsf{Bob} : k\langle \mathtt{String} \rangle.\mathbf{t}' \end{array} \right\} \tag{1}$$

The example above poses some questions on how projection should work. For Alice, should it be undefined since we cannot syntactically see her behaviour on the first branch? Or, can the projection first unfold on $\mathbf{t}$ and then generate a local type? We observe that the following global type is equivalent to (1) but does not violate our constraint on branches:

$$\mu\mathbf{t}.\ \mathsf{Alice} \to \mathsf{Bob} : k\langle \mathtt{String} \rangle.\ \mathsf{Carl} \to \mathsf{Dave} : k' \ \{\mathtt{Left} : \ \mathbf{t}, \quad \mathtt{Right} : \ \mathbf{t}\} \tag{2}$$

Since both recursive global types (1) and (2) seem to specify the same behaviour, we would assume that Alice is projected to $\mu\mathbf{t}.\ !k\langle \mathtt{String} \rangle.\ \mathbf{t}$, i.e., she repeatedly sends something of type String. The bad news is that the projection algorithms available in the literature do not allow global types like (1) to be projected while the equivalent type (2) can be projected.

The most common way of defining projection is as a structurally recursive partial function on global types, which we call *standard projection.* Recent work [13] defines projection as a coinductive relation on coinductive types, which intuitively are a complete (possibly infinite) unfolding of recursive protocols. Both approaches come with trade-offs. Standard projection is a computable procedure, which is necessary for multiparty session types to support decidable type checking, but it has limits as pointed out above. The coinductive approach is more general but, to the best of our knowledge, there are no equivalent computable algorithms available in the literature. The discrepancy between standard and coinductive projection was initially pointed out by Ghilezan et al. [13]. They correctly show that the canonical way partial projection treats the binders of μ-types causes standard projection to be undefined for some μ-types that have a coinductive projection, such as global type (1). In this paper, we define a procedure on μ-types that implements the projection on coinductive types.

Contributions and Structure. The main contribution of this paper is the definition of a computable projection function that is sound and complete with respect to a coinductive projection relation. All our proofs have been mechanised in the Coq [21] proof assistant[1].

We structure the paper as follows. Section 2 walks through existing variants of standard projection and their pitfalls. Section 3 introduces global and local coinductive types as well as a coinductive projection relation from the former to the latter. Section 4 introduces a projection function from global to local μ-types, proves that it is sound and complete with respect to its coinductive counterpart, and Section 5 proves that it is decidable. Section 6 describes key insights from our Coq mechanisation, Section 7 covers related and future work, and Section 8 concludes.

2 Global Types, Local Types, and Standard Projection

The purpose of this section is two-fold: introducing the syntax of global and local types and a walk through computable definitions of projection found in the literature.

Syntax. Let $\mathcal{P}$ be a set of roles (ranged over by $\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}, \mathsf{t}$), $\mathcal{L}$ a totally ordered set of labels (ranged over by l), and $\mathcal{X}$ a set of recursion variables ranged over by $\mathbf{t}$.

▶ **Definition 1** (Inductive Types [17]). *Global types G^μ and local types T^μ are μ-types generated inductively by the following grammars, where U represents primitive types:*

$$\begin{array}{ll} G^\mu ::= & \mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\langle U\rangle.G^\mu \mid \mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\{l_j : G^\mu_j\}_{j\in J} \mid \mu\mathbf{t}.G^\mu \mid \mathbf{t} \mid \mathsf{end}^\mu \\ T^\mu ::= & !^\mu k\langle U\rangle.T^\mu \mid ?^\mu k\langle U\rangle.T^\mu \mid k \oplus^\mu \{l_j : T^\mu_j\}_{j\in J} \mid k \&^\mu \{l_j : T^\mu_j\}_{j\in J} \mid \mu\mathbf{t}.T^\mu \mid \mathbf{t} \mid \mathsf{end}^\mu \end{array}$$

The type $\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\langle U\rangle.G^\mu$ denotes a communication between roles p_1 and p_2 via channel k of a message of type U, which then proceeds as G^μ. Similarly, the type $\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\{l_j : G^\mu_j\}_{j\in J}$ denotes a communication between two roles where, given the set of indices J, role p_1 selects a branch with label l_i, and then proceeds as G^μ_i. Types $\mu\mathbf{t}.G^\mu$ and $\mathbf{t}$ model recursive protocols. Finally, end^μ denotes the successful termination of a protocol. A message type U is just a basic value type: extensions of this are irrelevant for the focus of this paper.

For local types, the type $!^\mu k\langle U\rangle.T^\mu$ outputs a message of type U over channel k, while its dual, $?^\mu k\langle U\rangle.T^\mu$ receives a message of type U over k. Types $k \oplus^\mu \{l_j : T^\mu_j\}_{j\in J}$ and $k \&^\mu \{l_j : T^\mu_j\}_{j\in J}$ implement branching where the former is the type of a process that internally selects a branch l_i and communicates it over channel k, while the latter is the type of a process that offers choices $l_1, \ldots, l_n$ (for $J = \{1, \ldots, n\}$ with $n \geq 1$) over channel k. We overload the type end^μ and use it also for local types.

We deal with recursive variables in a standard way and write capture-avoiding substitution as $G^\mu_1[G^\mu_2/\mathbf{t}]$. Moreover, types can be contractive: a μ-type G^μ (or T^μ) is contractive if, for any of its subexpressions with shape $\mu\mathbf{t}_0.\mu\mathbf{t}_1 ... \mu\mathbf{t}_n.\mathbf{t}$, the body $\mathbf{t}$ is not $\mathbf{t}_0$ [22]. We allow non-contractive μ-types and will in the next section show how to enforce contractiveness by requiring that a μ-type is related to a coinductive type.

Overview of projections. For each role, we use projection to relate global and local types. We start our overview with the projection proposed by Castro-Perez et al. [7] which can be found in Figure 1. The projection $\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\langle U\rangle.\ G^\mu \restriction^\mu_{\mathsf{p}}$ produces either a sending or a

[1] `https://github.com/Tirore96/projection`

$$(\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\langle U\rangle.G^\mu) \upharpoonright^\mu_{\mathsf{p}} = \begin{cases} !^\mu k\langle U\rangle.(G^\mu \upharpoonright^\mu_{\mathsf{p}}) & \text{if } \mathsf{p} = \mathsf{p}_1 \text{ and } \mathsf{p}_1 \neq \mathsf{p}_2 \\ ?^\mu k\langle U\rangle.(G^\mu \upharpoonright^\mu_{\mathsf{p}}) & \text{if } \mathsf{p} = \mathsf{p}_2 \text{ and } \mathsf{p}_1 \neq \mathsf{p}_2 \\ G^\mu \upharpoonright^\mu_{\mathsf{p}} & \text{if } \mathsf{p} \notin \{\mathsf{p}_1, \mathsf{p}_2\} \end{cases}$$

$$(\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\{l_j : G^\mu_j\}_{j\in J}) \upharpoonright^\mu_{\mathsf{p}} = \begin{cases} k \oplus^\mu \{l_j : (G^\mu_j \upharpoonright^\mu_{\mathsf{p}})\}_{j\in J} & \text{if } \mathsf{p} = \mathsf{p}_1 \text{ and } \mathsf{p}_1 \neq \mathsf{p}_2 \\ k \,\&^\mu \{l_j : (G^\mu_j \upharpoonright^\mu_{\mathsf{p}})\}_{j\in J} & \text{if } \mathsf{p} = \mathsf{p}_2 \text{ and } \mathsf{p}_1 \neq \mathsf{p}_2 \\ (G^\mu_1 \upharpoonright^\mu_{\mathsf{p}}) & \text{if } \mathsf{p} \notin \{\mathsf{p}_1, \mathsf{p}_2\} \text{ and} \\ & \quad \forall i, j \in J.\ G^\mu_i \upharpoonright^\mu_{\mathsf{p}} = G^\mu_j \upharpoonright^\mu_{\mathsf{p}} \\ \perp & \textit{otherwise} \end{cases}$$

$$(\mu\mathbf{t}.G^\mu) \upharpoonright^\mu_{\mathsf{p}} = \begin{cases} \mu\mathbf{t}.(G^\mu \upharpoonright^\mu_{\mathsf{p}}) & \text{if } \mathsf{guardedVar}(\mathsf{t}, G^\mu \upharpoonright^\mu_{\mathsf{p}}) \\ \mathsf{end}^\mu & \text{otherwise} \end{cases} \qquad \mathbf{t} \upharpoonright^\mu_{\mathsf{p}} = \mathbf{t} \qquad \mathsf{end}^\mu \upharpoonright^\mu_{\mathsf{p}} = \mathsf{end}^\mu.$$

$$\mathsf{guardedVar}(\mathsf{t}, G^\mu) = \begin{cases} \mathsf{guardedVar}(\mathsf{t}, G^\mu_1) & \text{if } G^\mu = \mu\mathbf{t}'.G^\mu_1 \\ \mathbf{t} \neq \mathbf{t}' & \text{if } G^\mu = \mathbf{t}' \\ \mathsf{true} & \text{otherwise} \end{cases}$$

Figure 1 The *standard projection of G onto* p, written $G \upharpoonright^\mu_{\mathsf{p}}$ [7].

receiving action if the role p is equal to p_1 or p_2 respectively, otherwise the action is deleted. The projection of branching $\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\{l_j : G^\mu_j\}_{j\in J} \upharpoonright^\mu_{\mathsf{p}}$ works similarly but, when role p is not involved, all branches must project to exactly the same type. This requirement is known as *plain merge*. *Full merge*, used for example by Ghilezan et al. [13], is a more permissive operation which merges local types with distinct external choices. We discuss an extension of our work to full merge in Section 7. For recursion $\mu\mathbf{t}.G^\mu$, G^μ is projected only if the result is a contractive local type (checked by the $\mathsf{guardedVar}$ predicate). Finally, variable $\mathbf{t}$ and the type end^μ project directly to their local counterparts.

The use of $\mathsf{guardedVar}$ formally fixes a problem with the original projection [17] that could generate non-contractive types, which is unsound (informally fixed by forbidding non-contractive types). Alternatively, Demangeon and Yoshida [11] fix this issue by replacing the side condition with $G^\mu \upharpoonright^\mu_{\mathsf{p}} \neq \mathbf{t}$. However, all these projections invite the counterexample:

$$\mathsf{p} \xrightarrow{\mu} \mathsf{q} : k\langle U\rangle.\ \mu\mathbf{t}.\ \mathsf{r} \xrightarrow{\mu} \mathsf{s} : k'\{l_1 : \mathsf{end}^\mu,\ l_2 : \mathbf{t}\} \upharpoonright^\mu_{\mathsf{p}} \tag{3}$$

which is undefined because the branch condition fails. Since p is not a role in the branch, the desired result of this projection should be $!^\mu k\langle U\rangle.\ \mathsf{end}^\mu$. Bejleri and Yoshida [5] solve this with a recursion condition testing participation in the body

$$(\mu\mathbf{t}.G^\mu) \upharpoonright^\mu_{\mathsf{p}} = \begin{cases} \mu\mathbf{t}.(G^\mu \upharpoonright^\mu_{\mathsf{p}}) & \text{if } \mathsf{p} \in G^\mu \\ \mathsf{end}^\mu \end{cases} \tag{4}$$

This function always generates contractive types, but the projection of

$$\mu\mathbf{t}.\ \mathsf{p} \xrightarrow{\mu} \mathsf{q} : k\langle U\rangle.\ \mu\mathbf{t}'.\ \mathsf{r} \xrightarrow{\mu} \mathsf{s} : k'\langle U'\rangle.\ \mathbf{t} \upharpoonright^\mu_{\mathsf{p}} \tag{5}$$

incorrectly results in the local type $\mu\mathbf{t}.!^\mu k\langle U\rangle.\mathsf{end}^\mu$ rather than the desired $\mu\mathbf{t}.!^\mu k\langle U\rangle.\ \mathbf{t}$. Glabbeek et al. [27] fix it by adding a variable constraint to the recursion condition:

$$(\mu\mathbf{t}.G^\mu) \upharpoonright^\mu_{\mathsf{p}} = \begin{cases} \mu\mathbf{t}.(G^\mu \upharpoonright^\mu_{\mathsf{p}}) \\ \mathsf{end}^\mu & \text{if } \mathsf{p} \notin G^\mu \text{ and } \mu\mathbf{t}.G^\mu \text{ is closed} \end{cases} \tag{6}$$

This way, the projection in (5) correctly results in the type $\mu\mathbf{t}.!^{\mu}k\langle U\rangle.\mathbf{t}$. To the best of our knowledge, this is the most general and sound version of projection, but it still does not capture certain global types whose infinite unfolding is intuitively projectable. One such example is equivalent to (1), modulo renaming, from the introduction:

$$\mu\mathbf{t}.\ \mathsf{p} \xrightarrow{\mu} \mathsf{q} : k\langle U\rangle.\ \mu\mathbf{t}'.\ \mathsf{r} \xrightarrow{\mu} \mathsf{s} : k'\{l_1 : \mathbf{t},\quad l_2 : \mathsf{p} \xrightarrow{\mu} \mathsf{q} : k\langle U\rangle.\ \mathbf{t}'\} \upharpoonright^{\mu}_{\mathsf{p}} \tag{7}$$

Here, the branching condition fails because $\mathbf{t}$ is syntactically not the same type as $!^{\mu}k\langle U\rangle.\mathbf{t}'$. But how can we recognise that $\mathbf{t}$ and $!^{\mu}k\langle U\rangle.\mathbf{t}'$ are equivalent in this case? Our main insight is that standard projection can be performed in two steps: first, a boolean predicate tests projectability by unfolding μ-operators; and, when the check is passed, a translation function generates the local type by instead structurally recursing under the μ-operators. Checking projectability by unfolding μ-operators makes termination non-trivial and we explore this in Section 5. This approach lets us recognise (7) as projectable.

3 Projection on Coinductive Types

In this section, we define what an ideal projection is. The inductive definition of global types uses μ-types in order to represent infinite behaviour which, as shown by our examples, can create issues with projection. A possible solution to this issue is to get rid of μ-types and work with fully unfolded types (infinite trees). Originally, Honda et al. [17] suggested this approach informally. Later, Ghilezan et al. [13] turned this intuition into a version of global types which, instead of using an inductive definition, uses coinductive types. This had the drawback of projection not being computable. The goal of this section is to define coinductive types, a way to relate them to inductive types, and then a definition of projection without μ-types. Although we do not compute projections of coinductive types, we use them as a specification of how a correct projection should behave.

Syntax. We start by giving the coinductive definition of both global and local types.

▶ **Definition 2** (Coinductive Types). *The syntax of coinductive global and local types, denoted as G^{ν} and T^{ν} respectively, is coinductively defined as:*

$$\begin{array}{lcl} G^{\nu} & ::= & \mathsf{p}_1 \xrightarrow{\nu} \mathsf{p}_2 : k\langle U\rangle.G^{\nu} \mid \mathsf{p}_1 \xrightarrow{\nu} \mathsf{p}_2 : k\{l_j : G^{\nu}_j\}_{j\in J} \mid \mathsf{end}^{\nu} \\ T^{\nu} & ::= & !^{\nu}k\langle U\rangle.T^{\nu} \mid ?^{\nu}k\langle U\rangle.T^{\nu} \mid k \oplus^{\nu} \{l_j : T^{\nu}_j\}_{j\in J} \mid k \&^{\nu} \{l_j : T^{\nu}_j\}_{j\in J} \mid \mathsf{end}^{\nu} \end{array}$$

Coinductive types can be infinite but regular coinductive types can be finitely represented. A regular coinductive type has a finite set of distinct subterms [20] meaning that it must be circularly defined and have repeating structure if it is infinitely large. This makes it possible to store a regular coinductive type in, e.g., computer memory, or represent it as a μ-type.

In order to reason effectively about μ-types and their coinductive counterparts we need a means to relate the two. We follow the style of Castro-Perez et al. [7], using an unravelling relation $\mathcal{R}$, formally defined as:

▶ **Definition 3** (Unravelling). Unravelling*, for both global and local types, and denoted by* $G^\mu \mathrel{\mathcal{R}} G^\nu$ *and* $T^\mu \mathrel{\mathcal{R}} T^\nu$ *respectively, is defined by the following rules:*

$$\frac{}{\mathsf{end}^\mu \mathrel{\mathcal{R}} \mathsf{end}^\nu} \qquad \frac{G^\mu[\mu\mathbf{t}.G^\mu/\mathbf{t}] \mathrel{\mathcal{R}} G^\nu}{\mu\mathbf{t}.G^\mu \mathrel{\mathcal{R}} G^\nu} \qquad \frac{G^\mu \mathrel{\mathcal{R}} G^\nu}{\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\langle U\rangle.G^\mu \mathrel{\mathcal{R}} \mathsf{p}_1 \xrightarrow{\nu} \mathsf{p}_2 : k\langle U\rangle.G^\nu}$$

$$\frac{\forall j \in J.\ G^\mu_j \mathrel{\mathcal{R}} G^\nu_j}{\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\{l_j : G^\mu_j\}_{j\in J} \mathrel{\mathcal{R}} \mathsf{p}_1 \xrightarrow{\nu} \mathsf{p}_2 : k\{l_j : G^\nu_j\}_{j\in J}} \qquad \frac{\forall j \in J.\ T^\mu_j \mathrel{\mathcal{R}} T^\nu_j}{k \oplus^\mu \{l_j : T^\mu_j\}_{j\in J} \mathrel{\mathcal{R}} k \oplus^\nu \{l_j : T^\nu_j\}_{j\in J}}$$

$$\frac{T^\mu \mathrel{\mathcal{R}} T^\nu}{!^\mu k\langle U\rangle.T^\mu \mathrel{\mathcal{R}} !^\nu k\langle U\rangle.T^\nu} \qquad \frac{T^\mu \mathrel{\mathcal{R}} T^\nu}{?^\mu k\langle U\rangle.T^\mu \mathrel{\mathcal{R}} ?^\nu k\langle U\rangle.T^\nu} \qquad \frac{\forall j \in J.\ (T^\mu_j \mathrel{\mathcal{R}} T^\nu_j)}{k \,\&^\mu \{l_j : T^\mu_j\}_{j\in J} \mathrel{\mathcal{R}} k \,\&^\nu \{l_j : T^\nu_j\}_{j\in J}}$$

The unravelling relation is defined using both inductive and coinductive inference rules, where we use single lines for inductive rules and double lines for coinductive ones. A coinductive derivation may be circular and discharged by referring to a previous identical part of the inference tree whereas inductive leaves are discharged using an inductive base-case rule in the standard manner, which in our case are the rules relating end^μ and end^ν. The reason for this split is that if the μ-operator could be unravelled using a coinductive rule [unfold^ν] then we could relate any non-contractive μ-type to any coinductive type G^ν.

$$\text{Incorrect rule: } \frac{G[\mu\mathbf{t}.G] \mathrel{\mathcal{R}} G^\nu}{\mu\mathbf{t}.G \mathrel{\mathcal{R}} G^\nu}\,[\mathsf{unfold}^\nu] \qquad \text{Unwanted derivation: } \frac{\dfrac{}{\mu\mathbf{t}.t \mathrel{\mathcal{R}} G^\nu}}{\mu\mathbf{t}.t \mathrel{\mathcal{R}} G^\nu}\,[\mathsf{unfold}^\nu] \tag{8}$$

Castro-Perez et al. have a rule like [unfold^ν] and they solve this problem by requiring that all μ-types are contractive. We make this side condition redundant by making [unfold^ν] inductive and we have found that this simplifies our proofs. This is because the usual two conditions on μ-types, namely closedness and contractiveness, are captured by unravelling.

▶ **Proposition 4.** *G^μ is closed and contractive iff there exists G^ν such that $G^\mu \mathrel{\mathcal{R}} G^\nu$*

Proof. The direction ($\Longleftarrow$) is harder than the other. We prove it by contradiction, assuming both an inductive definition of non-contractiveness and $G^\mu \mathrel{\mathcal{R}} G^\nu$. ◀

The mixing of inductive and coinductive inference rules is non-standard and in Section 6 we show concretely how to formally define such inference systems. For now, we show an example of how to relate an inductive and coinductive global type by $\mathcal{R}$.

▶ **Example 5.** Consider the unravelling of

$$\mu\mathbf{t}.\ \mathsf{r} \xrightarrow{\mu} \mathsf{s} : k\{l_1 : \mathbf{t},\ l_2 : \mathsf{end}^\mu\}$$

One branch is a recursion variable and the other is end indicating we will need both inductive and coinductive rules to close the derivation. We show this inductive global type unravels to

$$G^\nu := \mathsf{r} \xrightarrow{\nu} \mathsf{s} : k\{l_1 : G^\nu,\ l_2 : \mathsf{end}^\nu\}$$

This is shown by the derivation below

$$\frac{\dfrac{\dfrac{}{\mu\mathbf{t}.\ \mathsf{r} \xrightarrow{\mu} \mathsf{s} : k\{l_1 : \mathbf{t},\ l_2 : \mathsf{end}^\mu\} \mathrel{\mathcal{R}} G^\nu} \qquad \dfrac{}{\mathsf{end}^\mu \mathrel{\mathcal{R}} \mathsf{end}^\nu}}{\mathsf{r} \xrightarrow{\mu} \mathsf{s} : k\{l_1 : \mu\mathbf{t}.\ \mathsf{r} \xrightarrow{\mu} \mathsf{s} : k\{l_1 : \mathbf{t},\ l_2 : \mathsf{end}^\mu\},\ l_2 : \mathsf{end}^\mu\} \mathrel{\mathcal{R}} G^\nu}}{\mu\mathbf{t}.\ \mathsf{r} \xrightarrow{\mu} \mathsf{s} : k\{l_1 : \mathbf{t},\ l_2 : \mathsf{end}^\mu\} \mathrel{\mathcal{R}} G^\nu} \tag{9}$$

where the arrow marks the cycle that solves the coinductive part of the proof. Visually, the arrow must pass a double line for the proof to be valid.

$$\frac{\mathsf{p} \in \{\mathsf{p}_1, \mathsf{p}_2\} \vee \mathsf{guarded}^{\nu}_{\mathsf{p}}(G^{\nu})}{\mathsf{guarded}^{\nu}_{\mathsf{p}}(\mathsf{p}_1 \xrightarrow{\nu} \mathsf{p}_2 : k\langle U\rangle.G^{\nu})} \qquad \frac{\mathsf{p} \in \{\mathsf{p}_1, \mathsf{p}_2\} \vee \forall j.\ \mathsf{guarded}^{\nu}_{\mathsf{p}}(G^{\nu}_j)}{\mathsf{guarded}^{\nu}_{\mathsf{p}}(\mathsf{p}_1 \xrightarrow{\nu} \mathsf{p}_2 : k\{l_j : G^{\nu}_j\}_{j\in J})} \qquad \frac{\mathsf{guarded}^{\mu}_{\mathsf{p}}(G[\mu\mathbf{t}.G/t])}{\mathsf{guarded}^{\mu}_{\mathsf{p}}(\mu\mathbf{t}.G)}$$

$$\frac{\mathsf{p} \in \{\mathsf{p}_1, \mathsf{p}_2\} \vee \mathsf{partOf}^{\nu}_{\mathsf{p}}(G^{\nu})}{\mathsf{partOf}^{\nu}_{\mathsf{p}}(\mathsf{p}_1 \xrightarrow{\nu} \mathsf{p}_2 : k\langle U\rangle.G^{\nu})} \qquad \frac{\mathsf{p} \in \{\mathsf{p}_1, \mathsf{p}_2\} \vee \exists j \in J.\ \mathsf{partOf}^{\nu}_{\mathsf{p}}(G^{\nu}_j)}{\mathsf{partOf}^{\nu}_{\mathsf{p}}(\mathsf{p}_1 \xrightarrow{\nu} \mathsf{p}_2 : k\{l_j : G^{\nu}_j\}_{j\in J})} \qquad \frac{\mathsf{partOf}^{\mu}_{\mathsf{p}}(G[\mu\mathbf{t}.G/t])}{\mathsf{partOf}^{\mu}_{\mathsf{p}}(\mu\mathbf{t}.G)}$$

Figure 2 Definitions of predicates $\mathsf{guarded}^{\nu}$, $\mathsf{guarded}^{\mu}$, partOf^{ν}, and partOf^{μ}. The $\mathsf{guarded}^{\mu}$ and partOf^{μ} predicates additionally have identical rules to their $\mathsf{guarded}^{\nu}$ and partOf^{ν} counterparts, except for being defined for G^{μ} and not G^{ν} – these rules have been elided.

$$\frac{G^{\nu} \downharpoonleft^{\nu}_{\mathsf{p}} T^{\nu}}{\begin{array}{c}\mathsf{p} \xrightarrow{\nu} \mathsf{p}_2 : k\langle U\rangle.G^{\nu} \downharpoonleft^{\nu}_{\mathsf{p}} \\ !^{\nu}k\langle U\rangle.T^{\nu}\end{array}}\ [\mathsf{M1}\!\downharpoonleft^{\nu}] \qquad \frac{\mathsf{p} \neq \mathsf{p}_1 \quad G^{\nu} \downharpoonleft^{\nu}_{\mathsf{p}} T^{\nu}}{\begin{array}{c}\mathsf{p}_1 \xrightarrow{\nu} \mathsf{p} : k\langle u\rangle.G^{\nu} \downharpoonleft^{\nu}_{\mathsf{p}} \\ ?^{\nu}k\langle U\rangle.T^{\nu}\end{array}}\ [\mathsf{M2}\!\downharpoonleft^{\nu}] \qquad \frac{\neg\mathsf{partOf}^{\nu}_{\mathsf{p}}(G^{\nu})}{G^{\nu} \downharpoonleft^{\nu}_{\mathsf{p}} \mathsf{end}^{\nu}}\ [\mathsf{End}\!\downharpoonleft^{\nu}]$$

$$\frac{\mathsf{p} \notin \{\mathsf{p}_1, \mathsf{p}_2\} \qquad \mathsf{guarded}^{\nu}_{\mathsf{p}}(G^{\nu}) \qquad G^{\nu} \downharpoonleft^{\nu}_{\mathsf{p}} T^{\nu}}{\mathsf{p}_1 \xrightarrow{\nu} \mathsf{p}_2 : k\langle U\rangle.G^{\nu} \downharpoonleft^{\nu}_{\mathsf{p}} T^{\nu}}\ [\mathsf{M}\!\downharpoonleft^{\nu}] \qquad \frac{\forall j.\ G^{\nu}_j \downharpoonleft^{\nu}_{\mathsf{p}} T^{\nu}_j}{\begin{array}{c}\mathsf{p} \xrightarrow{\nu} \mathsf{p}_2 : k\{l_j : G^{\nu}_j\}_{j\in J} \downharpoonleft^{\nu}_{\mathsf{p}} \\ k \oplus^{\nu} \{l_j : T^{\nu}_j\}_{j\in J}\end{array}}\ [\mathsf{B1}\!\downharpoonleft^{\nu}]$$

$$\frac{J \neq \{\} \qquad \mathsf{p} \notin \{\mathsf{p}_1, \mathsf{p}_2\} \qquad \begin{array}{r}\forall j.\ G^{\nu}_j \downharpoonleft^{\nu}_{\mathsf{p}} T^{\nu} \wedge \\ \mathsf{guarded}^{\nu}_{\mathsf{p}}(G^{\nu}_j)\end{array}}{\mathsf{p}_1 \xrightarrow{\nu} \mathsf{p}_2 : k\{l_j : G^{\nu}_j\}_{j\in J} \downharpoonleft^{\nu}_{\mathsf{p}} T^{\nu}}\ [\mathsf{B}\!\downharpoonleft^{\nu}] \qquad \frac{\mathsf{p} \neq \mathsf{p}_1 \qquad \forall j,\ G^{\nu}_j \downharpoonleft^{\nu}_{\mathsf{p}} T^{\nu}_j}{\begin{array}{c}\mathsf{p}_1 \xrightarrow{\nu} \mathsf{p} : k\{l_j : G^{\nu}_j\}_{j\in J} \downharpoonleft^{\nu}_{\mathsf{p}} \\ k\ \&^{\nu} \{l_j : T^{\nu}_j\}_{j\in J}\end{array}}\ [\mathsf{B2}\!\downharpoonleft^{\nu}]$$

Figure 3 The projection on coinductive types, denoted $G^{\nu} \downharpoonleft^{\nu}_{\mathsf{p}} T^{\nu}$, is defined by coinductive rules.

In order to define projection from coinductive global types to coinductive local types, we require the two auxiliary predicates $\mathsf{guarded}^{\nu}_{\mathsf{p}}(G^{\nu})$ and $\mathsf{partOf}^{\nu}_{\mathsf{p}}(G^{\nu})$. The former asserts that p appears in all branches of G^{ν} at finite depth, and the latter asserts that p occurs somewhere in G^{ν} at finite depth. To reason about finite depth these predicates are inductively defined. We also define similar predicates $\mathsf{guarded}^{\mu}_{\mathsf{p}}(G^{\mu})$ and $\mathsf{partOf}^{\mu}_{\mathsf{p}}(G^{\mu})$ for inductive global types G^{μ}. All four predicates are defined in Fig. 2.

The rules for projection are presented in Figure 3. Rules $[\mathsf{M1}\!\downharpoonleft^{\nu}]$, $[\mathsf{M2}\!\downharpoonleft^{\nu}]$, $[\mathsf{B1}\!\downharpoonleft^{\nu}]$, and $[\mathsf{B2}\!\downharpoonleft^{\nu}]$ handle the cases where a projected role p takes part in communication or branching. Note that our projection allows sender and receiver in a communication to be equal. This case is a special case of rule $[\mathsf{M1}\!\downharpoonleft^{\nu}]$. The rules $[\mathsf{M}\!\downharpoonleft^{\nu}]$, $[\mathsf{B}\!\downharpoonleft^{\nu}]$, and $[\mathsf{End}\!\downharpoonleft^{\nu}]$ handle the cases where p does not take part. In these cases, in order for projection to continue, p must occur in all possible future branches, otherwise the projection maps to end. These rules are similar to those given by Castro-Perez et al. [7] as well as Jacobs et al. [19].

Guardedness, enforced by the predicate $\mathsf{guarded}^{\nu}_{\mathsf{p}}$ in the $[\mathsf{M}\!\downharpoonleft^{\nu}]$ and $[\mathsf{B}\!\downharpoonleft^{\nu}]$ rules, is necessary in order to avoid unwanted derivations similar to that for unravelling in (8).

▶ **Example 6.** We can now use unravelling and coinductive projection to relate the global type $\mu\mathbf{t}.\ \mathsf{p} \xrightarrow{\mu} \mathsf{q} : k\langle U\rangle.\ \mu\mathbf{t}'.\ \mathsf{r} \xrightarrow{\mu} \mathsf{s} : k'\{l_1 : \mathbf{t},\quad l_2 : \mathsf{p} \xrightarrow{\mu} \mathsf{q} : k\langle U\rangle.\ \mathbf{t}'\}$ seen in (7) with $\mu\mathbf{t}.!^{\mu}k\langle U\rangle.\mathbf{t}$. They respectively unravel to

$$G^{\nu} := \mathsf{p} \xrightarrow{\nu} \mathsf{q} : k\langle U\rangle.\mathsf{r} \xrightarrow{\nu} \mathsf{s} : k'\{l_1 : G^{\nu},\quad l_2 : G^{\nu}\}$$
$$E^{\nu} :=!^{\nu}k\langle U\rangle.\ E^{\nu}$$

We can now derive $G^\nu \downharpoonright^\nu_\mathsf{p} E^\nu$ by $[\mathsf{M1}\downharpoonright^\nu]$, $[\mathsf{B}\downharpoonright^\nu]$ followed by $[\mathsf{M1}\downharpoonright^\nu]$ and mark a cycle to the conclusion $G^\nu \downharpoonright^\nu_\mathsf{p} E^\nu$. This precisely justifies why we wish to project the inductive global type in (7) over role p to the local type $\mu\mathbf{t}.!^\mu k\langle U\rangle.\mathbf{t}$.

4 Projection on Inductive Types: Soundness and Completeness

The coinductive projection predicate $\downharpoonright^\nu$ represents the specification of an ideal projection from coinductive global types to coinductive local types. In this section, we present a projection function proj on μ-types that is sound and complete with respect to the $\downharpoonright^\nu$ projection predicate. We extend on previous work by Castro-Perez et al. [7] whose projection function is shown to be sound but not complete.

▶ **Definition 7** (proj). *The function* $\mathsf{proj} : \mathcal{P} \to G^\mu \rightharpoonup T^\mu$*, written* $\mathsf{proj}_\mathsf{p}(G^\mu)$*, is the projection of the global* μ*-type* G^μ *with respect to the role* p *and is defined as:*

$$\mathsf{proj}_\mathsf{p}(G^\mu) = \begin{cases} \mathsf{trans}_\mathsf{p}(G^\mu) & \textit{if } \mathsf{projectable}_\mathsf{p}(G^\mu) \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

Our projection function features two auxiliary entities, namely the translation function trans and the predicate $\mathsf{projectable}$ which precisely separate the generation of the local type and the check for projectability respectively.

▶ **Definition 8** (trans). *The function* $\mathsf{trans} : \mathcal{P} \to G^\mu \to T^\mu$ *is identical to the function* $\downharpoonright^\mu$ *(see Figure 1) except for the branching case, defined as:*

$$\mathsf{trans}_\mathsf{p}(\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\{l_j : G^\mu_j\}_{j\in J}) \quad = \quad \begin{cases} k \oplus^\mu \{l_j : \mathsf{trans}_\mathsf{p}(G^\mu_j)\}_{j\in J} & \textit{if } \mathsf{p} = \mathsf{p}_1 \textit{ and } \mathsf{p}_1 \neq \mathsf{p}_2 \\ k \&^\mu \{l_j : \mathsf{trans}_\mathsf{p}(G^\mu_j)\}_{j\in J} & \textit{if } \mathsf{p} = \mathsf{p}_2 \textit{ and } \mathsf{p}_1 \neq \mathsf{p}_2 \\ \mathsf{trans}_\mathsf{p}(G^\mu_1) & \textit{if } \mathsf{p} \notin \{\mathsf{p}_1, \mathsf{p}_2\} \end{cases}$$

The only difference from Definition 2 is that the removal of the branching condition has made trans total. These conditions are checked by the $\mathsf{projectable}$ predicate and the challenging part of implementing proj is proving decidability of this predicate.

▶ **Definition 9** ($\mathsf{projectable}$). *The predicate* $\mathsf{projectable}_\mathsf{p}(G^\mu)$ *states that the global* μ*-type* G^μ *is projectable with respect to the role* p *and is defined as:*

$$\mathsf{projectable}_\mathsf{p}(G^\mu) = \exists G^\nu\ T^\nu.\ G^\mu \mathrel{\mathcal{R}} G^\nu \ \wedge\ \mathsf{trans}_\mathsf{p}(G^\mu) \mathrel{\mathcal{R}} T^\nu \ \wedge\ G^\nu \downharpoonright^\nu_\mathsf{p} T^\nu$$

The predicate states that the μ-types G^μ and $\mathsf{trans}_\mathsf{p}(G^\mu)$ are related by unravelling to some coinductive types G^ν and T^ν respectively, and that T^ν is the coinductive projection of G^ν with respect to p. This predicate is decidable and we detail why in Section 5.

Soundness. Proving that proj is sound with respect to $\downharpoonright^\nu$ is relatively straightforward.

▶ **Theorem 10.** *If* $\mathsf{proj}_\mathsf{p}(G^\mu)$ *is defined then there exist coinductive types* G^ν *and* T^ν *such that* $G^\mu \mathrel{\mathcal{R}} G^\nu$, $\mathsf{proj}_\mathsf{p}(G^\mu) \mathrel{\mathcal{R}} T^\nu$ *and* $G^\nu \downharpoonright_p T^\nu$.

Proof. Follows directly from the definition of proj and $\mathsf{projectable}$ by setting G^ν and T^ν to their corresponding types obtained from $\mathsf{projectable}$. ◀

$$\begin{tikzcd} G^\nu \arrow[r, "\downharpoonright^\nu"] & T^\nu \\ G^\mu \arrow[u, "\mathcal{R}"] \arrow[r, "\mathsf{proj}"] & T^\mu \arrow[u, "\mathcal{R}"'] \end{tikzcd}$$

Completeness. For completeness, we require an auxiliary operation $\mathsf{unfold}(\cdot)$ on global and local μ-types that unfolds all binders until an interaction prefix or end are exposed.

$$\mathsf{unfold}(G^\mu) = \mathsf{unfold\_once}^{|G^\mu|}(G^\mu) \qquad \mathsf{unfold\_once}(G^\mu) = \begin{cases} G_1^\mu[\mu\mathbf{t}.G_1^\mu/t], & \text{if } G^\mu = \mu\mathbf{t}.G_1^\mu \\ G^\mu & \text{otherwise} \end{cases}$$

Above, $|G^\mu|$ is the μ-height of G^μ, i.e., the number of initial consecutive binders found in G^μ. Here, f^n denotes repeated function composition. For example, $|\mu\mathbf{t}.\mathsf{end}| = 1$ and $|\mathsf{p} \xrightarrow{\mu} \mathsf{p}' : k\langle U\rangle.\mu\mathbf{t}.\ \mathsf{end}| = 0$. We overload unfolding with $\mathsf{unfold}(T^\mu)$ and $|T^\mu|$, for having the corresponding meaning on local types.

$$\begin{tikzcd} G^\nu \arrow[r, "\downharpoonright^\nu"] \arrow[d, "\mathcal{R}"'] & T^\nu \arrow[d, "\mathcal{R}"'] \\ G^\mu \arrow[r, "\mathsf{proj}"] & T^\mu \end{tikzcd}$$

In order to show completeness of proj with respect to $\downharpoonright^\nu$, we need to show that if $G^\nu \downharpoonright^\nu_\mathsf{p}\ T^\nu$ and $G^\mu\ \mathcal{R}\ G^\nu$ then $\mathsf{proj}_\mathsf{p}(G^\mu)$ is defined and $\mathsf{proj}_\mathsf{p}(G^\mu)\ \mathcal{R}\ T^\nu$. We prove this by showing that $\mathsf{trans}_\mathsf{p}(G^\mu)$ unravels to $\mathsf{tocoind}(\mathsf{trans}_\mathsf{p}(G^\mu))$; then, we show $\mathsf{trans}_\mathsf{p}(G^\mu) = \mathsf{proj}_\mathsf{p}(G^\mu)$ and $\mathsf{tocoind}(\mathsf{trans}_\mathsf{p}(G^\mu)) = T^\nu$. The function $\mathsf{tocoind}$ is defined as

▶ **Definition 11** (tocoind). *The corecursive function* $\mathsf{tocoind} : T^\mu \to T^\nu$ *is defined as*

$$\mathsf{tocoind}(T^\mu) = \begin{cases} !^\nu k\langle U\rangle.\mathsf{tocoind}(T^\mu) & \text{if } \mathsf{unfold}(T^\mu) = !^\mu k\langle U\rangle.T^\mu \\ ?^\nu k\langle U\rangle.\mathsf{tocoind}(T^\mu) & \text{if } \mathsf{unfold}(T^\mu) = ?^\mu k\langle U\rangle.T^\mu \\ k \oplus^\nu \{l_j : \mathsf{tocoind}(T_j^\mu)\}_{j\in J} & \text{if } \mathsf{unfold}(T^\mu) = k \oplus^\mu \{l_j : T_j^\mu\}_{j\in J} \\ k \&^\nu \{l_j : \mathsf{tocoind}(T_j^\mu)\}_{j\in J} & \text{if } \mathsf{unfold}(T^\mu) = k \&^\mu \{l_j : T_j^\mu\}_{j\in J} \\ \mathsf{end}^\nu & \text{otherwise} \end{cases}$$

Note that, $T^\mu\ \mathcal{R}\ \mathsf{tocoind}(T^\mu)$ does not always hold, as $\mathcal{R}$ is only defined for closed and contractive T^μ. However, for closed global types, trans does unravel to a coinductive type.

▶ **Lemma 12** (Unraveling of trans). *If G^μ is closed then* $\mathsf{trans}_\mathsf{p}(G^\mu)\ \mathcal{R}\ \mathsf{tocoind}(\mathsf{trans}_\mathsf{p}(G^\mu))$.

Proof. Since G^μ is closed, we know that $\mathsf{trans}_\mathsf{p}(G^\mu)$ is closed. Moreover, the image of $\mathsf{trans}_\mathsf{p}$ is always contractive. For any closed and contractive local type T^μ, we know that $T^\mu\ \mathcal{R}\ \mathsf{tocoind}(T^\mu)$, by coinduction on $\mathcal{R}$. In particular this holds for $\mathsf{trans}_\mathsf{p}(G^\mu)$. ◀

▶ **Lemma 13** (trans as projection). *If $G^\mu\ \mathcal{R}\ G^\nu$ and $G^\nu \downharpoonright^\nu_\mathsf{p}\ T^\nu$ then* $\mathsf{tocoind}(\mathsf{trans}_\mathsf{p}(G^\mu)) = T^\nu$.

Proof. By coinduction using the candidate relation $\{(\mathsf{tocoind}(\mathsf{trans}_\mathsf{p}(G^\mu)), T^\nu) \mid G^\nu \downharpoonright^\nu_\mathsf{p} T^\nu \wedge G^\mu\ \mathcal{R}\ G^\nu\}$. From $G^\nu \downharpoonright^\nu_\mathsf{p}\ T^\nu$, derive $\mathsf{guarded}^\nu_\mathsf{p}(G^\nu) \vee T^\nu = \mathsf{end}^\nu$. The first case is proven by induction on $\mathsf{guarded}^\nu_\mathsf{p}(G^\nu)$; for the second we know from $G^\nu \downharpoonright^\nu_\mathsf{p}\ \mathsf{end}^\nu$ and $G^\mu\ \mathcal{R}\ G^\nu$ that $\neg\mathsf{partOf}^\nu_\mathsf{p}(G^\mu)$ and hence $\mathsf{tocoind}(\mathsf{trans}_\mathsf{p}(G^\mu)) = \mathsf{end}^\nu$. ◀

From these Lemmas, completeness follows immediately.

▶ **Theorem 14.** *If $G^\nu \downharpoonright^{co}_p T^\nu$ and $G^\mu\ \mathcal{R}\ G^\nu$ then* $\mathsf{proj}_\mathsf{p}(G^\mu)$ *is defined and* $\mathsf{proj}_\mathsf{p}(G^\mu)\ \mathcal{R}\ T^\nu$.

Proof. From $G^\mu\ \mathcal{R}\ G^\nu$, we know using Proposition 4 that G^μ is closed. Applying Lemma 12, we have that $\mathsf{trans}_\mathsf{p}(G^\mu)\ \mathcal{R}\ \mathsf{tocoind}(\mathsf{trans}_\mathsf{p}(G^\mu))$. Finally, from Lemma 13, we have that $\mathsf{trans}_\mathsf{p}(G^\mu)\ \mathcal{R}\ T^\nu$. It thus holds that $\mathsf{projectable}_\mathsf{p}(G^\mu)$, so $\mathsf{proj}_\mathsf{p}(G^\mu)$ is defined and $\mathsf{trans}_\mathsf{p}(G^\mu) = \mathsf{proj}_\mathsf{p}(G^\mu)$ letting us conclude $\mathsf{proj}_\mathsf{p}(G^\mu)\ \mathcal{R}\ T^\nu$. ◀

$$\frac{\mathsf{unfold}(G^\mu) \downdownharpoons^\mu_{\mathsf{p}} \mathsf{unfold}(T^\mu)}{G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu}\ [\mathsf{Unf}\downharpoonleft^\mu] \qquad \frac{\mathsf{p} \notin \{\mathsf{p}_1, \mathsf{p}_2\} \quad \mathsf{guarded}^\mu_{\mathsf{p}}(G^\mu) \quad G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu}{\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\langle U\rangle.G^\mu \downdownharpoons^\mu_{\mathsf{p}} T^\mu}\ [\mathsf{M}\downharpoonleft^\mu]$$

$$\frac{G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu}{\mathsf{p} \xrightarrow{\mu} \mathsf{p}_1 : k\langle U\rangle.G^\mu \downdownharpoons^\mu_{\mathsf{p}} !^\mu k\langle U\rangle.T^\mu}\ [\mathsf{M1}\downharpoonleft^\mu] \qquad \frac{\forall j.\ G^\mu_j \downharpoonleft^\mu_{\mathsf{p}} T^\mu_j}{\mathsf{p} \xrightarrow{\mu} \mathsf{p}_1 : k\{l_j : G^\mu_j\}_{j\in J} \downdownharpoons^\mu_{\mathsf{p}} k \oplus^\mu \{l_j : T^\mu_j\}_{j\in J}}\ [\mathsf{B1}\downharpoonleft^\mu]$$

$$\frac{\mathsf{p} \neq \mathsf{p}_1 \quad G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu}{\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p} : k\langle U\rangle.G^\mu \downdownharpoons^\mu_{\mathsf{p}} ?^\mu k\langle U\rangle.T^\mu}\ [\mathsf{M2}\downharpoonleft^\mu] \qquad \frac{\mathsf{p} \neq \mathsf{p}_1 \quad \forall j.\ G^\mu_j \downharpoonleft^\mu_{\mathsf{p}} T^\mu_j}{\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p} : k\{l_j : G^\mu_j\}_{j\in J} \downdownharpoons^\mu_{\mathsf{p}} k \,\&^\mu \{l_j : T^\mu_j\}_{j\in J}}\ [\mathsf{B2}\downharpoonleft^\mu]$$

$$\frac{\neg\mathsf{partOf}^\mu_{\mathsf{p}}(G^\mu) \qquad \mathsf{Unravels}(G^\mu)}{G^\mu \downdownharpoons^\mu_{\mathsf{p}} \mathsf{end}_c}\ [\mathsf{End}\downharpoonleft^\mu] \qquad \frac{J \neq \{\} \quad \mathsf{p} \notin \{\mathsf{p}_1, \mathsf{p}_2\} \quad \forall j.\ G^\mu_j \downharpoonleft^\mu_{\mathsf{p}} T^\mu \wedge \mathsf{guarded}^\mu_{\mathsf{p}}(G^\mu_j)}{\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\{l_j : G^\mu_j\}_{j\in J} \downdownharpoons^\mu_{\mathsf{p}} T^\mu}\ [\mathsf{B}\downharpoonleft^\mu]$$

Figure 4 *Intermediate projection* on inductive types, written as $G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu$.

5 Deciding Projectability

In this section, we show that projectable is decidable. We do this in two steps: first, we define the *intermediate projection* $G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu$ and show that it is sound and complete with respect to our coinductive projection; second, given a pair (G^μ, T^μ), we construct a graph and show that deciding $G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu$ can be reduced to checking properties of that graph.

An Intermediate Projection. The rules defining $G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu$, presented in Figure 4, are similar to those for coinductive projection, but also enforce the unfolding operation unfold on both μ-types. Initially, the only applicable rule is $[\mathsf{Unf}\downharpoonleft^\mu]$, which unfolds μ-types. Then, the rules inspired by coinductive projection are used. In order to enforce unfolding every time we apply any other rule, we use the auxiliary relation $\downdownharpoons^\mu_{\mathsf{p}}$.

We now show that there is a correspondence between intermediate projection $\downharpoonleft^\mu_{\mathsf{p}}$ and coinductive projection $\downharpoonleft^\nu_{\mathsf{p}}$. In order to do so, we need to define how to construct a coinductive type from an inductive one. We have shown how to do this for inductive local types with $\mathsf{tocoind}(T^\mu)$ and we overload this tocoind function to similarly work with inductive global types G^μ. We use the abbreviations $\mathsf{Unravels}(G^\mu)$ and $\mathsf{Unravels}(T^\mu)$ for $G^\mu \mathrel{\mathcal{R}} \mathsf{tocoind}(G^\mu)$ and $T^\mu \mathrel{\mathcal{R}} \mathsf{tocoind}(T^\mu)$ respectively.

▶ **Lemma 15** (Unraveling of Projection).
$G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu$ *iff* $\mathsf{Unravels}(G^\mu)$ *and* $\mathsf{Unravels}(T^\mu)$ *and* $\mathsf{tocoind}(G^\mu) \downharpoonleft^\nu_{\mathsf{p}} \mathsf{tocoind}(T^\mu)$.

Proof. ($\Longrightarrow$) Derive both $\mathsf{Unravels}(G^\mu)$ and $\mathsf{Unravels}(T^\mu)$ by coinduction on $\mathcal{R}$ and inversion on $G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu$. Prove $\mathsf{tocoind}(G^\mu) \downharpoonleft^\nu_{\mathsf{p}} \mathsf{tocoind}(T^\mu)$ by coinduction on $\downharpoonleft^\nu_{\mathsf{p}}$ and derive from $G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu$ that $\mathsf{guarded}^\mu_{\mathsf{p}}(G^\mu) \vee \mathsf{unfold}(T^\mu) = \mathsf{end}^\mu$ and proceed as in Lemma 13.
($\Longleftarrow$) Proof by coinduction on $\downharpoonleft^\mu_{\mathsf{p}}$ and derive from $\mathsf{tocoind}(G^\mu) \downharpoonleft^\nu_{\mathsf{p}} \mathsf{tocoind}(T^\mu)$ that $\mathsf{guarded}^\nu_{\mathsf{p}}(\mathsf{tocoind}(G^\mu)) \vee \mathsf{tocoind}(T^\mu) = \mathsf{end}^\nu$, case analysis on the disjunction as in Lemma 13, inverting $\mathsf{Unravels}(G^\mu)$ and $\mathsf{Unravels}(T^\mu)$ to derive the shape of $G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu$. ◀

▶ **Corollary 16.** $\mathsf{projectable}_{\mathsf{p}}(G^\mu)$ *iff* $G^\mu \downharpoonleft^\mu_{\mathsf{p}} \mathsf{trans}_{\mathsf{p}}(G^\mu)$.

Proof. For ($\Longrightarrow$) we first show for any G^μ and G^ν, if $G^\mu \mathrel{\mathcal{R}} G^\nu$ then $G^\nu = \mathsf{tocoind}(G^\mu)$ (and similarly for local types). Then both directions follow from Lemma 15. ◀

Deciding $G^\mu \downharpoonleft^\mu_{\mathsf{p}} T^\mu$ is similar to deciding recursive type equivalence. Treatment of recursive types as graphs for equivalence testing is a well known approach [26] and solves the problem by testing properties of reachable nodes in a directed graph. In this section, we do the same for deciding $G^\mu \upharpoonright^\mu_{\mathsf{p}} T^\mu$. First, we show how to transform global and local types into graphs. Then, we obtain a graph of the pair (G^μ, T^μ) by joining the graphs of G^μ and T^μ. Deciding $G^\mu \upharpoonright^\mu_{\mathsf{p}} T^\mu$ corresponds to testing a property on all reachable nodes of that graph.

Graphs. We first give the formal definition of graph, following that of Eikelder [26].

▶ **Definition 17** (Graph)**.** *A directed graph is a triple* (Q, d, δ) *where:*

- Q *is a finite set of nodes*
- d*:* $Q \to \mathbb{N}$ *is a function returning the number of outgoing edges from a node*
- $\delta : (Q \times \mathbb{N}) \rightharpoonup Q$ *is the partial successor function such that* $\delta(q, i)$ *is the* i^{th} *successor of* q*, for* $0 < i \leq d(q)$ *nodes, and is undefined for all other* i*.*

Given a graph (Q, d, δ), we define the procedure sat_P which computes if all reachable nodes from an initial node q satisfy a given property P.

▶ **Definition 18** (sat_P)**.** *The function* $\mathsf{sat}_P : 2^Q \to Q \to \{0,1\}$*, parameterised by a boolean predicate* $P : Q \to \{0, 1\}$*, is defined as:*

$$\mathsf{sat}_P(V, q) = \begin{cases} 1 & \text{if } q \in V \\ P(q) \;\wedge\; \bigwedge_{i<d(q)} \mathsf{sat}_P\big(\{q\} \cup V,\ \delta(q, i)\big) & \text{otherwise} \end{cases}$$

Given a set of visited nodes V, a current state q, and the predicate P, the function returns 1 if the node has already been visited; otherwise, it will recursively check the successors.

Global and Local Types as Graphs. We now give a procedure for constructing a graph from a global type. The graph construction for local types is similar and therefore omitted.

▶ **Definition 19** (Global type graph)**.** *The graph of a global type* G^μ *is* $(\mathsf{enum}_g(G^\mu), d_g, \delta_g)$ *where* enum_g, d_g *and* δ_g *are defined as:*

$$\mathsf{enum}_g(\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\langle U\rangle.G^\mu) = \{\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\langle U\rangle.G^\mu\} \cup \mathsf{enum}_g(G^\mu) \qquad \mathsf{enum}_g(\mathsf{end}) = \{\mathsf{end}\}$$

$$\mathsf{enum}_g(\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\{l_j : G^\mu_j\}_{j\in J}) = \{\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\{l_j : G^\mu_j\}_{j\in J}\} \;\cup\; \bigcup_{j\in J} \mathsf{enum}_g(G^\mu_j)$$

$$\mathsf{enum}_g(\mathbf{t}) = \{\mathbf{t}\} \qquad \mathsf{enum}_g(\mu\mathbf{t}.G^\mu) = \{\mu\mathbf{t}.G^\mu\} \cup \{G^\mu_1[\mu\mathbf{t}.G^\mu/\ \mathbf{t}] \mid G^\mu_1 \in \mathsf{enum}_g(G^\mu)\}$$

$$d_g(G^\mu) = \begin{cases} 1 & \text{if } \mathsf{unfold}(G^\mu) = \mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\langle u\rangle.G^\mu \\ |J| & \text{if } \mathsf{unfold}(G^\mu) = \mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\{l_j : G^\mu_j\}_{j\in J} \\ 0 & \text{otherwise} \end{cases}$$

$$\delta_g(G^\mu, i) = \begin{cases} G^\mu_1 & \text{if } \mathsf{unfold}(G^\mu) = \mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\langle u\rangle.G^\mu_1 \;\wedge\; i = 1 \\ G^\mu_i & \text{if } \mathsf{unfold}(G^\mu) = \mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\{l_j : G^\mu_j\}_{j\in J} \;\wedge\; 0 < i \leq |J| \\ \textit{undefined} & \text{otherwise} \end{cases}$$

The enumeration function enum_g collects all subterms of a global type. In the case of $\mu\mathbf{t}.G^\mu$, it enumerates all subterms of the body G^μ that can contain free occurrences of $\mathbf{t}$, and substitute them all for $\mu\mathbf{t}.G^\mu$. These subterms are all nodes of the global type graph G^μ.

▶ **Example 20.** We show the global type graph of our main example from (7).

$$\begin{tikzcd} \boxed{\mu\mathbf{t}.G^\mu} \arrow[r, bend left=10, "1"] & \boxed{\mu\mathbf{t}'.(G^\mu_1[\mu\mathbf{t}.G^\mu/\mathbf{t}])} \arrow[l, bend left=10, "1"] \arrow[r, "2"] & \boxed{\mathsf{p} \to \mathsf{q} : k\langle U\rangle.\mu\mathbf{t}'.(G^\mu_1[\mu\mathbf{t}.G^\mu/\mathbf{t}])} \arrow[l, bend right=30, "1"'] \end{tikzcd}$$

where $G^\mu := \mathsf{p} \to \mathsf{q} : k\langle U\rangle.\ \mu\mathbf{t}'.G_1^\mu$ and $G_1^\mu := \mathsf{r} \to \mathsf{s} : k'\{l_1 : \mathbf{t},\ l_2 : \mathsf{p} \to \mathsf{q} : k\langle U\rangle.\ \mathbf{t}'\}$.

Given a global type G^μ, we wish to use $\mathsf{sat}_P(\{\}, G^\mu)$ to assert whether P holds for all nodes reachable by δ_g. To ensure termination of this procedure, we show that the set of reachable nodes is finite, a consequence of Q being closed under δ_g.

▶ **Lemma 21.** *If $G_1^\mu \in \mathit{enum}_g(G^\mu)$ and $0 \le i < d_g(G_1^\mu)$, then $\delta_g(G_1^\mu, i) \in \mathit{enum}_g(G^\mu)$.*

Proof. Let δ_{aux} be δ_g without the use of unfold in the first two case distinctions, i.e., $\delta_g = \delta_{aux} \circ \mathsf{unfold}$. Showing $\mathsf{enum}_g(G^\mu)$ is closed under δ reduces to showing $\mathsf{enum}_g(G^\mu)$ is closed under δ_{aux} and unfold. By definition, $\mathsf{enum}_g(G^\mu)$ is closed under δ_{aux}. For $\mathsf{enum}_g(G^\mu)$ to be closed under unfold, it suffices to show that it is closed under unfold_once, which follows by induction on the μ-height. ◀

We are now ready to show how to use our graph construction for proving a property of a global type using sat_P. We do that by proving that $\mathsf{Unravels}(G^\mu)$ is decidable. In this case, we instantiate P in sat_P with a predicate that disallows global types to unfold to a top level μ-operator or a recursion variable.

▶ **Definition 22** (UnravelPred). *The predicate $\mathit{UnravelPred} : G^\mu \to \{0,1\}$ is defined as:*

$$\mathit{UnravelPred}(G^\mu) = \begin{cases} 0 & \text{if } \mathit{unfold}(G^\mu) = \mu\mathbf{t}.G_1^\mu \vee \mathit{unfold}(G^\mu) = \mathbf{t} \\ 1 & \text{otherwise} \end{cases}$$

▶ **Lemma 23.** *$\mathit{Unravels}(G^\mu)$ iff $\mathit{sat}_{\mathit{UnravelPred}}(\{\}, G^\mu) = 1$*

The instantiation $\mathsf{sat}_{\mathsf{UnravelPred}}$ tests that G^μ and all successors of G^μ unfold to a message communication, a branching or end^μ. The procedure will for example fail for $\mu\mathbf{t}.\mathbf{t}$. More details on this procedure are given in Section 6.

We conclude this part by defining the partial functions LG, LT and PL_{p}. Given an inductive global type, function LG returns its *unfolded prefix*, i.e., information about its first occurring interaction.

▶ **Definition 24** (LG). *The function $LG \in G \rightharpoonup (\mathcal{P} \times \mathcal{P} \times \mathcal{C} \times (\{\bot\} \cup U))$ is defined as:*

$$LG(G^\mu) = \begin{cases} (\mathsf{p}_1, \mathsf{p}_2, k, U) & \text{if } \mathit{unfold}(G^\mu) = \mathsf{p}_1 \to \mathsf{p}_2 : k\langle U\rangle.G_1^\mu \\ (\mathsf{p}_1, \mathsf{p}_2, k, \bot) & \text{if } \mathit{unfold}(G^\mu) = \mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\{l_j : G_j^\mu\}_{j\in J} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Similar to how LG returns the unfolded prefix in a global type, we define the corresponding operation on local types as LT. We use the set $\{!, ?\}$ to indicate whether the communication is a send (!) or a receive (?).

▶ **Definition 25** (LT). *The function $LT : T \rightharpoonup (\{!, ?\} \times \mathcal{C} \times (\{\bot\} \cup U))$ is defined as:*

$$LT(T^\mu) = \begin{cases} (!, k, U) & \text{if } \mathit{unfold}(T^\mu) = !^\mu k\langle U\rangle.T_1^\mu \\ (?, k, U) & \text{if } \mathit{unfold}(T^\mu) = ?^\mu k\langle U\rangle.T_1^\mu \\ (!, k, \bot) & \text{if } \mathit{unfold}(T^\mu) = k \oplus^\mu \{l_j : T_j^\mu\}_{j\in J} \\ (?, k, \bot) & \text{if } \mathit{unfold}(T^\mu) = k\ \&^\mu \{l_j : T_j^\mu\}_{j\in J} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Finally, we can define a projection function on prefixes, i.e., a function that given a role and an unfolded prefix of a global type, returns an unfolded prefix of a local type.

▶ **Definition 26.** *The function $PL_{\mathsf{p}} \in (\mathcal{P} \times \mathcal{P} \times \mathcal{C} \times (\{\bot\} \cup U)) \rightharpoonup (\{!, ?\} \times \mathcal{C} \times (\{\bot\} \cup U))$ is defined as:*

$$PL_{\mathsf{p}}(\mathsf{p}_1, \mathsf{p}_2, k, U) = \begin{cases} (!, k, U), & \text{if } \mathsf{p}_1 = \mathsf{p} \\ (?, k, U), & \text{if } \mathsf{p}_2 = \mathsf{p} \text{ and } \mathsf{p} \neq \mathsf{p}_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Combining global and local type graphs. The next step towards deciding membership in $\downharpoonright^{\mu}_{\mathsf{p}}$ is to combine the graphs of G^μ and T^μ into a single graph with respect to a role p.

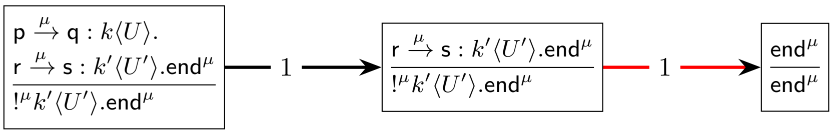
▶ **Definition 27** (Joint Global and Local Type Graph). *The graph of* (G^μ, T^μ) *with respect to* p *is the graph* $(\mathit{enum}(G^\mu, T^\mu), d, \delta_{\mathsf{p}})$*, where* enum, d *and* δ_{p} *defined as:*

$$\mathit{enum}(G^\mu, T^\mu) = \mathit{enum}(G^\mu) \times \mathit{enum}(T^\mu) \qquad d(G^\mu, T^\mu) = \mathit{min}(d_g(G^\mu), d_l(T^\mu))$$

$$\delta_{\mathsf{p}}((G^\mu, T^\mu), i) = \begin{cases} (\delta_g(G^\mu, i), \delta_l(T^\mu, i)) & \textit{if } \mathsf{p} \in LG(G^\mu) \wedge 0 < i \leq d(G^\mu, T^\mu) \\ (\delta_g(G^\mu, i), T^\mu) & \textit{if } \mathsf{p} \notin LG(G^\mu) \wedge 0 < i \leq d_g(G^\mu) \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

The set of nodes is the Cartesian product of the nodes in the global type graph and the local type graph. The successor function δ_{p} takes a step with respect to a role p and the case distinction depends on this role. If p is in $LG(G^\mu)$, the i^{th} successor of the graph is the i^{th} successor of the global and local type graph respectively. If p is not in the unfolded prefix, the global type moves to its successor while the local type stays fixed.

▶ **Example 28.** We show the joint graph of $\mathsf{p} \xrightarrow{\mu} \mathsf{q} : k\langle U\rangle.\mathsf{r} \xrightarrow{\mu} \mathsf{s} : k'\langle U'\rangle.\mathsf{end}^\mu$ and $!^\mu k\langle U'\rangle.\mathsf{end}^\mu$ with respect to role c marking the edges black when the local type stays fixed.


$\mathsf{p} \xrightarrow{\mu} \mathsf{q} : k\langle U\rangle.$
$\mathsf{r} \xrightarrow{\mu} \mathsf{s} : k'\langle U'\rangle.\mathsf{end}^\mu$
$!^\mu k'\langle U'\rangle.\mathsf{end}^\mu$
1
$\mathsf{r} \xrightarrow{\mu} \mathsf{s} : k'\langle U'\rangle.\mathsf{end}^\mu$
$!^\mu k'\langle U'\rangle.\mathsf{end}^\mu$
1
end^μ
end^μ

Deciding membership in $\downharpoonright^{\mu}_{\mathsf{p}}$. We define the predicate $\mathsf{ProjPred}_{\mathsf{p}}$ to decide membership in $\downharpoonright^{\mu}_{\mathsf{p}}$. Intuitively, this predicate partitions the rules of $\downharpoonright^{\mu}_{\mathsf{p}}$ into three sets such that the projected role p

1. is in the unfolded prefix ([M1$\downharpoonright^\mu$], [M2$\downharpoonright^\mu$], [B1$\downharpoonright^\mu$], [B2$\downharpoonright^\mu$])
2. is not in the unfolded prefix, but the role is guarded in the global type ([B$\downharpoonright^\mu$],[M$\downharpoonright^\mu$])
3. is not part of the global type ([End$\downharpoonright^\mu$]).

We call rules in the first set *prefix rules* and rules in the second set *guarded rules*. The only rule that is not yet mentioned is unfolding, [Unf$\downharpoonright^\mu$], which is implicitly applied by the definition of δ.

▶ **Definition 29** ($\mathsf{ProjPred}_{\mathsf{p}}$). *The boolean predicate* $\mathit{ProjPred}_{\mathsf{p}} \in G^\mu \times T^\mu \to \{0, 1\}$ *is defined as:*

$$\mathit{ProjPred}_{\mathsf{p}}(G^\mu, T^\mu) = \begin{cases} (\mathbf{1})\ PL_{\mathsf{p}}(LG(G^\mu)) = LT(T^\mu) \wedge & \\ \quad d_g(G^\mu) = d_l(T^\mu) & \textit{if } PL_{\mathsf{p}}(LG(G^\mu)) \textit{ is defined} \\ (\mathbf{2})\ 0 < d_g(G^\mu) & \textit{if } \mathsf{partOf}^\mu_{\mathsf{p}}(G^\mu) \textit{ and } \mathsf{guarded}^\mu_{\mathsf{p}}(G^\mu) \\ (\mathbf{3})\ \mathit{sat}_{\mathit{UnravelPred}}(\{\}, G^\mu) \wedge & \\ \quad \neg\mathsf{partOf}^\mu_{\mathsf{p}}(G^\mu) \wedge & \\ \quad \mathit{unfold}(T^\mu) = \mathit{end} & \textit{otherwise} \end{cases}$$

We explain the three cases of the predicate.

1. Attempt to apply a prefix rule: This requires p to be in the unfolded prefix of the global type. This is checked by requiring that PL_{p} is defined. We then apply PL_{p} to the unfolded prefix, and assert it equal to the unfolded prefix of the local type. All prefix rules require the global and local type to have equally many outgoing edges, which we check by $d_g(G^\mu) = d_l(T^\mu)$.

2. Attempt to apply a guarded rule: We rely on decidability of partOf^{μ} and $\mathsf{guarded}^{\mu}$ which is straightforward so we do not detail how[2]. All guarded-rules require the set of outgoing edges of the global type to be greater than zero, which we assert. Concretely this test corresponds to the first premise of rule $[\mathsf{B}{\downharpoonleft}^{\mu}]$, asserting its label set is non-empty.
3. Attempt to apply $[\mathsf{End}{\downharpoonleft}^{\mu}]$.

▶ **Theorem 30.** $G^{\mu} \downharpoonleft^{\mu}_{\mathsf{p}} T^{\mu}$ *iff* $\mathsf{sat}_{\mathsf{ProjPred}_{\mathsf{p}}}(\{\}, (G^{\mu}, T^{\mu})) = 1$

Proof. For ($\Longrightarrow$), we show the property for any visited list v, that is, $G^{\mu} \downharpoonleft^{\mu}_{\mathsf{p}} T^{\mu}$ implies $\mathsf{sat}_{\mathsf{ProjPred}_{\mathsf{p}}}(v, (G^{\mu}, T^{\mu})) = 1$. Proceed by functional induction on $\mathsf{sat}_{\mathsf{ProjPred}_{\mathsf{p}}}(v, (G^{\mu}, T^{\mu}))$. For ($\Longleftarrow$), for any v, it suffices to show $\mathsf{sat}_{\mathsf{ProjPred}_{\mathsf{p}}}(v, (G^{\mu}, T^{\mu})) = 1$ implies $(G^{\mu}, T^{\mu}) \in v \vee G^{\mu} \downharpoonleft^{\mu}_{\mathsf{p}} T^{\mu}$. Proceed by functional induction on $\mathsf{sat}_{\mathsf{ProjPred}_{\mathsf{p}}}(v, (G^{\mu}, T^{\mu}))$. In the second case where v is non-empty, pick the right disjunct $G^{\mu} \downharpoonleft^{\mu}_{\mathsf{p}} T^{\mu}$ and proceed by coinduction. ◀

▶ **Corollary 31.** $\mathsf{projectable}_{\mathsf{p}}(G^{\mu})$ *is decidable.*

Proof. Follows from Theorem 30 and Corollary 16. ◀

6 Mechanisation

All of our results are mechanised in Coq [6] using SSReflect [14] for writing proofs, the Paco library [18] for defining coinductive predicates, the Equations package [24] for defining functions by well-founded recursion (such as sat_{P}), and Autosubst2 [25] to generate syntax of inductive global and local types with binders represented by De Bruijn indices [10].

The mechanisation uses coinductive extensional equivalence relations to equate coinductive terms. For presentation purposes, e.g. in the conclusion of Lemma 12, we use propositional equality to equate coinductive types. These two types of equality are consistent [1].

In this section, we cover how to create predicates and relations that are defined using both inductive and coinductive inference rules, like our unravelling relation from Definition 3. We discuss how to create an inversion principle that allows us to do case analysis on predicates of the form $\mathsf{Unravels}(G^{\mu})$ which, as discussed in Section 5, is defined as $G^{\mu}\ \mathcal{R}\ \mathsf{tocoind}(G^{\mu})$. Finally, we show how we prove decidability of Unravels using sat_{P}.

Mixed inductive and coinductive definitions. The unravelling relation presented in Definition 3 uses a combination of inductive and coinductive rules, which is non-standard. We do this because it greatly simplifies our proofs and disallows unwanted derivations like the one presented in Section 3 (8) by construction. We mix inductive and coinductive rules by taking the greatest fixed point of a generating function defined as a least fixed point, a technique that Zakowski et al. [29] also have used to define weak bisimilarity of streams.

```
Definition grel := gType -> gcType -> Prop
Inductive UnravelF (R : grel) : grel := (* Generating function UnravelF *)
 | UnrF1 g gc a u : R g gc -> UnravelF R (GMsg a u g) (GCMsg a u gc)
 (* The branching rule is elided *)
 | UnrF_unf1 g gc : UnravelF R (unf1 (GRec g)) gc -> UnravelF R (GRec g) gc
 | UnrF_end       : UnravelF R GEnd GCEnd.
Definition Unravelling : grel := paco2 UnravelF bot2 (* gfp UnravelF *)
```

[2] We need to assert both $\mathsf{partOf}^{\mu}_{\mathsf{p}}(G^{\mu})$ and $\mathsf{guarded}^{\mu}_{\mathsf{p}}(G^{\mu})$ for completeness as we from $G^{\nu} \downharpoonleft^{\nu}_{\mathsf{p}} \mathsf{end}^{\nu}$ and $G^{\mu}\ \mathcal{R}\ G^{\nu}$ then can conclude the third case of $\mathsf{ProjPred}_{\mathsf{p}}$.

We represent $\mathsf{p}_1 \xrightarrow{\mu} \mathsf{p}_2 : k\langle U\rangle.G^\mu$ and $\mathsf{p}_1 \xrightarrow{\nu} \mathsf{p}_2 : k\langle U\rangle.G^\nu$ as GMsg a u g of type gType and GCMsg a u gc of type gcType respectively, where a contains roles p_1, p_2 and channel k, u is U, g is G^μ, and gc is G^ν. The terms GEnd and GCEnd represent end^μ and end^ν respectively and the function unf1 is the unfold_once function from Section 4.

UnravelF is an inductively defined relation relating global inductive types to global coinductive types. It is parameterised by a relation R of the same type where (g, gc) $\in$ UnravelF R if, after unfolding a finite number of binders from g resulting in type g', either g' = GEnd and gc = GCEnd, or g' = GMsg a u g'', gc = GCMsg a u gc'', and (g'', gc'') $\in$ R, or similarly for the elided branch case.

Intuitively, UnravelF is a generating function defined as a least fixed point and by taking the greatest fixed point of this function we obtain a hybrid inductive/coinductive relation where any occurrence of R in a premise of UnfoldF require us to take coinductive steps in our proofs and any recursive occurrence of UnfoldF requires us to take inductive steps. This allows us to do proofs like (9) where proofs are finished by circling back to previous equivalent nodes in the tree in the coinductive cases or by reaching a base case in the inductive cases. Moreover this approach forbids us from unfolding binders indefinitely since UnrF_unf1 is inductive and not coinductive.

We use paco2 from the Paco library to define Unravelling as the greatest fixed point of UnravelF. Paco stands for parameterised coinduction and paco2 F R defines the greatest fixed point of F parameterised by a binary relation R, which is equivalent to $\mathsf{gfp}(\lambda X.\ \mathsf{F}(X \cup \mathsf{R}))$. When R is the empty set this coincides with the standard greatest fixed point.

Custom inversion principles. Many proofs on inductive global types work up to unfolding. Unravelling, for instance, unravels a finite number of μ-binders at every step and our intermediate projection function $\downharpoonright^\mu_\mathsf{p}$ and sat procedure both work in a similar way. To abstract away from finite unfoldings we use the following InvPred predicate.

```
Variant InvPredF (P : gType -> Prop) : gType -> Prop :=
| HTM g a u : P g          -> InvPredF P (GMsg a u g)
| HTB gs d  : Forall P es -> InvPredF P (GBranch d gs)
| HTE       :                InvPredF P GEnd
Definition unf g := (iter (mu_height g) unf1 g).
Variant UnfoldF (P : gType -> Prop) : gType -> Prop :=
 | UnfF1 g :  P (unf g) -> UnfoldF g.
Definition InvPred : (gType -> Prop) := paco1 (UnfoldF \o InvPredF) bot.
                                                  (*function composition*)
```

We define two generating functions InvPredF and UnfoldF and generate InvPred as the greatest fixed point of their composition. The function unf corresponds to unfold from Section 4. InvPredF contains cases for all constructors of inductive global types except for $\mu\mathbf{t}$ and $\mathbf{t}$. UnfoldF unfolds the top-level μ-binders from a global type. The key insight is that InvPred(G^μ) is equivalent to asserting closedness and contractiveness of G^μ.

The inversion principle of InvPred is convenient for proving predicates P that are closed under unfolding of inductive global types, i.e. $\forall G.\ P\ \mu\mathbf{t}.G \iff P\ G[\mu\mathbf{t}.G]$, as any unfolding applied by inverting UnfoldF can similarly be applied in the goal. In particular the predicate Unravels(G^μ) is closed under unfolding and provable by inversion of UnfoldF.

Well-foundedness of sat$_P$. Lemma 23 proves decidability of Unravels. This proof is mechanised by proving decidability of InvPred(G^μ), which as we show above implies Unravels(G^μ). The invP predicate corresponds to Definition 22.

```
Definition invP g :=
match unf g with | GRec _ | GVar _ => false | _ => true end.

Definition invpred g := sat nil invP g.
Theorem InvPred_dec : forall g, InvPred g <-> invpred g = true
```

We use the Equations package to define sat_P by well-founded recursion on the decreasing measure gmeasure g V which is defined as the number of unique nodes in the graph created from g minus the cardinality of the visited set V. The successor function δ_g is implemented by nextg : gType -> seq gType.

```
1  Definition gmeasure (g : gType) (V : seq gType) :=
2    size (rep_rem V (undup (enumg g))).
3  Lemma closed_enum  : forall g0 g1 g2, g1 \in nextg (unf g) ->
4                                         g2 \in enumg g1  -> g2 \in enumg g.
5  Equations sat (V : seq  gType)  (P : gType -> bool)
6                        (g : gType) : bool by wf (gmeasure g V) :=
7   sat V P g with (dec (g \in V)) => {
8    sat  _ _ _ in_left  := true;
9    sat V P g  in_right :=  (P g) &&
10                           (foldInMap (nextg (unf g))
11                                       (fun g' _ => sat (g::V) P g')) }.
```

Defining sat generates one obligation that must be proved to show termination. If we write gmeasure g V as $M(g, V)$, then we must show it is decreasing for arguments to the recursive call, i.e. that $M(g', \{g\} \cup V) < M(g, V)$

Using a variant of the familiar map on inductive lists called foldInMap our obligation is enriched with the assumption that $g' = \delta(g, i)$ for some $0 < i \leq d_g(g)$. The boolean wrapper dec further enriches the obligation with the case of the if-statement, $g \notin V$.

What must be proven in this obligation is slightly different from the termination argument in Section 5 which relied on the finiteness of a graph's nodes. The obligation instead relies on a lemma closed_enum (l. 3). The lemma states that the enumerations of a global types continuations, will all be part of the initial global types enumeration. The proof of this lemma is short, less than 100 lines.

The full termination proof for sat is short (about 250 lines) and the approach is general. The mechanisation also proves termination of the decision procedure for membership in $\upharpoonright^{ind}$. This task only requires adapting the algorithm to pairs of terms. This termination proof is also short. The conciseness is due to the space of continuations being computed by structural recursion by enum. This makes it straightforward to prove substitution properties about it by induction on syntax.

7 Related Work and Discussion

Related Work. Ghilezan et al. [13] are the first to introduce coinductive projection on coinductive global and local types. They use it to show soundness and completeness of synchronous multiparty session subtyping. A key difference is that whereas we represent the infinite unfolding of a μ-type as a coinductive type, they represent it as a partial function. Projection on μ-types is then defined indirectly in terms of the coinductive projection of their corresponding partial functions. Because of this indirect definition, their projection is not computable. Our intermediate projection $\upharpoonright^{\mu}$ is similar to their projection on μ-types.

However, ours is defined with inference rules stated directly on the μ-types which is why we can decide membership and thus compute projection. Castro-Perez et al. [7] use coinductive projection to express their meta theory about multiparty session types. Their main result is trace equivalence between processes, coinductive local types and coinductive global types, which they mechanise in Coq. Like us, they show soundness of their projection on μ-types. Their projection is however not complete, which is what inspired us to investigate approaches to sound and complete projection. A consequence of their projection on μ-types not being complete, is that there are many inductive global types that have the trace equivalence property, but must be excluded since their projection is undefined. Jacob et al. [19] show deadlock and leak freedom of multiparty GV, an extension of the functional language GV [12, 28]. They use coinductive projection to define when local types are compatible and do not define a projection on μ-types. Other work has formalised the notion of projection in Coq. Cruz-Filipe et al. [9, 8] formalise syntax and semantics of tail-recursive choreographies and a projection that includes full merge. However, this work does not approach coinductive syntax and therefore does not show any soundness and completeness results.

Our graph algorithm from Section 5 implements a procedure proposed by Eikelder [26]. This work provides several algorithms for deciding recursive type equivalence that, like ours, use predicates on reachable nodes of a graph. Also, our proof of termination is quite similar to theirs. However, they define the set of reachable states as set comprehension, whereas we constructively produce a list of nodes. Similarly, showing their set comprehension is finite, boils down to substitution lemmas. Unlike ours, their work has not been mechanised in a proof assistant/theorem prover. The idea of defining the space of continuations for global and local type as an explicit enumeration is inspired by Asperti [3] who mechanise a concise proof of regular expression equivalence in the Matita theorem prover [4]. They do this by a new construction called pointed regular expressions. Essentially, this adds marks to a regular expression, such that one can encode state transitions by moving marks. This makes computing reachable configurations as trivial as computing all markings.

We define unravelling using a mix of inductive and coinductive rules. In Section 6, we make this precise by defining unravelling as the greatest fixed point of a generating function itself defined as a least fixed point. Zakowski et al. [29] use the same technique to define a weak bisimilarity on streams.

The primary focus of this work is on global types. Scalas and Yoshida [23] propose a more general approach that shows that properties such as deadlock freedom can be derived directly on local types without the need for global types and the corresponding projection. However, their approach misses the main advantage provided by global types which is providing a specification (blueprint) of the used protocols.

Discussion and Future work. This work is part of the MECHANIST project that aims at mechanising the full theory of multiparty asynchronous session types [17]. Our next step is to mechanise a proof of semantic equivalence between global types and their projections to local types through $\mathsf{proj_p}$. Semantic equivalence is a property similar to trace equivalence which Castro-Perez et al. [7] mechanised. However, there are some key differences in our objectives. Their main result is Zooid, a tool that extracts certified message-passing programs, which is why their process syntax differs significantly from the original syntax by Honda et al (e.g., no parallel composition). Instead, we aim at mechanising the exact process calculus presented by Honda et al.. As the meta theory in Castro-Perez et al. [7] is independent of their projection function, it would also be interesting future work to adapt $\mathsf{proj_p}$ to their setting. Finally, $\mathsf{proj_p}$ implements the restrictive plain merge but related work also uses full merge [13, 8]. It would be interesting to define a binder-agnostic projection using full merge.

8 Conclusions

Projection is a function that maps global types to local types. The projections found in the literature impose syntactic restrictions that make them incomplete with respect to coinductive projection. This work shows the existence of a decidable projection that is sound and complete. Our procedure works in two phases: first a decision procedure tests a soundness property and, if successful, a second procedure translates the global type to a local type. The latter is very similar to the existing projections in the literature. The novelty of our work is in the decision procedure. All results have been mechanised in Coq.

References

1 Coinductive types and corecursive functions. `https://coq.inria.fr/refman/language/core/coinductive.html`. Accessed: May 2023.

2 Session types in programming languages: A collection of implementations. `http://www.simonjf.com/2016/05/28/session-type-implementations.html`. Accessed: May 2023.

3 Andrea Asperti. A compact proof of decidability for regular expression equivalence. In *proceedings of ITP*, volume 7406 of *LNCS*, pages 283–298. Springer, 2012. `doi:10.1007/978-3-642-32347-8_19`.

4 Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. The Matita interactive theorem prover. In *Proceedings of CADE*, volume 6803 of *LNCS*, pages 64–69. Springer, 2011. `doi:10.1007/978-3-642-22438-6_7`.

5 Andi Bejleri and Nobuko Yoshida. Synchronous multiparty session types. In *Proceedings of PLACES*, volume 241 of *ENTCS*, pages 3–33. Elsevier, 2008. `doi:10.1016/j.entcs.2009.06.002`.

6 Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. `doi:10.1007/978-3-662-07964-5`.

7 David Castro-Perez, Francisco Ferreira, Lorenzo Gheri, and Nobuko Yoshida. Zooid: a DSL for certified multiparty computation: from mechanised metatheory to certified multiparty processes. In *Proceedings of PLDI*, pages 237–251. ACM, 2021. `doi:10.1145/3453483.3454041`.

8 Luís Cruz-Filipe, Fabrizio Montesi, and Marco Peressotti. Certifying choreography compilation. In Antonio Cerone and Peter Csaba Ölveczky, editors, *Proceedings of ICTAC 2021*, volume 12819 of *Lecture Notes in Computer Science*, pages 115–133. Springer, 2021. `doi:10.1007/978-3-030-85315-0_8`.

9 Luís Cruz-Filipe, Fabrizio Montesi, and Marco Peressotti. Formalising a turing-complete choreographic language in coq. In Liron Cohen and Cezary Kaliszyk, editors, *Proceedings of ITP 2021*, volume 193 of *LIPIcs*, pages 15:1–15:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.ITP.2021.15`.

10 Nicolaas Govert de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972. `doi:10.1016/1385-7258(72)90034-0`.

11 Romain Demangeon and Nobuko Yoshida. On the expressiveness of multiparty sessions. In *Proceedings of FSTTCS*, volume 45 of *LIPIcs*, pages 560–574, 2015. `doi:10.4230/LIPIcs.FSTTCS.2015.560`.

12 Simon Gay and Vasco Vasconcelos. Linear type theory for asynchronous session types. *Journal of Functional Programming*, 20(1):19–50, 2010. `doi:10.1017/S0956796809990268`.

13 Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. *Journal of Logical and Albegraic Methods in Programming*, 104:127–173, 2019. `doi:10.1016/j.jlamp.2018.12.002`.

14 Georges Gonthier, Assia Mahboubi, and Enrico Tassi. A small scale reflection extension for the coq system, 2016. URL: `https://inria.hal.science/inria-00258384`.

15 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *Proceedings of ESOP*, volume 1381 of *LNCS*, pages 122–138. Springer, 1998. `doi:10.1007/BFb0053567`.

16 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of POPL*, pages 273–284. ACM, 2008. `doi:10.1145/1328438.1328472`.

17 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *Journal of the ACM*, 63(1):9:1–9:67, 2016. `doi:10.1145/2827695`.

18 Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. The power of parameterization in coinductive proof. In *Proceedings of POPL*, pages 193–206. ACM, 2013. `doi:10.1145/2429069.2429093`.

19 Jules Jacobs, Stephanie Balzer, and Robbert Krebbers. Multiparty GV: functional multiparty session types with certified deadlock freedom. *Proceedings of the ACM on Programming Languages*, 6(ICFP):466–495, 2022. `doi:10.1145/3547638`.

20 Jean-Baptiste Jeannin, Dexter Kozen, and Alexandra Silva. Cocaml: Functional programming with regular coinductive types. *Fundamenta Informaticae*, 150:347–377, 2017. `doi:10.3233/FI-2017-1473`.

21 The Coq development team. The Coq Proof Assistant. `https://coq.inria.fr`. Accessed: May 2023.

22 Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

23 Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *Proc. ACM Program. Lang.*, 3(POPL):30:1–30:29, 2019. `doi:10.1145/3290343`.

24 Matthieu Sozeau and Cyprien Mangin. Equations reloaded: High-level dependently-typed functional programming and proving in coq. *Proceedings of the ACM on Programming Languages*, 3(ICFP):86:1–86:29, 2019. `doi:10.1145/3341690`.

25 Kathrin Stark, Steven Schäfer, and Jonas Kaiser. Autosubst 2: reasoning with multi-sorted de bruijn terms and vector substitutions. In *Proceedings of CPP*, pages 166–180. ACM, 2019. `doi:10.1145/3293880.3294101`.

26 Huub ten Eikelder. Some algorithms to decide the equivalence of recursive types. `https://pure.tue.nl/ws/files/2150345/9211264.pdf`, 1991. Accessed: May 2023.

27 Rob van Glabbeek, Peter Höfner, and Ross Horne. Assuming just enough fairness to make session types complete for lock-freedom. In *Proceedings of LICS*, pages 1–13. IEEE, 2021. `doi:10.1109/LICS52264.2021.9470531`.

28 Philip Wadler. Propositions as sessions. In *Proceedings of ICFP*, pages 273–286. ACM, 2012. `doi:10.1145/2364527.2364568`.

29 Yannick Zakowski, Paul He, Chung-Kil Hur, and Steve Zdancewic. An equational theory for weak bisimulation via generalized parameterized coinduction. In *Proceedings of CPP*, pages 71–84. ACM, 2020. `doi:10.1145/3372885.3373813`.