Minimal Structures for Program Analysis and Verification

Alen Arslanagić

Supervisors:

Prof. dr. Jorge A. Pérez
Prof. dr. Gerard Renardel de Lavalette

April 5, 2023

Abstract

Software systems are ubiquitous in today's world; their reliability and correctness are a key societal concern. Their engineering is becoming increasingly complex as software is expected to provide versatile functionalities in a myriad of contexts. As a result, the conception, design, and validation of software remains a very complex task.

Modern software consists of multiple components, usually separately developed by different vendors. In this context, the use of *compositional approaches* to software design provides a way of taming the complexity of developing correct programs and systems. Ideally, a system's correctness can be established by ensuring that each of its components conforms to a given local *specification*; then, by establishing the compatibility between these components one should obtain a correctness guarantee for the entire composed system.

This thesis concerns the *mathematical structures* used for the specification, analysis and (compositional) verification of programs. As such, these structures are inherently abstract: they capture the essential elements of the (compositional) analysis of software. We focus on structures that describe the *interface* of a single software component with respect to the rest of the system. These structures are commonplace; they are advantageous because they come with (rigorous) techniques for verifying that (i) a component's implementation respects its specification and (ii) a collection of components is composed correctly.

While the use of formal structures for specifying and analyzing programs is increasingly widespread, it is important to establish to what extent their expressivity and flexibility leads to potential overheads in program analysis and verification. This thesis investigates the follow research question: *to what extent existing structures for program analysis and verification admit simpler or even minimal formulations?* This is a pertinent question from a fundamental standpoint (as the theory underlying these structures can be simpler), but also for implementations based on them. We focus on two existing structures for compositional program design, namely *session types* for communicating programs and *typestates* for object-oriented programs. They provide techniques for statically enforcing so-called *temporal properties* of programs:

Typestates are a well-established technique for the verification of object-oriented programs. Here an interface's specification should prescribe valid interactions with an object: this includes operations but also valid sequences of invocations. Typestates refine the concept of a type: whereas a type denotes the valid operations on an object, a typestate denotes operations valid on an object in its *current program context.* Thus, they are fundamental structures in ensuring the correct use of code contracts for objects and APIs. Typestate analyses rely on Deterministic Finite Automata (DFAs) to specify properties of an object.

Session types are a type-based approach to the verification of message-passing programs. They have been much studied as type systems for programming calculi such as the π-calculus and have been integrated into mainstream languages such as Java, Scala, Go, and Rust. Session types specify communication structures essential for program correctness; a session type says what and when should be exchanged through a channel. Central to session-typed languages are constructs in types and processes that specify *sequencing*, which allow to express structures within protocols.

While different, session types and typestates share a fundamental trait: they specify temporal properties of program entities—roughly speaking, they both specify *when* a particular operation is enabled for execution. Accordingly, the contributions of the thesis come in two parts.

- We propose a *lightweight* approach to typestate analysis. Our proposal consists of (i) a *lightweight* specification language and (ii) an associated compositional analysis algorithm. While existing approaches to typestate analysis rely on DFAs to specify properties of an object, our approach induces and relies on a strict sub-class of DFAs (dubbed BFAs) that enables efficient *bit-vector* operations while retaining sufficient expressivity for the most object properties prescribed by typestates in practice. By implementing our lightweight approach in the static analyzer INFER, we demonstrate considerable performance and usability benefits when compared to existing techniques.

- We propose a minimal formulation of session types, dubbed *Minimal Session Types* (MSTs). In a nutshell, MSTs eschew sequencing at the level of types, in line with the (relatively simple) typing disciplines available in mainstream programming languages. We develop MSTs for two different programming calculi: one features higher-order concurrency, the other features first-order concurrency (name passing). We prove that ordinary session types are representable as MSTs by exploiting sequencing in processes. Because we consider both first- and higher-order concurrency, we show that our minimal formulation of session types is robust irrespectively of the kind of exchanged objects (names or processes).

All in all, by considering existing structures for program specification, this thesis demonstrates that they admit minimal formulations, which distill essential features for analyzing and verifying programs. In turn, the thesis provides evidence that these minimal formulations can lead to theoretical and practical benefits for program analysis and verification. On the one hand, MSTs show that sequencing is not an indispensable feature, thus allowing the embedding of session types into programming languages and tools that do not support sequencing natively. On the other hand, BFAs show that the full expressivity of DFAs can be cumbersome for specifying many practical code contracts and that it leads to sub-par performance of associated analysis algorithms.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context and Motivation

Software systems are ubiquitous in today's world; their correctness and reliability are a key societal concern. In fact, question 115 in the *Nationale Wetenschapsagenda*[1] (the Dutch National Research Agenda) is as follows:

How can we build and maintain future-proof software?

In recent years, software engineering has gained in complexity due to multiple factors: software systems are expected to provide complex and versatile functionalities while supporting multiple deploying platforms and heterogenous computing environments.

A common engineering approach for taming such complexity is *compositional design*. This way, software systems are constructed by combining multiple components, usually developed separately. Each component is meant to provide a specific functionality; its intended usage is commonly specified only as an informal documentation and/or illustrated by code examples. Such specifications are useful in early stages of development. Informal in nature, they are usually ambiguous or incomplete and there is no formal guarantee that a client code adheres to them. Further, as components can interact between them in complex ways, their integration is a potential source of insidious bugs. Hence, ultimately, we would like precise specifications for individual components and a correctness guarantee for a composed system.

In this day and age, it is widely accepted that *formal methods* can play a decisive role towards reliable software verification and enhanced software quality. Formal methods broadly refer to mathematical tools that allow the precise description of software and a rigorous reasoning about its properties. Roughly speaking, these methods provide ways of producing a *formal specification* and techniques to verify that a program *complies* with such a specification. There is a myriad of such techniques tailored to different settings and requirements, including model checking, abstract interpretation, deductive verification, and type systems. There is a wide range of applicability of formal methods: from source code to models of software and hardware architectures. Besides code verification, formal methods also include techniques for program synthesis and automatic program repair. Success stories for formal methods in practice include systems and frameworks such as, e.g., CompCert (the formally verified compiler for C [47]), Infer (the static program analyzer for C and Java [12, 1]), Astrée (the static analyzer for C and C++, based on abstract interpretation [13]), and ESC/Java (the static checker for Java [26]).

We are concerned with formal methods from the perspective of *program analysis and verification*. Different techniques for program analysis can be distinguished by the time they are used—most notably, whether an analysis is employed during compile-time or at run-time. *Static (program) analysis* encompasses techniques that are performed without executing a program, that is, they judge a program only by inspecting its source code. In contrast, *dynamic (program) analysis* refers to techniques that analyze a program during its execution. Static and dynamic techniques are designed to accomplish fundamentally distinct goals, and come with different benefits and drawbacks. In this thesis, the focus is on techniques for static (program) analysis.

[1]See `https://2.wetenschapsagenda.nl/publicatie/nationale-wetenschapsagenda-nederlands/`

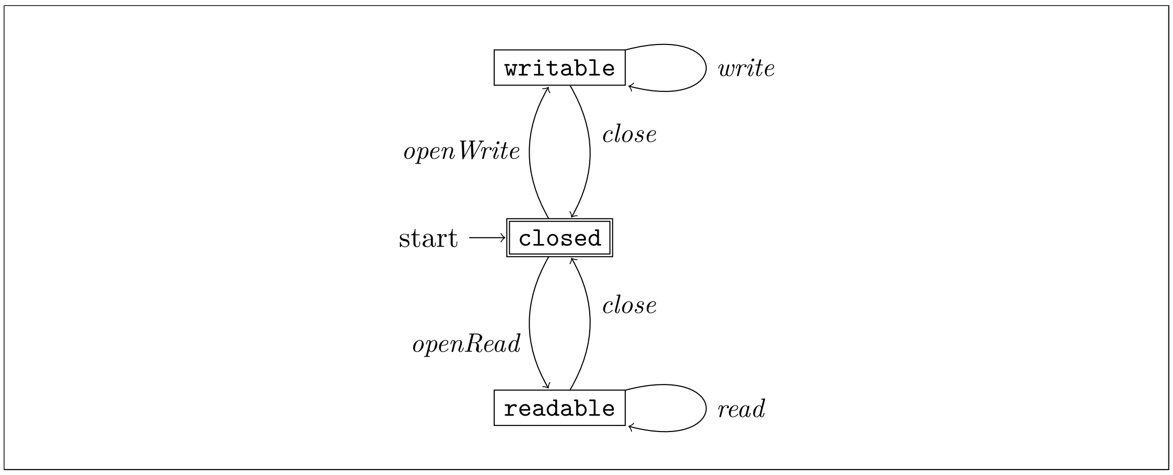


Figure 1.1: State diagram of a file contract

Because static techniques target (non-trivial) semantic properties, a fully precise analysis is out of reach (a consequence of the undecidability of the Halting problem). A solution is to relax the precision of the analysis while aiming at results that are still useful for improving software development. Put differently, there is a compromise that concerns forgoing either soundness and/or completeness of the analysis. The typical situation is to have techniques that are sound but incomplete; roughly speaking, this ensures that accepted programs conform to a given specification, at the price of rejecting some valid programs. This thesis is concerned with a particular class of static techniques, namely **type systems**.

Type systems are formal program specifications that are machine-checked, leveraging types, abstractions of a program's functionality. Types classify program (data) values based on their semantic properties. For example, in imperative languages, a data type is an abstraction of a concrete memory layout of a value. A type prescribes valid uses of a program value—the operations that can be used on a value. The process of analyzing a program with respect to its type specification is called *type-checking*.

Types allow compositional (i.e., modular) reasoning about programs (in the sense discussed earlier), and specify component functionality that is not context-dependent. This way, types express that a component should have the same functionality in all contexts. In turn, this feature enables compositional type-checking—a component's implementation is checked in isolation, then its use can be checked only on type-level. Type systems have been widely studied and developed for functional languages such as OCaml or Haskell, where functions provide a convenient abstraction for composition.

Different program contexts ask for different type-based abstractions. Many programming models rely heavily on *interactions* between components. In this setting, a component's functionalities are not fixed, but vary depending on prior interactions. Accordingly, the properties of a component that we would like to check vary over time; in this thesis, we shall refer to them as **temporal properties**. As simple examples, a file must be created and opened *before* it can be read from or written to; failure to respect this intended order leads to inconsistencies. A state diagram that represents the intended behaviour of a file object is given in Figure 1.1. Similarly, a typical online protocol consists of communication exchanges that must be performed in a predetermined order; this way, e.g., shipping details must be sent before a payment can be carried on (cf. Figure 1.2). Here again, not respecting this order can have a disastrous ripple effect on the state of all protocol participants (denoted "Buyer", "PaymentService" and "Seller" in the figure).

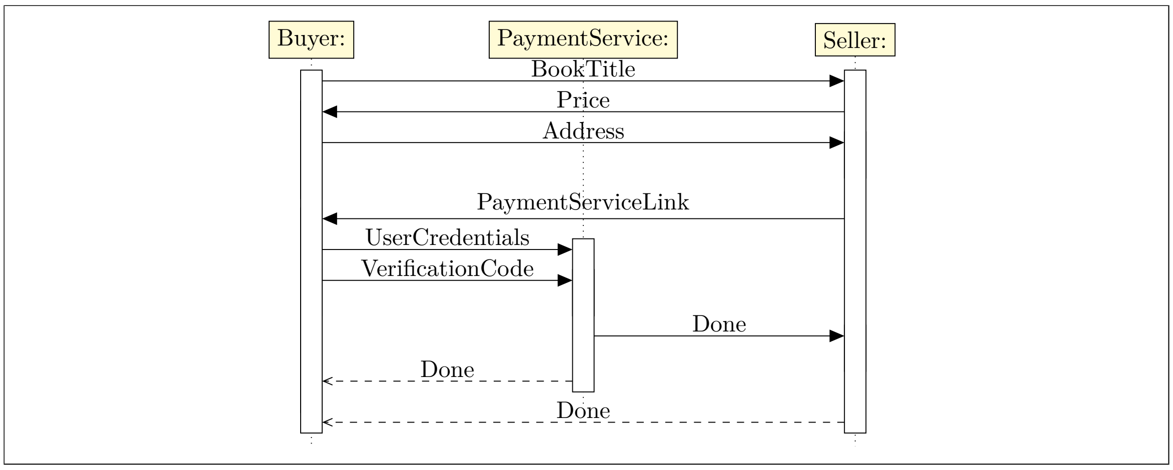


Figure 1.2: Sequence diagram of an e-commerce protocol

1.2 Enforcing Temporal Properties: Two Static Approaches

This thesis investigates two type-based abstractions that model temporal properties of programs, namely *typestates* and *(binary) session types*. Conceptually, these two type systems fall into the category of behavioural types [32], which classify the *behaviour* of a program component. This sets behavioral types in contrast with traditional data types, which describe *static* properties of a component that cannot be changed during its lifetime. We see typestates and session types as two manifestations of a very similar concept that arise in two different programming settings. We briefly describe these two type-based abstractions.

1.2.1 Typestates

In object-oriented programming (OOP), an object is the main program component, an encapsulation of data and functions. In most mainstream OOP languages, an object is stateful: its state is given either by internal (enclosed) data or by external resources it manages (e.g., a file or stream handlers). The correct and safe usage of an object depends on its current state, which dictates *available operations*; in turn, these operations can change the underlying state. We expect a precise specification to prescribe valid sequences of object operations.

Here comes the concept of *typestates*, which was first introduced for imperative languages without objects [55]. A typestate refines the concept of type: whereas a type denotes a set of operations for a program variable, a typestate specifies a subset of available operations in a particular program context. This concept has been further developed for OOP [19], where a typestate can be viewed as an abstraction of an object memory state—each method invocation can potentially change an object's typestate. This is explicitly given as pre- and post-conditions on methods: each method requires its receiver object to be in a typestate of a pre-condition, and ensures that it will leave the object in a typestate of a post-condition. Recall the state diagram in Figure 1.1, for the file object; here states in the diagram correspond to typestates, while transitions represent method invocations. The object is initialized in typestate closed; then, calling a method openWrite leads it to typestate writable in which we can repeatedly write into the file. Similarly, a closed file can be opened for reading (as depicted by readable typestate). Finally, once writing/reading is done, it must be ensured that a method for closing a file is invoked.

It is natural to relate this vision of typestates with finite state machines, in particular

Deterministic Finite Automata (DFAs, in the following). Static typestate checkers define the analysis property using a DFA. The abstract domain of the analysis is the set of states; each operation on the object modifies the set of reachable states. If the set of abstract states contains an error state, then the analyzer warns the user that a code contract may be violated. Widely applicable and conceptually simple, DFAs are the de facto model in typestate analyses.

Typestates are particularly useful in the context of application programming interfaces (APIs): a collection of software components provided by separate developers. Scalable software development relies highly on reusable APIs. The intended usage of a component is usually only given as an informal *code contract* as part of API specifications. An important aspect of API specifications is related to temporal properties, which in this case specify dependencies of operations on components. This is usually only informally described as a part of an API documentation and/or given by dynamic checks (assertions). A client program using APIs must closely follow their contracts. This can be particularly tricky: the protocols underlying APIs can be intricate, and a program usually consists of multiple objects that can have interdependencies. As a result, the integration of APIs components is a source of subtle and insidious bugs. This stresses the importance of typestate analysis techniques in the (automated) enforcement of correct API usages.

1.2.2 Session Types

Modern distributed systems are composed of individual programs that run concurrently and coordinate with partners by exchanging messages. As such, these systems heavily rely on *message-passing* concurrency. The kind of messages and the order in which they are exchanged should follow some pre-established scheme. Hence, the natural abstraction for verification in this setting is given by the *protocols* that describe the structure of message exchanges.

Session types are a type-based approach to the specification and verification of communicating programs. The main object of interest for analysis and verification is that of a (communication) channel: in order for programs to interact, they must establish a connection along a channel. A session type says what and when should be exchanged through a channel. Central is the notion of a *session* between two parties, which allow them to perform a given sequence of actions (a protocol) without external interferences.

Session types have been introduced and much studied as type systems for programming calculi, in particular the π-calculus [30]. In the π-calculus, session types are assigned to *names*, which can be assimilated to channels; a process can have multiple names to interact with other processes, and each of these names has a corresponding session type. In this setting, type-checking can be used to enforce that processes enjoy relevant safety and liveness properties.

After their introduction in the 90's, session types have been extended in many directions and they have been integrated into mainstream languages such as Java, Scala, Go, and Rust. We find, for instance, multiparty session types [31], and extensions with dependent types, assertions, exceptions, and time (cf. [21, 32] for surveys). All these extensions seek to address natural research questions on the expressivity and applicability of session types theories.

Central to session-typed languages are constructs that specify *sequencing*, which conveniently express structures within protocols. In turn, these structures are tightly related to temporal properties of interest. Consider, for example, the session type:

$$S = ?(\mathsf{int}); ?(\mathsf{int}); !\langle \mathsf{bool} \rangle; \mathtt{end}$$

In S, sequencing (denoted ';') allows us to specify a protocol for a channel that *first* receives

(?) two integers, *then* sends (!) a boolean, and *finally* ends. As such, S could type a service that checks for integer equality.

Sequencing in types goes hand-in-hand with sequencing in processes, which is specified using the prefix construct (denoted '.'). As a simple example, the π-calculus process $P = s?(x_1).s?(x_2).s!\langle x_1 = x_2\rangle.\mathbf{0}$ is an implementation of the equality service: it *first* expects two values on name s, *then* outputs a boolean on s, and *finally* stops. Thus, name s in P conforms to the session type S. Session types can also specify sequencing within labeled choices and recursion; these typed constructs are also in close match with their respective process expressions.

1.3 Research Questions

While formal methods provide strong mathematical guarantees for ensuring software correctness, it is well-known that their adoption in practical software engineering lags behind (see, e.g., [25]). Interestingly, some of the distinguishing merits of formal methods, such as precision and thoroughness, can also represent burden and friction for practitioners. Indeed, the use of sophisticated mathematical languages and notations for specification comes at cost of an increased learning curve for developers. In turn, analysis and verification techniques associated with such formalisms come with their own complexity. It is not too difficult to imagine practical scenarios in which programmers would favor analysis techniques that are unsound but *fast* over techniques that are *probably correct* but slow.

The effective integration of formal methods in practical software development is a relevant (and difficult) challenge that surely goes beyond the scope of this thesis. Still, it provides a solid motivation for the specific research questions that we consider here. Indeed, the present thesis aims to establish to what extent the expressivity and flexibility associated with formal methods can potentially lead to redundancies in program analysis and verification. As already stated, we are specifically interested in typestates and session types, and in their associated challenges for the verification of temporal properties.

Given this, there is both theoretical and practical interest in investigating *simpler* formulations of typestates and session types. From a theoretical perspective, studying simpler variants of these formalisms can lead to a principled distillation of their core features, streamline the definition of their essential notions, and, ultimately, shed new light on the fundamental correctness properties they can enforce on programs. From a practical standpoint, simpler formalisms can be advantageous, for several reasons:

- They can unlock the development of more effective specification languages and implementation machineries, which in turn can help improve analysis performance.
- Simpler specification languages can (partially) relieve developers from reasoning in terms of rigorous but abstract mathematical models, therefore lowering the learning requirements.
- Simpler formalisms can be a useful tool for sound yet pragmatic program analysis, as they can provide a reference for the specification and analysis of verification properties that are impractical or even unfeasible to enforce in the fully-fledged variant of the corresponding formalism.
- Finally, simpler formalisms can help bridge the gap between the programming languages and tools developed for research purposes and those used in software industry. Indeed, simpler formalisms can enable the applicability of formal methods in a wider class of existing programming languages.

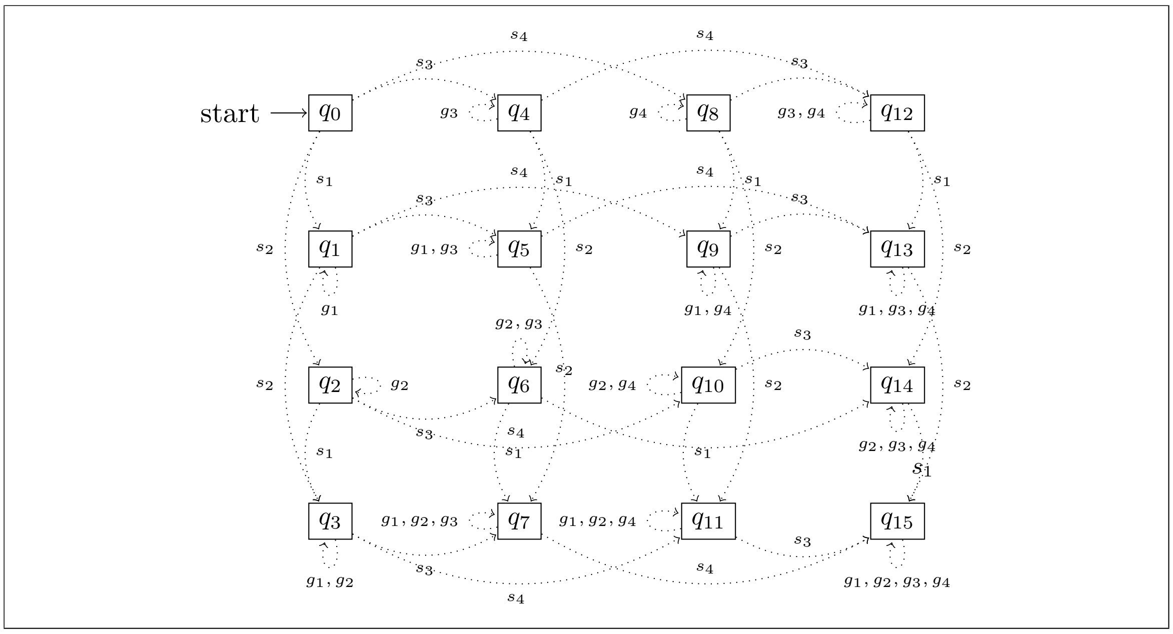


Figure 1.3: State diagram of a DFA-based setter/getter contract (case $n = 4$, with 16 states and 64 transitions).

1.3.1 A Challenge for Typestates

Current Limitations Typestates for OOP are very useful for expressing and enforcing temporal properties. As already mentioned, typestate analysis resorts to DFAs as underlying model. Unfortunately, there are many practical temporal properties for which a DFA-based contract can be cumbersome or impractical to produce. Even worse, the analysis of such contracts can be infeasible for their integration into *low-latency environments*, such as, e.g., Integrated Development Environments (IDEs). In this context, to avoid noticeable disruptions in the users' workflow, the analysis should ideally run *under a second* [3]. However, relying on DFAs jeopardizes this goal, as it can lead to scalability issues. That is, even though DFAs provide an expressive reference model for typestate analysis, such expressivity can lead to sub-par performance and usability.

To illustrate these limitations, consider the representative example of a class with four setter/getter method pairs, where each setter method *enables* a corresponding getter method and then *disables* itself; the intention is that values can be set up once and accessed multiple times. The associated DFA contract has 2^4 states, as any subset of getter methods can be available at a particular program point, depending on previous calls. Additionally, the full DFA-based specification requires as many as 64 state transitions. To see this, in each of 16 states we have 4 transitions available such that the setter and getter methods enabled are complementary (cf. Figure 1.3). In the general case, contracts in which each of the n methods *enables* another one and then *disables* itself, can lead to a DFA with 2^n states. Even with a small n, as in Figure 1.3, such contracts are impractical to codify manually and are likely to result in sub-par performance.

This kind of enable/disable properties are referred to as *'may call'*. Interestingly, the specification of common *'must call'* properties can also result in prohibitively large DFA state-space. As an example, consider a class that has m pairs of methods for acquiring/releasing some resources. The contract should ensure that all acquired resources are released before an object is destructed. Because states would need to track unreleased resources, the contract requires 2^m DFA states.

Additionally, the class inheritance for DFA-based typestates represents a challenge. First, reasoning about *contract subtyping* in terms of DFA, as required by class inheritance relations, can be particularly tricky. Intuitively, a subclass contract must be a *at least as permissive* as its superclass contract. More precisely, a set of allowed sequences of a subclass contract must subsume that of its superclass contract. For this we need a DFA subsumption algorithm which is typically expensive, and, again, it crucially depends on the number of states.

An Observation Interestingly, many practical contracts do not require the full expressiveness of a DFA. This insight is the leading motivation for the investigation of alternative typestate systems that can offer scalability benefits over DFA-based techniques.

To elaborate on this observation, we briefly recall main concepts of *interprocedural analysis*, the class of static techniques that aims at analyzing programs consisting of multiple functions and function calls. In the sense of the abstract interpretation framework, the main ingredients required are an *analysis domain*, a *transfer function*, and a *join function.* Intuitively, a domain defines an abstraction of a concrete program state; a transfer function interprets program instructions in the domain; and a join function joins abstract states of different program branches.

There are different approaches to interprocedural analysis, with different implications for the scalability of analyses. The basic approach is to inline function bodies—essentially transferring a problem to the intraprocedural analysis. This is an inefficient approach, as it can lead to redundancies. What one would like is a *compositional* interprocedural analysis that treats procedure bodies separately, producing so-called *procedure summaries.* These summaries can then be used for analysing code invoking procedures.

In this context, the so-called class of *distributive analysis* is particularly interesting, as it admits a fully precise interprocedural analysis in polynomial time, provided a finite-state domain and a distributive transfer function (see IFDS [52]). While typestate analysis satisfies these requirements, an efficient compositional analysis crucially depends on the number of states: in the worst-case it takes $|Q|^3$ operations per a method invocation, where Q is the set of states of the underlying DFA. To see why this is the case, we may notice that a function can be invoked in any state—thus, we need to analyze a function with every state as a potential entry state. Furthermore, this per-state analysis must deal with subsets of states. Thus, contracts that induce a large state space can severely impact the performance of the compositional analysis.

Given these considerations, the challenge is to devise a *lightweight* typestate analysis that satisfies two requirements: (i) it should be more scalable than usual DFA-based analyses, such that it is suitable for integration in low-latency environments , and at the same time (ii) it must be expressive enough to specify and enforce many commonly used contracts (such as for File, Socket, and Stream). Therefore, we formulate our first research question:

(Q1) *Is there a class of typestates that is more scalable than DFA-based analyses (in specification and performance) while retaining sufficient expressivity?*

1.3.2 A Challenge for Session Types

As discussed above, sequencing is the most prominent feature of session types. In the context of process calculi, such as π-calculus, sequencing is very natural as well: sequencing in session types goes hand-in-hand with sequencing in processes. Moreover, sequencing is an essential construct in imperative languages, where it specifies the intended execution order of operations.

However, in a broader, practical setting, sequencing is quite uncommon as a construct for (mainstream) programming languages; in fact their associated type systems do not natively support it. For instance, the Go programming language offers primitive support for message-passing concurrency; it comes with a static verification mechanism which can only enforce that messages exchanged along channels correspond with their declared payload types—this is insufficient to ensure correctness properties associated with the ordering of messages and protocol structure. This observation has motivated the development of advanced static verification tools based on session types for Go programs; see, e.g., [49, 45].

In this context, sequencing in implementations of session types requires non-trivial, language-specific machineries, which may represent a hurdle for the practical adoption of session types. It is worth noticing that all other features of session types, such as labelled choice and recursion constructs, have corresponding features in type systems of most programming languages. Thus, the sequencing construct (and its interplay with other constructs) stands out as the only feature that is not natively supported.

It is interesting to investigate whether we can forgo sequencing at the level of types. Because sequencing is such a distinctive feature, it appears fundamental to aim at retaining the full expressivity of session types. This indicates that a sequencing information should not be discarded, but should be recovered by some other means. This raises a foundational question: *to what extent session types can admit simpler formulations that dispense with the sequencing construct?*

This questions requires a rigorous approach that explains precisely what it means dispensing with sequencing, and how it can be recovered. As hinted at above, such an approach is beneficial in that it can reveal new insights on the relation between the sequencing construct and other constructs, such as labelled choices and recursion. This in turn can potentially help in streamlining other theories of session types and their associated verification frameworks. This discussion leads us to our second research question:

(Q2) *Is there a simpler formulation of session types that dispenses with sequencing construct, while retaining full expressivity?*

1.4 Our Approach

This thesis addresses (Q1) and (Q2) by presenting new, alternative formulations of typestates and session types. Our formulations are meant to be *simpler* or even *minimal* with respect to their corresponding full-fledged counterparts. Clearly, there is a tension between simplicity and convenience, in the sense that the simpler formulations should nevertheless bring concrete benefits (conceptual and/or practical) for specification and analysis. Considering this, we do not limit to devise new formulations but also seek to establish a precise relationship between our formulations and the original ones. This amounts to elucidating whether our formulations partially or fully characterize the analysis/verification capabilities of the original formalism. Put differently, we are interested in whether moving to a simpler formulation entails or not an expressivity loss and under which conditions. Because both typestates and session types rest upon formal foundations, we are uniquely positioned to provide this kind of precise results.

To accomplish this, we proceed as follows. First, we identify features that appear to be responsible for the intrinsic complexity in the analysis/verification based on typestates and session types. Then, we define alternative formal definitions that partially or completely disregard those features. Finally, we study the resulting *minimal structures* and their formal properties; in particular, we determine to what extent these features are indispensable and what their omission precisely entails.

1.4.1 Typestates

In response to (Q1), we devise a new *lightweight* typestate language that corresponds to a strict sub-class of DFAs. We shall devise a typestate system with the following features: (i) a lightweight typestate language that enable succinct and scalable specifications; (ii) a scalable compositional analysis (in the sense discussed earlier); and (iii) an efficient contract subtyping algorithm.

Lightweight typestate language Our lightweight typestate language relies on the following central insight: many practical contracts only specify *local* method dependencies. For enable/disable contracts (as described in Section 1.3.1), the method dependencies are *local* to a subset of methods: a enabling/disabling relation is established between pairs of methods. By definition, in DFA-based approaches are have a global standpoint, and so local method dependencies can impact transitions of unrelated methods. Thus, using DFAs as underlying model for contracts that specify dependencies that are local to each method (or to a few methods) is redundant and/or prone to inefficient implementations.

Our lightweight annotation language succinctly encodes temporal properties by describing only local method dependencies, thus avoiding an explicit DFA specification. Our specifications define code contracts by using atomic combinations of method annotations that can refer to other methods:

- Given a set of method names n, ‘`@Enable(`n`)` m’ asserts that invoking method m makes calling methods in n valid in a continuation.
- Dually, ‘`@Disable(`n`)` m’ asserts that a call to m disables calls to all methods in n in the continuation.

These annotations specify so-called *‘may call’* properties; our formalism also supports *‘must call’* properties, in which a method *demands* other methods to be called in a continuation. Now, the complete setter/getter contract from Figure 1.3 can be specified with *only four annotations*, namely:

$$\begin{gathered}\texttt{@Enable}(g_1)\ \texttt{@Disable}(s_1)\ s_1\\ \texttt{@Enable}(g_2)\ \texttt{@Disable}(s_2)\ s_2\\ \texttt{@Enable}(g_3)\ \texttt{@Disable}(s_3)\ s_3\\ \texttt{@Enable}(g_4)\ \texttt{@Disable}(s_4)\ s_4\end{gathered}$$

We show that our annotation language induces a strict sub-class of DFA that we call *Bit-vector Finite Automata* (BFA, in what follows) for reasons that will be explained later. We characterize a *context-independency* property that precisely captures restrictions that BFAs impose over standard DFAs. We show that this property is satisfied by all BFAs but not by all DFAs.

Scalable compositional analysis Now, we describe our approach for accomplishing an efficient compositional algorithm (feature (ii)). As mentioned in Section 1.3.1, the performance of a compositional DFA-based analysis depends on the number of states.

In DFA-based analyses, the analysis domain is given by $\mathcal{P}(Q)$, where Q is the set of states. In the intraprocedural analysis, at each method call, the transfer function would need to transition each state in the abstract state according to a given DFA. That is, the transfer function is the DFA's transition function lifted to a subset of states (with signature $\mathcal{P}(Q) \mapsto \mathcal{P}(Q)$). Clearly, the intraprocedural analysis depends linearly in the number of DFA states.

As already mentioned, the compositional interprocedural analysis is affected by the number of states. Each procedure has to be analyzed taking *each* state as an entry state: thus, effectively, we would need to run the intraprocedural analysis $|Q|$ times. Now, as a procedure body can contain branches, the analysis can result in *a set of states* for a given input state: the procedure summary is a mapping from a state into a set of states. For a procedure call, the transfer function would need to apply this mapping, thus taking $|Q|^2$ in the worst-case. Overall, the compositional analysis takes $|Q|^3$ operations in the worst-case per a procedure call.

Now, our approach is to abstract DFA states and transitions such that it allows efficient transfer and join functions. The ability for abstracting is conditioned by a loss on expressivity or analysis precision. As alluded, our abstraction is enabled by expressivity loss (i.e., a context-independency property). Informally, this property says that an effect of a transition on subsequent transition does not depend on previous transitions (a *context*). Relying on this property, we represent a set of states as a bit-vector that, roughly speaking, records the validity of transitions for all states in this set. Moreover, we represent transitions as a bit-vector that records the effects of a transition on the validity of subsequent transitions. Taking BFAs as the basis for our analysis, an abstract domain is a set of bit-vectors; also, both transfer and join functions are bit-vector operations. The resulting intraprocedural analysis thus requires a *constant* number of operations per method invocation.

More importantly, the compositional analysis also has a constant number of operations per method invocation. In fact, the bit-vector abstraction allows a uniform treatment of intraprocedural analysis and procedure summary computation. That is, our compositional analysis is insensitive to the number of states, which is in sharp contrast with DFA-based analyses.

We prove the correctness of our bit-vector abstraction and associated compositional analysis algorithm with respect to DFA-based analysis. That is, we show that for any contract expressible in *both* BFA- and DFA-based analysers, the BFA-based analysis reports an error *if and only if* DFA-based analysis reports an error (i.e., soundness and completeness).

We have implemented our technique in INFER [1], an industrial-strength static analyzer. We provide extensive evaluations for our compositional analysis algorithm that demonstrate considerable gains in performance and usability.

Efficient contract subtyping Finally, checking contract subtyping with BFAs boils down to usual set inclusion. As BFA annotations represent local method dependencies, the subsumption can be checked on *per-method* basis. Intuitively, for each method, we check whether subclass annotations are at least as permissive as corresponding annotations of its superclass, which can be accomplished by a constant number of set inclusions.

1.4.2 Session Types

Question (Q2) is a foundational question with practical ramifications. From a theoretical perspective, an investigation into a minimal formulation of session types represents a study of their *absolute expressiveness*, i.e., whether session types can be explained in terms of themselves. From a practical perspective this question seems relevant as well: identifying the "core" of session types could simplify or even enable their integration in languages whose type systems do not support advanced constructs in session types, in particular sequencing.

To address (Q2) we shall use process languages equipped with *minimal* session types (MSTs, in the following). MSTs are session types without sequencing. This way, in session types such as '$!\langle U\rangle;S$' and '$?(U);S$', we stipulate that S can only correspond to `end`, the type of the terminated protocol. Thus, MSTs eschew sequencing by definition, and determine a strict subset of standard session types.

While adopting minimality in this way may appear as a far too drastic restriction, it turns out that it is not: we shall show that for every process typable under standard (non minimal) session types, there is a semantically equivalent process typable under MSTs. We refer to this as the *minimality result.* The minimality result is significant because it justifies session types in terms of themselves, and shows that sequentiality in types is useful but not indispensable.

We establish our minimality result by devising *encodings* (language translations) between typed processes. That is, we show how to translate a session-typed process into a process typable with MSTs. The main idea behind our encodings is to codify sequencing information give at the type level on leveraging sequentiality at the process level. To argue for the robustness of our approach and results, we establish the minimality result for two different process calculi with session types, namely *higher-order* and *first-order* π-calculi. While the first-order π-calculus is the standard formulation in which channel names can be exchanged through communications, higher-order process calculi are expressive to specify concurrency scenarios in which processes (protocol participants) can exchange abstractions (i.e., values that contain processes) along communication channels. As such, higher-order concurrency can express *code mobility* while retaining a precise and pleasant link with functional calculi.

Now, we provide a brief overview of the mathematical approach to studying MSTs. We base our investigation on the *higher-order session-typed calculus* ($\mathsf{HO}\pi$) [39]. In particular, our developments concern two sub-calculi of $\mathsf{HO}\pi$, namely, HO and π-calculus. HO is a compact blend of λ- and π-calculi in which only abstractions can be exchanged. Although HO does not natively support name passing nor recursion (two features present in π), it can encode both precisely. The two sub-calculi are also interesting because there are mutual encodings between them [39], which will prove to be useful for our study of MSTs.

We first investigate MSTs (and a minimality result) for the HO calculus. Our development draws inspiration from Parrow's work [50] on the minimal fragment of untyped π-calculus that is able to capture its full expressive power. In other words, Parrow established a strict subset of π processes such that any π-process can be translated into a *weakly* equivalent member of this subset. In particular, this fragment restricts the number of nested sequential prefixes of processes to at most three (thus, called *trios* processes). We cast Parrow's results in the realm of typed processes: we simultaneously *decompose* session types and processes. That is, we first show how to decompose standard session types into MSTs; then, we provide the encoding of HO processes typed under ordinary session types into HO processes typed with MSTs. Thus, we refer to this encoding as the *decomposition.* Our main results show the correctness of the decomposition. Namely, we show that a decomposed process satisfies two important properties: it is well-typed using MSTs (*static correctness*); second, it is behaviorally equivalent to the original process (*dynamic correctness*).

Leveraging our results for MSTs in HO, we then define MSTs for the first-order π-calculus. This line of developments comes in two parts.

- In the first part, we utilize existing results: we compose our HO decomposition with the mutual encodings between HO and π given in [39, 41]. As the correctness of the composed encoding follows immediately from that of its constituent functions, this approach is an elegant way to establish the minimality result for π. However, the composed encoding is not entirely satisfactory. A side effect of composing these existing encodings is that the resulting decomposition is inefficient, as it includes redundant synchronizations. These shortcomings are particularly noticeable in the treatment of recursive processes.

- Thus, in the second part, we develop an *optimized* variant of the decomposition in which we remove redundant synchronizations and target recursive processes and variables directly, exploiting the fact that π supports recursion natively. Similarly to the treatment of HO decomposition, we provide full proofs of static and dynamic correctness for the optimized encoding.

To sum up, we prove that standard session types are representable as MSTs by exploiting sequencing in processes. Because we consider both first- and higher-order concurrency, we show that our minimal formulation of session types is *robust*: it stands on its own irrespectively of the kind of exchanged objects (names or processes).

1.5 Summary of Contributions

Having elaborated on our approach, we now summarize the original contributions of our work concerning research questions (Q1) and (Q2).

1.5.1 Bit-vector Typestate Analysis

The contributions addressing (Q1) are the following:

- A specification language for typestates based on *lightweight annotations.* Whereas DFA-based specifications take a *global* standpoint, our specification language rests upon the central insight that many contracts can be decomposed into method dependencies that are *local.* As such, it enables succinct specifications, and induces considerable scalability improvements. Our language induces a new subclass of DFA based on bit-vectors, dubbed BFAs, and can express both ‘may call’ and ‘must call’ properties.
- An efficient contract subtyping algorithm enabled by BFA annotations.
- A compositional analysis technique based on BFAs that takes only a constant number of bit-vector operations per method invocation.
- An implementation of our technique in Infer (available at [8]).[2]
- Extensive evaluations for our analysis technique, which demonstrate considerable gains in performance and usability.

1.5.2 Minimal Session Types

The contributions concerning (Q2) and MSTs can be divided in two parts:

- MSTs for the HO calculus:
    - *Minimal session types* (MSTs): a subclass of standard session types for HO without sequencing that retains its absolute expressiveness.
    - Procedures to decompose (i) standard session types into minimal session types and (ii) HO processes typable with standard session types into HO processes typable with minimal session types. These procedures define a minimality result, and are backed up by a result of *static correctness.*
    - A corresponding result of *dynamic correctness* that attests that a process and its decomposition are *behaviorally equivalent.* This is formalized in terms of *MST bisimulations*, a new typed behavioral equivalence introduced here.
    - Optimizations of our decompositions that bear witness to their robustness.

[2]Our code is available at `https://github.com/aalen9/lfa.git`

- MSTs for the π-calculus:
  - The minimality result but now for the π-calculus. This result is obtained by composing our decomposition for HO (see above) and bidirectional encodings between HO into π (developed in prior work).
  - An optimized variant of the decomposition, which removes redundant communications and natively supports recursion.
  - Full proofs of static and dynamic correctness for the optimized decomposition. For the later, we introduce *characteristic MST-bisimilarity*, a variant of characteristic bisimilarity given in [40].
  - Examples for both composed and optimized decompositions.

1.6 Structure of the Dissertation

The rest of this document is structured as follows. In Chapter 2 we lay down the technical foundations of session types, including an overview of prior work on the HO calculus and the π-calculus, with their respective session type systems. The three subsequent chapters present our original contributions; each of them contains its own revision of related work and concluding remarks.

- In Chapter 3 we address (Q1) by presenting our bit-vector typestate analysis. First, we introduce our BFA specification language. Next, we formulate a strict sub-class of DFAs, called BFAs, that is induced by this typestate language. Then, we present our compositional analysis algorithm based on BFAs; we provide proofs of its correctness. We also present a more general BFA formalisms which includes 'must call' logic. Finally, we provide extensive evaluations of our technique implemented in Infer.

- The next two chapters address (Q2). In Chapter 4 we present *minimal session types*, and the decomposition of well-typed HO processes into minimal session types processes, accompanied by explanations and examples. Next, we show the correctness of the decomposition, by establishing an *MST bisimulation* between an HO process and its decomposition. Then, we examine two optimizations of the decomposition that are enabled by the higher-order nature of our setting. Finally, we discuss extensions of our approach to consider constructs for branching and selection.

- In Chapter 5 we investigate minimal session types for first-order π-calculus. First, we construct the decomposition for π by composing our HO decomposition (from Chapter 4) and bidirectional encodings between HO and π. Then, we develop the optimizations of the decomposition removing redundant synchronizations and leveraging the native support for recursion. Finally, we provide complete proofs of the static and dynamic correctness of the optimized decomposition.

Additional material and proofs omitted in the main text are contained in Chapter 6.

1.7 Origin of the Results

This dissertation is derived from the following peer-reviewed publications:

- Alen Arslanagić, Pavle Subotić, and Jorge A. Pérez. *Scalable Typestate Analysis for Low-latency Environments.* In *Integrated Formal Methods - 17th International Conference, (IFM 2022)*, volume 13274 of Lecture Notes in Computer Science, pages 322–340.

Springer, 2022.

Note: This paper received the *Best Artifact Award* at the conference.
An extended and revised version of this paper is currently under submission to *Formal Aspects of Computing* (special issue of iFM 2022).

- Alen Arslanagić, Jorge A. Pérez, and Erik Voogd. *Minimal Session Types (Pearl).* In *European Conference on Object-Oriented Programming, (ECOOP 2019)*, volume 134 of LIPIcs, pages 23:1–23:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

  Note: An extended and revised version of this paper is currently under submission to *Logical Methods in Computer Science.*

- Alen Arslanagić, Anda-Amelia Palamariuc, and Jorge A. Pérez. *Minimal Session Types for the π-calculus.* In *International Symposium on Principles and Practice of Declarative Programming (PPDP 2021)*, pages 12:1–12:15. ACM, 2021.

  Note: An extended and revised version of this paper will be submitted to a special issue of *Information and Computation* on March 31st, 2023.

2 Preliminaries on Session Types

In this chapter we recall technical foundations needed for our developments on minimal session types. The material in this chapter mostly originates from previous works by Kouzapas et al. [40, 41]. We present HOπ and its two sub-languages HO and π; for which we recall their syntax, semantics, and session type system. We also recall the mutual encodings between HO and π. Finally, we overview two relevant behavioral equivalences from [41]: *characteristic* and *higher-order* bisimilarity. This is relevant for our work as we shall develop variants of these bisimilarities for proving dynamic correctness of our decompositions.

2.1 HOπ (and its Sub-languages HO and π)

$$
\begin{aligned}
n &::= a, b \mid s, \overline{s} \\
u, w &::= n \mid x, y, z \\
V, W &::= \colorbox{lightgray}{u} \mid \boxed{\lambda x.\, P} \mid \boxed{x, y, z} \\
P, Q &::= u!\langle V \rangle.P \mid u?(x).P \\
&\quad \mid \boxed{V\,u} \mid P \mid Q \mid (\nu\, n)\, P \mid \mathbf{0} \mid \colorbox{lightgray}{$X \mid \mu X.P$}
\end{aligned}
$$

Figure 2.1: Syntax of HOπ. The sub-language HO lacks $\colorbox{lightgray}{shaded}$ constructs, while π lacks $\boxed{\text{boxed}}$ constructs.

Figure 2.1 gives the **syntax** of processes $P, Q, \dots$, values $V, W, \dots$, and conventions for names. Identifiers $a, b, c, \dots$ denote *shared names*, while $s, \overline{s}, \dots$ are used for *session names*. Duality is defined only on session names, thus $\overline{\overline{s}} = s$, but $\overline{a} = a$. *Names* (shared or sessions) are denoted by $m, n \dots$, and $x, y, z, \dots$ range over variables. We write $\widetilde{x}$ to denote a tuple $(x_1, \dots, x_n)$, and use ϵ to denote the empty tuple.

An abstraction $\lambda x.\, P$ binds x to its body P. In processes, sequencing is specified via prefixes. The *output* prefix, $u!\langle V \rangle.P$, sends value V on name u, then continues as P. Its dual is the *input* prefix, $u?(x).P$, in which variable x is bound. Parallel composition, $P \mid Q$, reflects the combined behaviour of two processes running simultaneously. Restriction $(\nu\, n)\, P$ binds the endpoints $n, \overline{n}$ in process P. Process $\mathbf{0}$ denotes inaction. Recursive variables and recursive processes are denoted X and $\mu X.P$, respectively. Replication is denoted by the shorthand notation $*\, P$, which stands for $\mu X.(P \mid X)$.

The sets of free variables, sessions, and names of a process are denoted $\mathtt{fv}(P)$, $\mathtt{fs}(P)$, and $\mathtt{fn}(P)$. A process P is *closed* if $\mathtt{fv}(P) = \emptyset$. We write $u!\langle\rangle.P$ and $u?().P$ when the communication objects are not relevant. Also, we omit trailing occurrences of $\mathbf{0}$.

As Figure 2.1 details, the **sub-languages** π and HO of HOπ differ as follows: application $V\,u$ is only present in HO; constructs for recursion $\mu X.P$ are present in π but not in HO.

The **operational semantics** of HOπ, enclosed in Figure 2.2, is expressed through a *reduction relation*, denoted $\longrightarrow$. Reduction is closed under *structural congruence*, $\equiv$, which

$$
\begin{array}{rr}
(\lambda x.\, P)\, u \longrightarrow P\{u/x\} & \text{[App]} \\
n!\langle V\rangle.P \mid \overline{n}?(x).Q \longrightarrow P \mid Q\{V/x\} & \text{[Pass]} \\
P \longrightarrow P' \Rightarrow (\nu\, n)\, P \longrightarrow (\nu\, n)\, P' & \text{[Res]} \\
P \longrightarrow P' \Rightarrow P \mid Q \longrightarrow P' \mid Q & \text{[Par]} \\
P \equiv Q \longrightarrow Q' \equiv P' \Rightarrow P \longrightarrow P' & \text{[Cong]}
\end{array}
$$

$$
\begin{array}{l}
P_1 \mid P_2 \equiv P_2 \mid P_1 \quad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3 \\
P \mid \mathbf{0} \equiv P \quad P \mid (\nu\, n)\, Q \equiv (\nu\, n)\, (P \mid Q) \;\; (n \notin \mathtt{fn}(P)) \\
(\nu\, n)\, \mathbf{0} \equiv \mathbf{0} \qquad \mu X.P \equiv P\{\mu X.P/X\} \qquad P \equiv Q \text{ if } P \equiv_\alpha Q
\end{array}
$$

Figure 2.2: Operational Semantics of $\mathsf{HO}\pi$.

identifies equivalent processes from a structural perspective. We write $P\{V/x\}$ to denote the capture-avoiding substitution of variable x with value V in P. We write '$\{\}$' to denote the empty substitution. In Figure 2.2, Rule [App] denotes application, which only concerns names. Rule [Pass] defines a shared or session interaction on channel n's endpoints. The remaining rules are standard. We write $\longrightarrow^k$ for a k-step reduction, and $\longrightarrow^*$ for the reflexive, transitive closure of $\longrightarrow$.

Example 2.1.1 (Encoding Name-Passing)**.** The HO calculus lacks the name-passing primitives of $\mathsf{HO}\pi$. Hence, it cannot express reductions of the form

$$n!\langle m\rangle.P \mid \overline{n}?(x).Q \longrightarrow P \mid Q\{m/x\} \tag{2.1}$$

Fortunately, name-passing can be encoded in HO in a fully-abstract way: as shown in [39], one can use abstraction passing to "pack" a name. To this end, Figure 2.3 defines the required syntactic sugar, at the level of processes and types. Then, the reduction (2.1) can be mimicked as

$$
\begin{array}{rcl}
n!\langle \ulcorner m \urcorner\rangle.P \mid \overline{n}?(\ulcorner x \urcorner).Q = & & n!\langle \lambda z.\; z?(x).(x\, m)\rangle.P \mid \overline{n}?(y).(\nu\, s)(y\, s \mid \overline{s}!\langle \lambda x.\, Q\rangle) \\
& \longrightarrow & P \mid (\nu\, s)(\lambda z.\; z?(x).(x\, m)\, s \mid \overline{s}!\langle \lambda x.\, Q\rangle) \\
& \longrightarrow & P \mid (\nu\, s)(s?(x).(x\, m) \mid \overline{s}!\langle \lambda x.\, Q\rangle) \\
& \longrightarrow & P \mid (\lambda x.\, Q)\, m \\
& \longrightarrow & P \mid Q\{m/x\}
\end{array}
$$

$\lhd$

Remark 1 (Polyadic Communication)**.** $\mathsf{HO}\pi$ as presented above allows only for *monadic communication*, i.e., the exchange of tuples of values with length 1. We will find it convenient to use $\mathsf{HO}\pi$ with *polyadic communication*, i.e., the exchange of tuples of values $\widetilde{V} = (V_1, \ldots, V_k)$, with length $|\widetilde{V}| = k$. We will use ϵ to denote the empty tuple.

In HO, polyadicity appears in session synchronizations and applications, but not in synchronizations on shared names. This entails having the following reduction rules:

$$
\begin{array}{c}
(\lambda \widetilde{x}.\, P)\, \widetilde{u} \longrightarrow P\{\widetilde{u}/\widetilde{x}\} \\
s!\langle \widetilde{V}\rangle.P \mid \overline{s}?(\widetilde{x}).Q \longrightarrow P \mid Q\{\widetilde{V}/\widetilde{x}\}
\end{array}
$$

$$\begin{aligned}
t!\langle \ulcorner \tilde{u} \urcorner \rangle.P &\triangleq t!\langle \lambda z.z?(x).(x\ \tilde{u})\rangle.P \\
t?(\ulcorner \tilde{x} \urcorner).Q &\triangleq t?(y).((\nu\, z)\,(y\ z \mid z!\langle \lambda \tilde{x}.Q\rangle)) \\
\ulcorner \widetilde{S} \urcorner &\triangleq (?(\llbracket \widetilde{S} \rrbracket {\multimap} \diamond);\mathtt{end}) {\multimap} \diamond \\
\ulcorner \langle \widetilde{S} \rangle \urcorner &\triangleq (?(\langle \llbracket \widetilde{S} \rrbracket \rangle {\multimap} \diamond);\mathtt{end}) {\multimap} \diamond \\
\ulcorner \widetilde{C} {\multimap} \diamond \urcorner &\triangleq \llbracket \widetilde{C} \rrbracket {\multimap} \diamond \\
\ulcorner \widetilde{C} {\rightarrow} \diamond \urcorner &\triangleq \llbracket \widetilde{C} \rrbracket {\rightarrow} \diamond \\
\llbracket !\langle U \rangle;S \rrbracket &\triangleq !\langle \ulcorner U \urcorner \rangle;\llbracket S \rrbracket \\
\llbracket ?(U);S \rrbracket &\triangleq ?(\ulcorner U \urcorner);\llbracket S \rrbracket \\
\llbracket C_1, \ldots, C_n \rrbracket &\triangleq \llbracket C_1 \rrbracket, \ldots, \llbracket C_n \rrbracket
\end{aligned}$$

Figure 2.3: Encoding name passing in HO

$$\begin{array}{lll}
U ::= C \mid L & \qquad & C ::= S \mid \langle S \rangle \mid \langle L \rangle \\
L ::= U {\rightarrow} \diamond \mid U {\multimap} \diamond & & S ::= !\langle U \rangle;S \mid ?(U);S \\
 & & \quad \mid \mu \mathtt{t}.S \mid \mathtt{t} \mid \mathtt{end}
\end{array}$$

Figure 2.4: STs for HOπ

where the simultaneous substitutions $P\{\tilde{u}/\tilde{x}\}$ and $P\{\widetilde{V}/\tilde{x}\}$ are as expected. This polyadic HO can be readily encoded into (monadic) HO [40]; for this reason, by a slight abuse of notation we will often write HO when we actually mean “polyadic HO”.

2.2 Session Types for HOπ

Figure 2.4 gives the syntax of types. Value types U include the first-order types C and the higher-order types L. Session types are denoted with S and shared types with $\langle S \rangle$ and $\langle L \rangle$. We write $\diamond$ to denote the *process type*. The functional types $U {\rightarrow} \diamond$ and $U {\multimap} \diamond$ denote *shared* and *linear* higher-order types, respectively. The *output type* $!\langle U \rangle;S$ is assigned to a name that first sends a value of type U and then follows the type described by S. Dually, $?(U);S$ denotes an *input type*. Type end is the termination type. We assume the *recursive type* $\mu\mathtt{t}.S$ is guarded, i.e., the type variable t only appears under prefixes. This way, e.g., the type $\mu\mathtt{t}.\mathtt{t}$ is not allowed. The sets of free/bound variables of a type S are defined as usual; the sole binder is $\mu\mathtt{t}.S$. Closed session types do not have free type variables.

Session types for HO exclude C from value types U; session types for π exclude L from value types U and $\langle L \rangle$ from C.

Notation 2.2.1. *As mentioned in the introduction, we shall study session types in which the continuation S in $!\langle U \rangle;S$ and $?(U);S$ is always* end*. Given this, we may sometimes omit trailing* end*'s and write $!\langle U \rangle$ and $?(U)$ rather than $!\langle U \rangle;\mathtt{end}$ and $?(U);\mathtt{end}$, respectively.*

In theories of session types *duality* is a key notion: implementations derived from dual session types will respect their protocols at run-time, avoiding communication errors. Intuitively, duality is obtained by exchanging ! by ? (and vice versa), including the fixed point construction.

We write S dual T if session types S and T are dual according to this intuition; the formal definition is coinductive, and given in [40] (see also [28]).

We consider shared, linear, and session *environments*, denoted Γ, Λ, and Δ, resp.:

$$\begin{array}{l}\Gamma \ ::= \ \emptyset \ \mid \ \Gamma, x : U \!\rightarrow\! \diamond \ \mid \ \Gamma, u : \langle S \rangle \ \mid \ \Gamma, u : \langle L \rangle \ \mid \ \Gamma, X : \Delta \\ \Lambda \ ::= \ \emptyset \ \mid \ \Lambda, x{:}U \!\multimap\! \diamond \qquad\qquad \Delta \ ::= \ \emptyset \ \mid \ \Delta, u{:}S\end{array}$$

Γ, Λ, and Δ satisfy different structural principles. Γ maps variables and shared names to value types, and recursive variables to session environments; it admits weakening, contraction, and exchange principles. While Λ maps variables to linear higher-order types, Δ maps session names to session types. Both Λ and Δ are only subject to exchange. The domains of Γ, Λ and Δ (denoted $\mathtt{dom}(\Gamma)$, $\mathtt{dom}(\Lambda)$, and $\mathtt{dom}(\Delta)$) are assumed pairwise distinct.

Given Γ, we write $\Gamma\backslash x$ to denote the environment obtained from Γ by removing the assignment $x : U \rightarrow \diamond$, for some U. Given tuple of variables $\widetilde{x}$, notation $\Gamma\backslash\widetilde{x}$ is defined similarly and has the expected readings. These notations apply similarly to Δ and Λ; we write $\Delta\backslash\Delta'$ (and $\Lambda\backslash\Lambda'$) with the expected meaning. With a slight abuse of notation, we sometimes write $(\Gamma, \Delta)(\widetilde{x})$ to denote the tuple of types assigned to the variables in $\widetilde{x}$ by the environments Γ and Δ. Notation $\Delta_1 \cdot \Delta_2$ means the disjoint union of Δ_1 and Δ_2. We define *typing judgements* for values V and processes P:

$$\Gamma; \Lambda; \Delta \vdash V \triangleright U \qquad\qquad\qquad \Gamma; \Lambda; \Delta \vdash P \triangleright \diamond$$

The judgement on the left says that under environments Γ, Λ, and Δ value V has type U; the judgement on the right says that under environments Γ, Λ, and Δ process P has the process type $\diamond$.

Figure 2.5 shows the typing rules; we briefly describe them and refer the reader to [40] for a full account. The shared type $C \rightarrow \diamond$ is derived using Rule (Prom) only if the value has a linear type with an empty linear environment. Rule (EProm) allows us to freely use a shared type variable as linear. Abstraction values are typed with Rule (Abs). Application typing is governed by Rule (App): the type C of an application name u must match the type of the application variable x ($C \multimap \diamond$ or $C \rightarrow \diamond$). Rules (Req) and (Acc) type interaction along shared names; the type of the sent/received object V (i.e., U) should match the type of the subject s ($\langle U \rangle$). In Rule (Send), the type U of the value V should appear as a prefix in the session type $!\langle U \rangle;S$ of u. Rule (Rcv) is its dual.

Type soundness for $\mathsf{HO}\pi$ relies on two auxiliary notions:

Definition 2.2.1 (Session Environments: Balanced/Reduction)**.** Let Δ be a session environment.

- Δ is *balanced* if whenever $s : S_1, \overline{s} : S_2 \in \Delta$ then S_1 dual S_2.
- We define reduction $\longrightarrow$ on session environments as:

$$\begin{array}{c}\Delta, s :!\langle U \rangle;S_1, \overline{s} :?(U);S_2 \longrightarrow \Delta, s : S_1, \overline{s} : S_2 \\ \Delta, s : \oplus\{l_i : S_i\}_{i\in I}, \overline{s} : \&\{l_i : S_i'\}_{i\in I} \longrightarrow \Delta, s : S_k, \overline{s} : S_k' \ (k \in I)\end{array}$$

Theorem 2.2.1 (Type Soundness [40])**.** Suppose $\Gamma; \emptyset; \Delta \vdash P \triangleright \diamond$ with $\mathsf{balanced}(\Delta)$. Then $P \longrightarrow P'$ implies $\Gamma; \emptyset; \Delta' \vdash P' \triangleright \diamond$ and $\Delta = \Delta'$ or $\Delta \longrightarrow \Delta'$ with $\mathsf{balanced}(\Delta')$.

Lemma 2.2.1 (Substitution Lemma [39])**.** $\Gamma; \Lambda; \Delta, x : S \vdash P \triangleright \diamond$ *and* $u \notin$ *dom*$(\Gamma, \Lambda, \Delta)$ *implies* $\Gamma; \Lambda; \Delta, u : S \vdash P\{u/x\} \triangleright \diamond$.

$$\textsc{(Sess)}\quad \Gamma; \emptyset; \{u : S\} \vdash u \triangleright S \qquad \textsc{(Sh)}\quad \Gamma, u : U; \emptyset; \emptyset \vdash u \triangleright U$$

$$\textsc{(LVar)}\quad \Gamma; \{x : C \multimap \diamond\}; \emptyset \vdash x \triangleright C \multimap \diamond \qquad \textsc{(RVar)}\quad \Gamma, X : \Delta; \emptyset; \Delta \vdash X \triangleright \diamond$$

$$\textsc{(Abs)}\quad \frac{\Gamma; \Lambda; \Delta_1 \vdash P \triangleright \diamond \quad \Gamma; \emptyset; \Delta_2 \vdash x \triangleright C}{\Gamma \backslash x; \Lambda; \Delta_1 \backslash \Delta_2 \vdash \lambda x.\, P \triangleright C \multimap \diamond}$$

$$\textsc{(App)}\quad \frac{\Gamma; \Lambda; \Delta_1 \vdash V \triangleright C \leadsto \diamond \quad \leadsto \in \{\multimap, \to\} \quad \Gamma; \emptyset; \Delta_2 \vdash u \triangleright C}{\Gamma; \Lambda; \Delta_1, \Delta_2 \vdash V\, u \triangleright \diamond}$$

$$\textsc{(Prom)}\quad \frac{\Gamma; \emptyset; \emptyset \vdash V \triangleright C \multimap \diamond}{\Gamma; \emptyset; \emptyset \vdash V \triangleright C \to \diamond} \qquad \textsc{(EProm)}\quad \frac{\Gamma; \Lambda, x : C \multimap \diamond; \Delta \vdash P \triangleright \diamond}{\Gamma, x : C \to \diamond; \Lambda; \Delta \vdash P \triangleright \diamond} \qquad \textsc{(End)}\quad \frac{\Gamma; \Lambda; \Delta \vdash P \triangleright T \quad u \notin \mathtt{dom}(\Gamma, \Lambda, \Delta)}{\Gamma; \Lambda; \Delta, u : \mathsf{end} \vdash P \triangleright \diamond}$$

$$\textsc{(Rec)}\quad \frac{\Gamma, X : \Delta; \emptyset; \Delta \vdash P \triangleright \diamond}{\Gamma; \emptyset; \Delta \vdash \mu X.P \triangleright \diamond} \qquad \textsc{(Par)}\quad \frac{\Gamma; \Lambda_i; \Delta_i \vdash P_i \triangleright \diamond \quad i = 1, 2}{\Gamma; \Lambda_1, \Lambda_2; \Delta_1, \Delta_2 \vdash P_1 \mid P_2 \triangleright \diamond} \qquad \textsc{(Nil)}\quad \frac{}{\Gamma; \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond}$$

$$\textsc{(Send)}\quad \frac{u : S \in \Delta_1, \Delta_2 \quad \Gamma; \Lambda_1; \Delta_1 \vdash P \triangleright \diamond \quad \Gamma; \Lambda_2; \Delta_2 \vdash V \triangleright U}{\Gamma; \Lambda_1, \Lambda_2; ((\Delta_1, \Delta_2) \setminus u : S), u : !\langle U \rangle; S \vdash u!\langle V \rangle.P \triangleright \diamond}$$

$$\textsc{(Req)}\quad \frac{\Gamma; \Lambda; \Delta_1 \vdash P \triangleright \diamond \quad \Gamma; \emptyset; \emptyset \vdash u \triangleright \langle \mathcal{U} \rangle \quad \Gamma; \emptyset; \Delta_2 \vdash V \triangleright \mathcal{U} \quad \mathcal{U} \in \{S, L\}}{\Gamma; \Lambda; \Delta_1, \Delta_2 \vdash u!\langle V \rangle.P \triangleright \diamond}$$

$$\textsc{(Rcv)}\quad \frac{\Gamma; \Lambda_1; \Delta_1, u : S \vdash P \triangleright \diamond \quad \Gamma; \Lambda_2; \Delta_2 \vdash x \triangleright U}{\Gamma \backslash x; \Lambda_1 \backslash \Lambda_2; \Delta_1 \backslash \Delta_2, u : ?(U); S \vdash u?(x).P \triangleright \diamond} \qquad \textsc{(Acc)}\quad \frac{\begin{array}{c}\Gamma; \Lambda_1; \Delta_1 \vdash P \triangleright \diamond \quad \Gamma; \emptyset; \emptyset \vdash u \triangleright \langle \mathcal{U} \rangle \\ \Gamma; \Lambda_2; \Delta_2 \vdash x \triangleright \mathcal{U} \quad \mathcal{U} \in \{S, L\}\end{array}}{\Gamma \backslash x; \Lambda_1 \backslash \Lambda_2; \Delta_1 \backslash \Delta_2 \vdash u?(x).P \triangleright \diamond}$$

$$\textsc{(Bra)}\quad \frac{\forall i \in I \quad \Gamma; \Lambda; \Delta, u : S_i \vdash P_i \triangleright \diamond}{\Gamma; \Lambda; \Delta, u : \&\{l_i : S_i\}_{i \in I} \vdash u \triangleright \{l_i : P_i\}_{i \in I} \triangleright \diamond} \qquad \textsc{(Sel)}\quad \frac{\Gamma; \Lambda; \Delta, u : S_j \vdash P \triangleright \diamond \quad j \in I}{\Gamma; \Lambda; \Delta, u : \oplus\{l_i : S_i\}_{i \in I} \vdash u \triangleleft l_j.P \triangleright \diamond}$$

$$\textsc{(ResS)}\quad \frac{\Gamma; \Lambda; \Delta, s : S_1, \overline{s} : S_2 \vdash P \triangleright \diamond \quad S_1\ \mathsf{dual}\ S_2}{\Gamma; \Lambda; \Delta \vdash (\nu\, s)\, P \triangleright \diamond} \qquad \textsc{(Res)}\quad \frac{\Gamma, a : \langle S \rangle; \Lambda; \Delta \vdash P \triangleright \diamond}{\Gamma; \Lambda; \Delta \vdash (\nu\, a)\, P \triangleright \diamond}$$

Figure 2.5: Typing Rules for $\mathsf{HO}\pi$.

Lemma 2.2.2 (Shared environment weakening)**.** *If* $\Gamma; \Lambda; \Delta \vdash P \triangleright \diamond$ *then* $\Gamma, x : C \to \diamond; \Lambda; \Delta \vdash P \triangleright \diamond$ *and* $\Gamma, u : \langle U \rangle; \Lambda; \Delta \vdash P \triangleright \diamond$*.*

$$\text{(PolyVar)}\quad \Gamma, \widetilde{x} : \widetilde{U}_x ; \widetilde{y} : \widetilde{U}_y ; \emptyset \vdash \widetilde{x}\widetilde{y} : \widetilde{U}_x\widetilde{U}_y \qquad \text{(PolySess)}\quad \Gamma, \widetilde{x} : \widetilde{U}_x ; \widetilde{y} : \widetilde{U}_y ; \emptyset \vdash \widetilde{x}\widetilde{y} : \widetilde{U}_x\widetilde{U}_y$$

$$\text{(PolyRcv)}\ \frac{\Gamma; \Lambda_1; \Delta, u : S \vdash P \triangleright \diamond \quad \Gamma; \Lambda_2; \emptyset \vdash \widetilde{x} \triangleright \widetilde{U}}{\Gamma \setminus \widetilde{x}; \Lambda_1 \setminus \Lambda_2; \Delta, u : ?(\widetilde{U});S \vdash u?(\widetilde{x}).P \triangleright \diamond} \qquad \text{(PolySend)}\ \frac{u : S \in \Delta \quad \Gamma; \Lambda_1; \Delta \vdash P \quad \Gamma; \Lambda_2; \emptyset \vdash \widetilde{x} \triangleright \widetilde{U}}{\Gamma; \Lambda_1, \Lambda_2; (\Delta \setminus u : S), u : !\langle \widetilde{U} \rangle;S \vdash u!\langle \widetilde{x} \rangle.P}$$

$$\text{(PolyApp)}\ \frac{\leadsto \in \{\multimap, \rightarrow\} \quad \Gamma; \Lambda; \Delta_1 \vdash V \triangleright \widetilde{C} \leadsto \diamond \quad \Gamma; \emptyset; \Delta_2 \vdash \widetilde{u} \triangleright \widetilde{C}}{\Gamma; \Lambda; \Delta_1, \Delta_2 \vdash V\,\widetilde{u}}$$

$$\text{(PolyAbs)}\ \frac{\Gamma; \Lambda; \Delta_1 \vdash P \triangleright \diamond \quad \Gamma; \emptyset; \Delta_2 \vdash \widetilde{x} \triangleright \widetilde{C}}{\Gamma \setminus \widetilde{x}; \Lambda; \Delta_1 \setminus \Delta_2 \vdash \lambda \widetilde{x}.\, P \triangleright \widetilde{C} \multimap \diamond} \qquad \text{(PolyApp)}\ \frac{\Gamma, \widetilde{a} : \widetilde{\langle U \rangle}; \Lambda; \Delta \vdash P}{\Gamma; \Lambda; \Delta \vdash (\nu\, \widetilde{a})\, P}$$

$$\text{(PolyApp)}\ \frac{\Gamma; \Lambda; \Delta, \widetilde{s} : \widetilde{S}_1, \overline{\widetilde{s}} : \widetilde{S}_2 \vdash P \quad \widetilde{S}_1\ \mathsf{dual}\ \widetilde{S}_2}{\Gamma; \Lambda; \Delta \vdash (\nu\, \widetilde{s})\, P}$$

Figure 2.6: Polyadic typing rules for $\mathsf{HO}\pi$

Lemma 2.2.3 (Shared environment strengthening).

- *If* $\Gamma; \Lambda; \Delta \vdash P \triangleright \diamond$ *and* $x \notin \mathtt{fv}(P)$ *then* $\Gamma \setminus x; \Lambda; \Delta \vdash P \triangleright \diamond$.
- *If* $\Gamma; \Lambda; \Delta \vdash P \triangleright \diamond$ *and* $u \notin \mathtt{fn}(P)$ *then* $\Gamma \setminus u; \Lambda; \Delta \vdash P \triangleright \diamond$.

Remark 2 (Typed Polyadic Communication). For typing processes with polyadic communication (cf. Remark 1), we derive polyadic rules in Figure 2.6 as an expected extensions of $\mathsf{HO}\pi$ typing rules (given in Figure 2.5).

Example 2.2.1 (Typing name-passing constructs). In Example 2.1.1 we recalled how to encode name-passing constructs in HO; now we show that this translation is typed. Following the name-passing encoding from [39] we define a syntactic sugar for types. The following typing rules for name-passing are derivable:

$$\text{(SendN)}\ \frac{\Gamma; \Lambda_1; \Delta_1 \vdash P \triangleright \diamond \quad \Gamma; \Lambda_2; \Delta_2 \vdash \widetilde{b} \triangleright \widetilde{C}}{\Gamma; \Lambda_1, \Lambda_2; \Delta_1, \Delta_2, t : !\langle \ulcorner \widetilde{C} \urcorner \rangle;\mathtt{end} \vdash t!\langle \ulcorner \widetilde{b} \urcorner \rangle.P \triangleright \diamond}$$

$$\text{(RcvN)}\ \frac{\Gamma; \Lambda_1; \Delta_1 \vdash P \triangleright \diamond \quad \Gamma; \Lambda_2; \Delta_2 \vdash \widetilde{x} \triangleright \widetilde{C}}{\Gamma \backslash x; \Lambda_1 \backslash \Lambda_2; \Delta_1 \backslash \Delta_2, t : ?(\ulcorner \widetilde{C} \urcorner);\mathtt{end} \vdash t?(\ulcorner \widetilde{x} \urcorner).P \triangleright \diamond}$$

Notation 2.2.2 (Type Annotations). *We shall often annotate bound names and variables with their respective type. We will write, e.g.,* $(\nu\, s : S)\, P$ *to denote that the type of* s *in* P *is* S*. Similarly for values: we shall write* $\lambda u : C.\, P$*. Also, letting* $\leadsto \in \{\multimap, \rightarrow\}$*, we may write* $\lambda u : C^{\leadsto}.\, P$ *to denote that the value is linear (if* $\leadsto = \multimap$*) or shared (if* $\leadsto = \rightarrow$*). That is, we write* $\lambda u : C^{\leadsto}.\, P$ *if* $\Gamma; \Lambda; \Delta \vdash \lambda u.\, P \triangleright C \leadsto \diamond$*, for some* Γ*,* Λ*, and* Δ*.*

2.3 Mutual Encodings Between π and HO

The encodings $[\![\cdot]\!]^1_g : \pi \to \mathsf{HO}$ and $[\![\cdot]\!]^2 : \mathsf{HO} \to \pi$ are *typed*: each consists of a translation on processes and a translation on types. This way, $(\!(\cdot)\!)^1$ translates types for first-order processes into types for higher-order processes, while $(\!(\cdot)\!)^2$ operates in the opposite direction—see Figure 2.8 and Figure 2.9, respectively. Remarkably, these translations on processes and types do not alter their sequentiality.

From π to HO To mimic the sending of name w, the encoding $[\![\cdot]\!]^1_g$ (given in Figure 2.8) encloses w within the body of an input-guarded abstraction. The corresponding input process receives this higher-order value, applies it on a restricted session, and sends the encoded continuation through the session's dual.

Several auxiliary notions are used to encode recursive processes, we describe them intuitively (see [41] for full details). The key idea is to encode recursive processes in π using a "duplicator" process in HO, circumventing linearity by replacing free names with variables. The parameter g is a map from process variables to sequences of name variables. To handle linearity, auxiliary mappings are defined: $\|\cdot\|$ maps sequences of session names into sequences of variables, and $\llbracket \cdot \rrbracket_\emptyset$ maps processes with free names to processes without free names (but with free variables instead):

Definition 2.3.1 (Auxiliary Mappings)**.** We define mappings $\| \cdot \|$ and $\llbracket \cdot \rrbracket_\sigma$ as follows:

- $\| \cdot \| : 2^{\mathcal{N}} \longrightarrow \mathcal{V}^\omega$ is a map of sequences of lexicographically ordered names to sequences of variables, defined inductively as:

$$\begin{aligned} \|\epsilon\| &= \epsilon \\ \|n, \tilde{m}\| &= x_n, \|\tilde{m}\| \quad (x \text{ fresh}) \end{aligned}$$

- Given a set of session names and variables σ, the map $\llbracket \cdot \rrbracket_\sigma : \mathsf{HO} \to \mathsf{HO}$ is as in Figure 2.7.

The encoding $(\!(\cdot)\!)^1$ depends on the auxiliary function $\lfloor \cdot \rfloor^1$, defined on value types. Following the encoding on processes, this mapping on values takes a first-order value type and encodes it into a linear higher-order value type, which encloses an input type that expects to receive another higher-order type. Notice how the innermost higher-order value type is either shared or linear, following the nature of the given type.

From HO to π The encoding $[\![\cdot]\!]^2$ (given in Figure 2.9) simulates higher-order communication using first-order constructs, following Sangiorgi [53]. The idea is to use *trigger names*, which point towards copies of input-guarded server processes that should be activated. The encoding of abstraction sending distinguishes two cases: if the abstraction body does not contain any free session names (which are linear), then the server can be replicated. Otherwise, if the value contains session names then its corresponding server name must be used exactly once. The encoding of abstraction receiving proceeds inductively, noticing that the variable is now a placeholder for a first-order name. The encoding of application is also in two cases; both of them depend on the creation of a fresh session, which is used to pass around the applied name.

2.4 Labelled Transition System for Processes

We define the interaction of processes with their environment using action labels ℓ:

$$
\begin{array}{rcl@{\quad}rcl}
\llbracket w!\langle \lambda x.\, Q\rangle.P \rrbracket_\sigma & \stackrel{\mathsf{def}}{=} & u!\langle \lambda x.\, \llbracket Q \rrbracket_{\sigma,x}\rangle.\llbracket P \rrbracket_\sigma & \llbracket w \triangleright \{l_i : P_i\}_{i\in I} \rrbracket_\sigma & \stackrel{\mathsf{def}}{=} & u \triangleright \{l_i : \llbracket P_i \rrbracket_\sigma\}_{i\in I} \\
\llbracket w?(x).P \rrbracket_\sigma & \stackrel{\mathsf{def}}{=} & u?(x).\llbracket P \rrbracket_\sigma & \llbracket w \triangleleft l.P \rrbracket_\sigma & \stackrel{\mathsf{def}}{=} & u \triangleleft l.\llbracket P \rrbracket_\sigma \\
\llbracket (\nu\, n)\, P \rrbracket_\sigma & \stackrel{\mathsf{def}}{=} & (\nu\, n)\, \llbracket P \rrbracket_{\sigma,n} & \llbracket (\lambda x.Q)\, w \rrbracket_\sigma & \stackrel{\mathsf{def}}{=} & (\lambda x.\llbracket Q \rrbracket_{\sigma,x})\, u \\
\llbracket P \mid Q \rrbracket_\sigma & \stackrel{\mathsf{def}}{=} & \llbracket P \rrbracket_\sigma \mid \llbracket Q \rrbracket_\sigma & \llbracket x\, w \rrbracket_\sigma & \stackrel{\mathsf{def}}{=} & x\, u \\
\llbracket \mathbf{0} \rrbracket_\sigma & \stackrel{\mathsf{def}}{=} & \mathbf{0} & & &
\end{array}
$$

$$
\text{In all cases: } u = \begin{cases} x_n & \text{if } w \text{ is a name } n \text{ and } n \notin \sigma \ (x \text{ fresh}) \\ w & \text{otherwise: } w \text{ is a variable or a name } n \text{ and } n \in \sigma \end{cases}
$$

Figure 2.7: Auxiliary mapping used to encode $\mathsf{HO}\pi$ into HO (Definition 2.3.1).

Terms:

$$
\begin{array}{rcl}
\llbracket u!\langle w\rangle.P \rrbracket^1_g & \stackrel{\mathsf{def}}{=} & u!\langle \lambda z.\ z?(x).(x\, w)\rangle.\llbracket P \rrbracket^1_g \\
\llbracket u?(x{:}C).Q \rrbracket^1_g & \stackrel{\mathsf{def}}{=} & u?(y).(\nu\, s)(y\, s \,|\, \overline{s}!\langle \lambda x.\ \llbracket Q \rrbracket^1_g\rangle.\mathbf{0}) \\
\llbracket P \mid Q \rrbracket^1_g & \stackrel{\mathsf{def}}{=} & \llbracket P \rrbracket^1_g \mid \llbracket Q \rrbracket^1_g \\
\llbracket (\nu\, n)\, P \rrbracket^1_g & \stackrel{\mathsf{def}}{=} & (\nu\, n)\, \llbracket P \rrbracket^1_g \\
\llbracket \mathbf{0} \rrbracket^1_g & \stackrel{\mathsf{def}}{=} & \mathbf{0} \\
\llbracket \mu X.P \rrbracket^1_g & \stackrel{\mathsf{def}}{=} & (\nu\, s)(\overline{s}!\langle V\rangle.\mathbf{0} \mid s?(z_X).\llbracket P \rrbracket^1_{g,\{X\to\tilde{n}\}}) \\
 & & \text{where } (\tilde{n} = \mathtt{fn}(P)) \\
 & & V = \lambda(\|\tilde{n}\|, y).\ y?(z_X).\llbracket \llbracket P \rrbracket^1_{g,\{X\to\tilde{n}\}} \rrbracket_\emptyset \\
\llbracket X \rrbracket^1_g & \stackrel{\mathsf{def}}{=} & (\nu\, s)(z_X\, (\tilde{n}, s) \mid \overline{s}!\langle z_X\rangle.\mathbf{0}) \quad (\tilde{n} = g(X))
\end{array}
$$

Types:

$$
\begin{array}{rcl@{\qquad}rcl}
\lfloor S \rfloor^1 & \stackrel{\mathsf{def}}{=} & (?(\langle\!\langle S \rangle\!\rangle^1 \multimap \diamond);\mathsf{end}) \multimap \diamond & & & \\
\lfloor \langle S \rangle \rfloor^1 & \stackrel{\mathsf{def}}{=} & (?(\langle \langle\!\langle S \rangle\!\rangle^1 \rangle \to \diamond);\mathsf{end}) \multimap \diamond & & & \\
\langle\!\langle !\langle U\rangle;S \rangle\!\rangle^1 & \stackrel{\mathsf{def}}{=} & !\langle \lfloor U \rfloor^1 \rangle;\langle\!\langle S \rangle\!\rangle^1 & & & \\
\langle\!\langle ?(U);S \rangle\!\rangle^1 & \stackrel{\mathsf{def}}{=} & ?(\lfloor U \rfloor^1);\langle\!\langle S \rangle\!\rangle^1 & & & \\
\langle\!\langle \langle S \rangle \rangle\!\rangle^1 & \stackrel{\mathsf{def}}{=} & \langle \langle\!\langle S \rangle\!\rangle^1 \rangle & \langle\!\langle \mu \mathsf{t}.S \rangle\!\rangle^1 & \stackrel{\mathsf{def}}{=} & \mu \mathsf{t}.\langle\!\langle S \rangle\!\rangle^1 \\
\langle\!\langle \mathsf{end} \rangle\!\rangle^1 & \stackrel{\mathsf{def}}{=} & \mathsf{end} & \langle\!\langle \mathtt{t} \rangle\!\rangle^1 & \stackrel{\mathsf{def}}{=} & \mathtt{t}
\end{array}
$$

Figure 2.8: Typed encoding of π into HO, selection from [41]. Above, $\mathtt{fn}(P)$ is a lexicographically ordered sequence of free names in P. Maps $\| \cdot \|$ and $\llbracket \cdot \rrbracket_\sigma$ are in Definition 2.3.1 and Figure 2.7.

$$
\ell \ ::= \ \tau \ \mid \ (\nu\, \widetilde{m})\, n!\langle V\rangle \ \mid \ n?\langle V\rangle \ \mid \ n \oplus l \ \mid \ n\&l
$$

Label τ defines internal actions. Action $(\nu\, \widetilde{m})\, n!\langle V\rangle$ denotes the sending of value V over channel n with a possible empty set of restricted names $\widetilde{m}$ (we may write $n!\langle V\rangle$ when $\widetilde{m}$ is

Terms:

$$\llbracket u!\langle \lambda x.\, Q\rangle.P \rrbracket^2 \overset{\mathtt{def}}{=}$$

$$\begin{cases} (\nu\, a)(u!\langle a\rangle.(\llbracket P\rrbracket^2 \mid *\; a?(y).y?(x).\llbracket Q\rrbracket^2)) & \text{if } \mathtt{fs}(Q) = \emptyset \\ (\nu\, a)(u!\langle a\rangle.(\llbracket P\rrbracket^2 \mid a?(y).y?(x).\llbracket Q\rrbracket^2)) & \text{otherwise} \end{cases}$$

$$\text{where:} \qquad *\, P = \mu X.(P \mid X)$$

$$\begin{aligned} \llbracket u?(x).P\rrbracket^2 &\overset{\mathtt{def}}{=} u?(x).\llbracket P\rrbracket^2 \\ \llbracket x\, u\rrbracket^2 &\overset{\mathtt{def}}{=} (\nu\, s)(x!\langle s\rangle.\overline{s}!\langle u\rangle.\mathbf{0}) \\ \llbracket (\lambda x.\, P)\, u\rrbracket^2 &\overset{\mathtt{def}}{=} (\nu\, s)(s?(x).\llbracket P\rrbracket^2 \mid \overline{s}!\langle u\rangle.\mathbf{0}) \end{aligned}$$

Types:

$$\begin{aligned} (\!(!\langle S \multimap \diamond\rangle;S_1)\!)^2 &\overset{\mathtt{def}}{=} !\langle\langle ?((\!(S)\!)^2);\mathtt{end}\rangle\rangle;(\!(S_1)\!)^2 \\ (\!(?(S \multimap \diamond);S_1)\!)^2 &\overset{\mathtt{def}}{=} ?(\langle ?((\!(S)\!)^2);\mathtt{end}\rangle);(\!(S_1)\!)^2 \end{aligned}$$

Figure 2.9: Typed encoding of HO into π [41].

empty). Dually, the action for value reception is $n?\langle V\rangle$. Actions for select and branch on a label l are denoted $n \oplus l$ and $n\&l$, respectively. We write $\mathtt{fn}(\ell)$ and $\mathtt{bn}(\ell)$ to denote the sets of free/bound names in ℓ, respectively. Given $\ell \neq \tau$, we say ℓ is a *visible action*; we write $\mathtt{subj}(\ell)$ to denote its *subject*. This way, we have: $\mathtt{subj}((\nu\, \widetilde{m})\, n!\langle V\rangle) = \mathtt{subj}(n?\langle V\rangle) = \mathtt{subj}(n \oplus l) = \mathtt{subj}(n\&l) = n$.

The (early) labelled transition system (LTS) for *untyped* processes is given in Figure 2.10. We write $P_1 \xrightarrow{\ell} P_2$ with the usual meaning. The rules are standard [43, 42]; we comment on some of them. A process with an output prefix can interact with the environment with an output action that carries a value V (Rule $\langle\text{Snd}\rangle$). Dually, in Rule $\langle\text{Rv}\rangle$ a receiver process can observe an input of an arbitrary value V. Select and branch processes observe the select and branch actions in Rules $\langle\text{Sel}\rangle$ and $\langle\text{Bra}\rangle$, respectively. Rule $\langle\text{Res}\rangle$ enables an observable action from a process with an outermost restriction, provided that the restricted name does not occur free in the action. If a restricted name occurs free in the carried value of an output action, the process performs scope opening (Rule $\langle\text{New}\rangle$). Rule $\langle\text{Rec}\rangle$ handles recursion unfolding. Rule $\langle\text{Tau}\rangle$ states that two parallel processes which perform dual actions can synchronise by an internal transition. Rules $\langle\text{Par}_L\rangle/\langle\text{Par}_R\rangle$ and $\langle\text{Alpha}\rangle$ define standard treatments for actions under parallel composition and α-renaming.

2.5 Environmental Labelled Transition System

Our typed LTS is obtained by coupling the untyped LTS given before with a labelled transition relation on typing environments, given in Figure 5.6. Building upon the reduction relation for session environments in Definition 2.2.1, such a relation is defined on triples of environments by extending the LTSs in [43, 42]; it is denoted

$$(\Gamma_1, \Lambda_1, \Delta_1) \xrightarrow{\ell} (\Gamma_2, \Lambda_2, \Delta_2)$$

$$\langle\textsc{App}\rangle\quad \frac{}{(\lambda x.\, P)\, V \xrightarrow{\tau} P\{V/x\}} \qquad \langle\textsc{Snd}\rangle\quad \frac{}{n!\langle V\rangle.P \xrightarrow{n!\langle V\rangle} P} \qquad \langle\textsc{Rv}\rangle\quad \frac{}{n?(x).P \xrightarrow{n?\langle V\rangle} P\{V/x\}}$$

$$\langle\textsc{Sel}\rangle\quad \frac{}{s \triangleleft l.P \xrightarrow{s\oplus l} P} \qquad \langle\textsc{Bra}\rangle\quad \frac{j \in I}{s \triangleright \{l_i : P_i\}_{i\in I} \xrightarrow{s\& l_j} P_j} \qquad \langle\textsc{Alpha}\rangle\quad \frac{P \equiv_\alpha Q \qquad Q \xrightarrow{\ell} P'}{P \xrightarrow{\ell} P'}$$

$$\langle\textsc{Res}\rangle\quad \frac{P \xrightarrow{\ell} P' \qquad n \notin \mathtt{fn}(\ell)}{(\nu\, n)\, P \xrightarrow{\ell} (\nu\, n)\, P'} \qquad \langle\textsc{New}\rangle\quad \frac{P \xrightarrow{(\nu\, \widetilde{m})\, n!\langle V\rangle} P' \qquad m_1 \in \mathtt{fn}(V)}{(\nu\, m_1)\, P \xrightarrow{(\nu\, m_1, \widetilde{m})\, n!\langle V\rangle} P'}$$

$$\langle\textsc{Par}_L\rangle\quad \frac{P \xrightarrow{\ell} P' \qquad \mathtt{bn}(\ell) \cap \mathtt{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\ell} P' \mid Q} \qquad \langle\textsc{Tau}\rangle\quad \frac{P \xrightarrow{\ell_1} P' \qquad Q \xrightarrow{\ell_2} Q' \qquad \ell_1 \asymp \ell_2}{P \mid Q \xrightarrow{\tau} (\nu\, \mathtt{bn}(\ell_1) \cup \mathtt{bn}(\ell_2))(P' \mid Q')}$$

$$\langle\textsc{Rec}\rangle\quad \frac{P\{\mu X.P/X\} \xrightarrow{\ell} P'}{\mu X.P \xrightarrow{\ell} P'}$$

Figure 2.10: The Untyped LTS for $\mathsf{HO}\pi$ processes. We omit Rule $\langle\textsc{Par}_R\rangle$.

Recall that Γ admits weakening. Using this principle (not valid for Λ and Δ), we have $(\Gamma', \Lambda_1, \Delta_1) \overset{\ell}{\longmapsto} (\Gamma', \Lambda_2, \Delta_2)$ whenever $(\Gamma, \Lambda_1, \Delta_1) \overset{\ell}{\longmapsto} (\Gamma', \Lambda_2, \Delta_2)$.

Input Actions are defined by Rules [SRv] and [ShRv]. In Rule [SRv] the type of value V and the type of the object associated to the session type on s should coincide. The resulting type tuple must contain the environments associated to V. The dual endpoint $\overline{s}$ cannot be present in the session environment: if it were present the only possible communication would be the interaction between the two endpoints (cf. Rule [Tau]). Following similar principles, Rule [ShRv] defines input actions for shared names.

Output Actions are defined by Rules [SSnd] and [ShSnd]. Rule [SSnd] states the conditions for observing action $(\nu\, \widetilde{m})\, s!\langle V\rangle$ on a type tuple $(\Gamma, \Lambda, \Delta \cdot s{:}S)$. The session environment $\Delta\,,\ s{:}S$ should include the session environment of the sent value V (denoted Δ' in the rule), *excluding* the session environments of names m_j in $\widetilde{m}$ which restrict the scope of value V (denoted Δ_j in the rule). Analogously, the linear variable environment Λ' of V should be included in Λ. The rule defines the scope extrusion of session names in $\widetilde{m}$; consequently, environments associated to their dual endpoints (denoted Δ'_j in the rule) appear in the resulting session environment. Similarly for shared names in $\widetilde{m}$ that are extruded. All free values used for typing V (denoted Λ' and Δ' in the rule) are subtracted from the resulting type tuple. The prefix of session s is consumed by the action. Rule [ShSnd] follows similar ideas for output actions on shared names: the name must be typed with $\langle U\rangle$; conditions on value V are identical to those on Rule [SSnd].

Other Actions Rules [Sel] and [Bra] describe actions for select and branch. Rule [Tau] defines internal transitions: it reduces the session environment (cf. Definition 2.2.1) or keeps it unchanged.

We illustrate Rule [SSnd] by means of an example:

Example 2.5.1. Consider environment tuple $(\Gamma;\, \emptyset;\, s :!\langle(!\langle S\rangle;\mathtt{end}) \multimap \diamond\rangle;\mathtt{end}, s' : S)$ and typed value $V = \lambda x.\, x!\langle s'\rangle.m?(z).\mathbf{0}$ with

$$\Gamma; \emptyset; s' : S, m :?(\mathtt{end});\mathtt{end} \vdash V \triangleright (!\langle S\rangle;\mathtt{end}) \multimap \diamond$$

Then, by Rule [SSnd], we can derive:

$$(\Gamma; \emptyset; s :!\langle(!\langle S\rangle;\mathtt{end}) \multimap \diamond\rangle;\mathtt{end}, s' : S) \xrightarrow{(\nu\, m)\, s!\langle V\rangle} (\Gamma; \emptyset; s : \mathtt{end}, \overline{m} :!\langle\mathtt{end}\rangle;\mathtt{end})$$

Observe how the protocol along s is partially consumed; also, the resulting session environment is extended with $\overline{m}$, the dual endpoint of the extruded name m.

Notation 2.5.1. *Given a value V of type U, we sometimes annotate the output action $(\nu\, \widetilde{m})\, n!\langle V\rangle$ with the type of V as $(\nu\, \widetilde{m})\, n!\langle V : U\rangle$.*

The typed LTS combines the LTSs in Figure 2.10 and Figure 5.6.

Definition 2.5.1 (Typed Transition System)**.** A *typed transition relation* is a typed relation $\Gamma; \Delta_1 \vdash P_1 \xrightarrow{\ell} \Delta_2 \vdash P_2$ where:

1. $P_1 \xrightarrow{\ell} P_2$ and
2. $(\Gamma, \emptyset, \Delta_1) \xrightarrow{\ell} (\Gamma, \emptyset, \Delta_2)$ with $\Gamma; \emptyset; \Delta_i \vdash P_i \triangleright \diamond$ $(i = 1, 2)$.

We write $\Rightarrow$ for the reflexive and transitive closure of $\rightarrow$, $\overset{\ell}{\Rightarrow}$ for the transitions $\Rightarrow\xrightarrow{\ell}\Rightarrow$, and $\overset{\hat{\ell}}{\Rightarrow}$ for $\overset{\ell}{\Rightarrow}$ if $\ell \neq \tau$ otherwise $\Rightarrow$.

A typed transition relation requires type judgements with an empty Λ, i.e., an empty environment for linear higher-order types. Notice that for open process terms (i.e., with free variables), we can always apply Rule [EProm] (cf. Figure 2.5) and obtain an empty Λ. As it will be clear below (cf. Definition 5.3.12), we will be working with closed process terms, i.e., processes without free variables.

2.6 Typed Relations

We now define *typed relations* and *contextual equivalence* (i.e., barbed congruence). To define typed relations, we first define *confluence* over session environments Δ. Recall that Δ captures session communication, which is deterministic. The notion of confluence allows us to abstract away from alternative computation paths that may arise due to non-interfering reductions of session names.

Definition 2.6.1 (Session Environment Confluence)**.** Two session environments Δ_1 and Δ_2 are *confluent*, denoted $\Delta_1 \rightleftharpoons \Delta_2$, if there exists a Δ such that: i) $\Delta_1 \longrightarrow^* \Delta$ and ii) $\Delta_2 \longrightarrow^* \Delta$ (here we write $\longrightarrow^*$ for the multi-step reduction in Definition 2.2.1).

We illustrate confluence by means of an example:

[SRv]
$$\frac{\overline{s} \notin \mathtt{dom}(\Delta) \qquad \Gamma; \Lambda'; \Delta' \vdash V \triangleright U}{(\Gamma; \Lambda; \Delta, s : ?(U);S) \xrightarrow{s?\langle V \rangle} (\Gamma; \Lambda, \Lambda'; \Delta, \Delta', s : S)}$$

[ShRv]
$$\frac{\Gamma; \emptyset; \emptyset \vdash a \triangleright \langle U \rangle \qquad \Gamma; \Lambda'; \Delta' \vdash V \triangleright U}{(\Gamma; \Lambda; \Delta) \xrightarrow{a?\langle V \rangle} (\Gamma; \Lambda, \Lambda'; \Delta, \Delta')}$$

[SSnd]
$$\frac{\begin{array}{ccc}\Gamma, \Gamma'; \Lambda'; \Delta' \vdash V \triangleright U & \Gamma'; \emptyset; \Delta_j \vdash m_j \triangleright U_j & \overline{s} \notin \mathtt{dom}(\Delta) \\ \Delta' \backslash (\cup_j \Delta_j) \subseteq (\Delta, s : S) & \Gamma'; \emptyset; \Delta'_j \vdash \overline{m}_j \triangleright U'_j & \Lambda' \subseteq \Lambda\end{array}}{(\Gamma; \Lambda; \Delta, s : !\langle U \rangle;S) \xrightarrow{(\nu\, \widetilde{m})\, s!\langle V \rangle} (\Gamma, \Gamma'; \Lambda \backslash \Lambda'; (\Delta, s : S, \cup_j \Delta'_j) \backslash \Delta')}$$

[ShSnd]
$$\frac{\begin{array}{ccc}\Gamma, \Gamma'; \Lambda'; \Delta' \vdash V \triangleright U & \Gamma'; \emptyset; \Delta_j \vdash m_j \triangleright U_j & \Gamma; \emptyset; \emptyset \vdash a \triangleright \langle U \rangle \\ \Delta' \backslash (\cup_j \Delta_j) \subseteq \Delta & \Gamma'; \emptyset; \Delta'_j \vdash \overline{m}_j \triangleright U'_j & \Lambda' \subseteq \Lambda\end{array}}{(\Gamma; \Lambda; \Delta) \xrightarrow{(\nu\, \widetilde{m})\, a!\langle V \rangle} (\Gamma, \Gamma'; \Lambda \backslash \Lambda'; (\Delta, \cup_j \Delta'_j) \backslash \Delta')}$$

[Sel]
$$\frac{\overline{s} \notin \mathtt{dom}(\Delta) \qquad j \in I}{(\Gamma; \Lambda; \Delta, s : \oplus\{l_i : S_i\}_{i \in I}) \xrightarrow{s \oplus l_j} (\Gamma; \Lambda; \Delta, s : S_j)}$$

[Bra]
$$\frac{\overline{s} \notin \mathtt{dom}(\Delta) \quad j \in I}{(\Gamma; \Lambda; \Delta, s : \&\{l_i : T_i\}_{i \in I}) \xrightarrow{s \& l_j} (\Gamma; \Lambda; \Delta, s : S_j)}$$

[Tau]
$$\frac{\Delta_1 \longrightarrow \Delta_2 \vee \Delta_1 = \Delta_2}{(\Gamma; \Lambda; \Delta_1) \xrightarrow{\tau} (\Gamma; \Lambda; \Delta_2)}$$

Figure 2.11: Labelled Transition System for Typed Environments.

Example 2.6.1 (Session Environment Confluence)**.** Consider the (balanced) session environments:

$$\begin{array}{rcl}\Delta_1 & = & \{s_1 : T_1, s_2 : ?(U_2);\mathtt{end}, \overline{s_2} : !\langle U_2 \rangle;\mathtt{end}\} \\ \Delta_2 & = & \{s_1 : T_1, s_2 : !\langle U_1 \rangle;?(U_2);\mathtt{end}, \overline{s_2} : ?(U_1);!\langle U_2 \rangle;\mathtt{end}\}\end{array}$$

Following Definition 2.2.1, we have that $\Delta_1 \longrightarrow \{s_1 : T_1, s_2 : \mathtt{end}, \overline{s_2} : \mathtt{end}\}$ and $\Delta_2 \longrightarrow\longrightarrow \{s_1 : T_1, s_2 : \mathtt{end}, \overline{s_2} : \mathtt{end}\}$. Therefore, Δ_1 and Δ_2 are confluent. $\Box$

Typed relations relate only closed processes whose session environments are balanced and confluent:

Definition 2.6.2 (Typed Relation)**.** We say that a binary relation over typing judgements

$$\Gamma; \emptyset; \Delta_1 \vdash P_1 \triangleright \diamond \; \Re \; \Gamma; \emptyset; \Delta_2 \vdash P_2 \triangleright \diamond$$

is a *typed relation* whenever:

1. P_1 and P_2 are closed;

$$
\begin{array}{rcl@{\qquad}rcl}
\llparenthesis ?(U);S \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & u?(x).(t!\langle u\rangle.\mathbf{0} \mid \llparenthesis U \rrparenthesis^{x}) & \llparenthesis S \rrparenthesis_{\mathsf{c}} & \stackrel{\mathsf{def}}{=} & s \;\; (s \text{ fresh}) \\
\llparenthesis !\langle U\rangle;S \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & u!\langle \llparenthesis U \rrparenthesis_{\mathsf{c}}\rangle.t!\langle u\rangle.\mathbf{0} & \llparenthesis \langle S\rangle \rrparenthesis_{\mathsf{c}} & \stackrel{\mathsf{def}}{=} & a \;\; (a \text{ fresh}) \\
\llparenthesis \oplus\{l : S\} \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & u \triangleleft l.t!\langle u\rangle.\mathbf{0} & \llparenthesis \langle L\rangle \rrparenthesis_{\mathsf{c}} & \stackrel{\mathsf{def}}{=} & a \;\; (a \text{ fresh}) \\
\llparenthesis \&\{l_i : S_i\}_{i\in I} \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & u \triangleright \{l_i : t_i!\langle u\rangle.\mathbf{0}\}_{i\in I} & \llparenthesis U \to \diamond \rrparenthesis_{\mathsf{c}} & \stackrel{\mathsf{def}}{=} & \lambda x.\, \llparenthesis U \rrparenthesis^{x} \\
\llparenthesis \mu\mathsf{t}.S \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & \llparenthesis S\{\mathsf{end}/\mathsf{t}\} \rrparenthesis^{u} & \llparenthesis U \multimap \diamond \rrparenthesis_{\mathsf{c}} & \stackrel{\mathsf{def}}{=} & \lambda x.\, \llparenthesis U \rrparenthesis^{x} \\
\llparenthesis \mathtt{end} \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & \mathbf{0} & & & \\
\llparenthesis \langle S\rangle \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & u!\langle \llparenthesis S \rrparenthesis_{\mathsf{c}}\rangle.t!\langle u\rangle.\mathbf{0} & & & \\
\llparenthesis \langle L\rangle \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & u!\langle \llparenthesis L \rrparenthesis_{\mathsf{c}}\rangle.t!\langle u\rangle.\mathbf{0} & & & \\
\llparenthesis U \to \diamond \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & u\, \llparenthesis U \rrparenthesis_{\mathsf{c}} & & & \\
\llparenthesis U \multimap \diamond \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & u\, \llparenthesis U \rrparenthesis_{\mathsf{c}} & & &
\end{array}
$$

Figure 2.12: Characteristic processes (left) and characteristic values (right).

2. Δ_1 and Δ_2 are balanced (cf. Definition 2.2.1); and

3. $\Delta_1 \rightleftharpoons \Delta_2$ (cf. Definition 5.3.11).

Notation 2.6.1 (Typed Relations)**.** *We write*

$$\Gamma; \Delta_1 \vdash P_1 \;\Re\; \Delta_2 \vdash P_2$$

to denote the typed relation $\Gamma; \emptyset; \Delta_1 \vdash P_1 \triangleright \diamond \;\Re\; \Gamma; \emptyset; \Delta_2 \vdash P_2 \triangleright \diamond$.

2.7 Characteristic Values and Refined LTS

We first define characteristic processes/values and trigger values:

Definition 2.7.1 (Characteristic Process and Values)**.** Let u and U be a name and a type, respectively. The *characteristic process* of U (along u), denoted $\llparenthesis U \rrparenthesis^{u}$, and the *characteristic value* of U, denoted $\llparenthesis U \rrparenthesis_{\mathsf{c}}$, are defined in Figure 2.12.

Definition 2.7.2 (Trigger Value)**.** Given a fresh name t, the *trigger value* on t is defined as the abstraction $\lambda x.\, t?(y).(y\, x)$.

Definition 2.7.3 (Refined Typed Labelled Transition System)**.** The refined typed labelled transition relation on typing environments

$$(\Gamma_1; \Lambda_1; \Delta_1) \overset{\ell}{\longmapsto} (\Gamma_2; \Lambda_2; \Delta_2)$$

is defined on top of the rules in Figure 5.6 using the following rules:

$$\textsc{[Tr]}\quad \frac{(\Gamma_1; \Lambda_1; \Delta_1) \xrightarrow{\ell} (\Gamma_2; \Lambda_2; \Delta_2) \qquad \ell \neq n?\langle V\rangle}{(\Gamma_1; \Lambda_1; \Delta_1) \overset{\ell}{\longmapsto} (\Gamma_2; \Lambda_2; \Delta_2)}$$

$$\textsc{[RRcv]}\quad \frac{(\Gamma_1; \Lambda_1; \Delta_1) \xrightarrow{n?\langle V\rangle} (\Gamma_2; \Lambda_2; \Delta_2) \qquad V = m \vee V \equiv \llparenthesis U \rrparenthesis_{\mathsf{c}} \vee V \equiv \lambda x.\, t?(y).(y\, x)\ t \text{ fresh}}{(\Gamma_1; \Lambda_1; \Delta_1) \xmapsto{n?\langle V\rangle} (\Gamma_2; \Lambda_2; \Delta_2)}$$

Then, the refined typed labelled transition system

$$\Gamma; \Delta_1 \vdash P_1 \overset{\ell}{\hookrightarrow} \Delta_2 \vdash P_2$$

is given as in Definition 5.3.10, replacing the requirement $(\Gamma, \emptyset, \Delta_1) \xrightarrow{\ell} (\Gamma, \emptyset, \Delta_2)$ with $(\Gamma_1; \Lambda_1; \Delta_1) \overset{\ell}{\hookrightarrow} (\Gamma_2; \Lambda_2; \Delta_2)$, as just defined. Following Definition 5.3.10, we write $\Mapsto$ for the reflexive and transitive closure of $\overset{\tau}{\hookrightarrow}$, $\overset{\ell}{\Mapsto}$ for the transitions $\Mapsto\overset{\ell}{\hookrightarrow}\Mapsto$, and $\overset{\hat{\ell}}{\Mapsto}$ for $\overset{\ell}{\Mapsto}$ if $\ell \neq \tau$ otherwise $\Mapsto$.

2.8 Higher-Order and Characteristic Bisimilarities ($\approx^{\mathtt{H}}$ and $\approx^{\mathtt{C}}$)

Having presented a refined LTS on $\mathsf{HO}\pi$ processes, we now recall the definitions of *higher-order bisimilarity* and *characteristic bisimilarity*, two tractable bisimilarity relations. The two bisimulations use two different trigger processes:

$$t \hookleftarrow_{\mathtt{H}} V \overset{\mathsf{def}}{=} \begin{cases} t?(x).(\nu\, s)(s?(y).(x\, y) \mid \overline{s}!\langle V\rangle.\mathbf{0}) & \text{if } V \text{ is a first-order value} \\ t?(x).(\nu\, s)(s?(y).(y\, x) \mid \overline{s}!\langle V\rangle.\mathbf{0}) & \text{if } V \text{ is a higher-order value} \end{cases} \tag{2.2}$$

$$t \Leftarrow_{\mathtt{C}} V : U \overset{\mathsf{def}}{=} t?(x).(\nu\, s)(s?(y).\llbracket U \rrbracket^{y} \mid \overline{s}!\langle V\rangle.\mathbf{0}) \tag{2.3}$$

The process in (2.2) is called *higher-order trigger process*, while process in (2.3) is called *characteristic trigger process*. Notice that while in (2.2) there is a higher-order input on t, in (2.3) the variable x does not play any rôle.

We use higher-order trigger processes to define *higher-order bisimilarity*:

Definition 2.8.1 (Higher-Order Bisimilarity)**.** A typed relation $\Re$ is a *higher-order bisimulation* if for all $\Gamma; \Delta_1 \vdash P_1 \Re \Delta_2 \vdash Q_1$

1) Whenever $\Gamma; \Delta_1 \vdash P_1 \xmapsto{(\nu\, \widetilde{m_1})\, n!\langle V_1\rangle} \Delta_1' \vdash P_2$, there exist Q_2, V_2, Δ_2' such that $\Gamma; \Delta_2 \vdash Q_1 \overset{(\nu\, \widetilde{m_2})\, n!\langle V_2\rangle}{\Longmapsto} \Delta_2' \vdash Q_2$ and, for a fresh t,
$$\Gamma; \Delta_1'' \vdash (\nu\, \widetilde{m_1})(P_2 \mid t \hookleftarrow_{\mathtt{H}} V_1) \;\Re\; \Delta_2'' \vdash (\nu\, \widetilde{m_2})(Q_2 \mid t \hookleftarrow_{\mathtt{H}} V_2)$$

2) For all $\Gamma; \Delta_1 \vdash P_1 \overset{\ell}{\hookrightarrow} \Delta_1' \vdash P_2$ such that ℓ is not an output, there exist Q_2, Δ_2' such that $\Gamma; \Delta_2 \vdash Q_1 \overset{\hat{\ell}}{\Mapsto} \Delta_2' \vdash Q_2$ and $\Gamma; \Delta_1' \vdash P_2 \Re \Delta_2' \vdash Q_2$; and

3) The symmetric cases of 1 and 2.

The largest such bisimulation is called *higher-order bisimilarity*, denoted by $\approx^{\mathtt{H}}$.

We exploit characteristic trigger processes to define *characteristic bisimilarity*:

Definition 2.8.2 (Characteristic Bisimilarity)**.** A typed relation $\Re$ is a *characteristic bisimulation* if for all $\Gamma; \Delta_1 \vdash P_1 \Re \Delta_2 \vdash Q_1$,

1) Whenever $\Gamma; \Delta_1 \vdash P_1 \xmapsto{(\nu\, \widetilde{m_1})\, n!\langle V_1 : U_1\rangle} \Delta_1' \vdash P_2$ then there exist Q_2, V_2, Δ_2' such that $\Gamma; \Delta_2 \vdash Q_1 \overset{(\nu\, \widetilde{m_2})\, n!\langle V_2 : U_2\rangle}{\Longmapsto} \Delta_2' \vdash Q_2$ and, for a fresh t,
$$\Gamma; \Delta_1'' \vdash (\nu\, \widetilde{m_1})(P_2 \mid t \Leftarrow_{\mathtt{C}} V_1 : U_1) \;\Re\; \Delta_2'' \vdash (\nu\, \widetilde{m_2})(Q_2 \mid t \Leftarrow_{\mathtt{C}} V_2 : U_2)$$

2) For all $\Gamma; \Delta_1 \vdash P_1 \overset{\ell}{\hookrightarrow} \Delta_1' \vdash P_2$ such that ℓ is not an output, there exist Q_2, Δ_2' such that $\Gamma; \Delta_2 \vdash Q_1 \overset{\hat{\ell}}{\Mapsto} \Delta_2' \vdash Q_2$ and $\Gamma; \Delta_1' \vdash P_2 \Re \Delta_2' \vdash Q_2$; and

3) The symmetric cases of 1 and 2.

The largest such bisimulation is called *characteristic bisimilarity*, denoted by $\approx^{\mathtt{C}}$.

3 Bit-vector Typestate Analysis

3.1 Introduction

Industrial-scale software is generally composed of multiple interacting components, which are typically produced separately. As a result, software integration is a major source of bugs [51]. Many integration bugs can be attributed to violations of *code contracts*. Because these contracts are implicit and informal in nature, the resulting bugs are particularly insidious. To address this problem, formal code contracts are an effective solution [24], because static analyzers can automatically check whether client code adheres to ascribed contracts.

Typestate is a fundamental concept in ensuring the correct use of contracts and APIs. A typestate refines the concept of a type: whereas a type denotes the valid operations on an object, a typestate denotes operations valid on an object in its *current program context* [55]. Typestate analysis is a technique used to enforce temporal code contracts. In object-oriented programs, where objects change state over time, typestates denote the valid sequences of method calls for a given object. The behavior of the object is prescribed by the collection of typestates, and each method call can potentially change the object's typestate.

Given this, it is natural for static typestate checkers, such as FUGUE [19], SAFE [60], and INFER's TOPL checker [2], to define the analysis property using Deterministic Finite Automata (DFAs). The abstract domain of the analysis is a set of states in the DFA; each operation on the object modifies the set of possible reachable states. If the set of abstract states contains an error state, then the analyzer warns the user that a code contract may be violated. Widely applicable and conceptually simple, DFAs are the de facto model in typestate analyses.

Here we target the analysis of realistic code contracts in low-latency environments such as, e.g., Integrated Development Environments (IDEs) [57, 56]. In this context, to avoid noticeable disruptions in the users' workflow, the analysis should ideally run *under a second* [3]. However, relying on DFAs jeopardizes this goal, as it can lead to scalability issues. Consider, e.g., a class with n methods in which each method *enables* another one and then *disables* itself: the contract can lead to a DFA with 2^n states. Even with a small n, such a contract can be impractical to codify manually and will likely result in sub-par performance.

Interestingly, many practical contracts do not require a full DFA. In our enable/disable example, the method dependencies are *local* to a subset of methods: a enabling/disabling relation is established between pairs of methods. DFA-based approaches have a *whole class* expressivity; as a result, local method dependencies can impact transitions of unrelated methods. Thus, using DFAs for contracts that specify dependencies that are local to each method (or to a few methods) is redundant and/or prone to inefficient implementations. Based on this observation, we present a *lightweight* typestate analyzer for *locally dependent* code contracts in low-latency environments. It rests upon two insights:

1. *Allowed and disallowed sequences of method calls for objects can be succinctly specified without using DFAs.* To unburden the task of specifying typestates, we introduce *lightweight annotations* to specify *method dependencies* as annotations on methods. Lightweight annotations can specify code contracts for usage scenarios commonly encountered when using libraries such as File, Stream, Socket, etc. in considerably fewer lines of code than DFAs.

2. *A sub-class of DFAs suffices to express many useful code contracts.* To give semantics to lightweight annotations, we define *Bit-Vector Finite Automata (BFAs)*: a sub-class of DFAs whose analysis uses *bit-vector* operations. In many practical scenarios, BFAs suffice to capture information about the enabled and disabled methods at a given point. Because this information can be codified using bit-vectors, associated static analyses can be performed efficiently; in particular, our technique is not sensitive to the number of BFA states, which in turn ensures scalability with contract and program size.

We have implemented our lightweight typestate analysis in the industrial-strength static analyzer INFER [1]. Our analysis exhibits concrete usability and performance advantages and is expressive enough to encode many relevant typestate properties in the literature. On average, compared to state-of-the-art typestate analyses, our approach requires less annotations than DFA-based analyzers and does not exhibit slow-downs due to state increase. We summarise our contributions as follows:

- A specification language for typestates based on *lightweight annotations* (Section 3.2). Our language rests upon BFAs, a new sub-class of DFA based on bit-vectors.
- A lightweight analysis technique for code contracts, implemented in INFER (our artifact is available at [8]).[1]
- Extensive evaluations for our lightweight analysis technique, which demonstrate considerable gains in performance and usability (Section 3.5).

3.2 Bit-vector Typestate Analysis

3.2.1 Annotation Language

We introduce BFA specifications, which succinctly encode temporal properties by describing *local method dependencies*, thus avoiding the need for a full DFA specification. BFA specifications define code contracts by using atomic combinations of annotations '`@Enable`(n)' and '`@Disable`(n)', where n is a set of method names. Intuitively:

- '`@Enable`(n) m' asserts that invoking method m makes calling methods in n valid in a continuation.
- Dually, '`@Disable`(n) m' asserts that a call to m disables calls to all methods in n in the continuation.

Notation 3.2.1. *We define some base sets and notations.*

- *We write Classes to denote the finite set of all classes under consideration. We use $c, c', \ldots$ to denote elements of Classes.*
- *The set $\Sigma_c = \{m^{\uparrow}, m_1, \ldots, m_n, m^{\downarrow}\}$ denotes the n methods of a class c. In Σ_c, both $m^{\uparrow}$ and $m^{\downarrow}$ are notations reserved for the constructor and destructor methods of the class, respectively. We assume a single constructor and destructor for simplicity and clarity; our formalism can be extended to support multiple constructors without difficulties.*
- *The set $\Sigma_c^{\bullet}$ is defined as $\Sigma_c \setminus \{m^{\uparrow}, m^{\downarrow}\}$. For convenience, we will assume a total ordering on $\Sigma_c^{\bullet}$; this will be useful when defining BFAs in the next section.*

We will often use E and D to denote subsets of $\Sigma_c^{\bullet}$. Also, we shall write $\tilde{x}$ to denote finite sequences of elements $x_1, \ldots, x_k$ (with $k > 0$).

[1]Our code is available at `https://github.com/aalen9/lfa.git`

Definition Following the above intuitions on '`@Enable`(n) m' and '`@Disable`(n) m', we define BFA annotations per method and a corresponding notion of valid method sequences:

Definition 3.2.1 (Annotation Language)**.** Let $c \in$ *Classes* such that $\Sigma_c = \{m^{\uparrow}, m_1, \ldots, m_n, m^{\downarrow}\}$. We have:

- The constructor method $m^{\uparrow}$ is annotated by

  $$\texttt{@Enable}(E^c)\ \texttt{@Disable}(D^c)\ m^{\uparrow}$$

  where $E^c \cup D^c = \Sigma_c^{\bullet}$ and $E^c \cap D^c = \emptyset$;

- Each $m_i \in \Sigma_c^{\bullet}$ is annotated by

  $$\texttt{@Enable}(E_i)\ \texttt{@Disable}(D_i)\ m_i$$

  where $E_i \subseteq \Sigma_c^{\bullet}$, $D_i \subseteq \Sigma_c^{\bullet}$, and $E_i \cap D_i = \emptyset$.

Let $\tilde{x} = m^{\uparrow}, x_1, x_2, \ldots$ be a sequence where each $x_i \in \Sigma_c^{\bullet}$. We say that $\tilde{x}$ is *valid (w.r.t. annotations)* if for all substrings $\tilde{x}' = x_i, \ldots, x_k$ of $\tilde{x}$ such that $x_k \in D_i$ there is j $(i < j \leq k)$ such that $x_k \in E_j$.

The formal semantics for these specifications is given in Section 3.2.2. We note that if E_i or D_i is $\emptyset$ then we omit the corresponding annotation.

Derived Annotations The annotation language can be used to derive other useful annotations:

$$\texttt{@EnableOnly}(E_i)\ m_i \overset{\text{def}}{=} \texttt{@Enable}(E_i)\ \texttt{@Disable}(C \setminus E_i)\ m_i$$
$$\texttt{@DisableOnly}(D_i)\ m_i \overset{\text{def}}{=} \texttt{@Disable}(D_i)\ \texttt{@Enable}(C \setminus E_i)\ m_i$$
$$\texttt{@EnableAll}\ m_i \overset{\text{def}}{=} \texttt{@Enable}(C)\ m_i$$

This way, the annotation '`@EnableOnly`(E_i) m_i' asserts that a call to method m_i enables only calls to methods in E_i while disabling all other methods in $\Sigma_c^{\bullet}$. The annotation '`@DisableOnly`(D_i) m_i' is defined dually. Finally, the annotation '`@EnableAll` m_i' asserts that a call to method m_i enables all methods in a class; an annotation '`@DisableAll` m_i' can be defined dually.

Examples We illustrate the expressivity and usability of BFA annotations by the means of the following example. We consider the `SparseLU` class from `Eigen C++` library.[2] For brevity, we consider representative methods for a typestate specification (we also omit return types):

```
class SparseLU {
    void analyzePattern(Mat a);
    void factorize(Mat a);
    void compute(Mat a);
    void solve(Mat b);  }
```

The class `SparseLU` implements a lower-upper (LU) decomposition of a sparse matrix. `Eigen`'s implementation uses assertions to dynamically check that: (i) `analyzePattern` is called prior to `factorize` and (ii) `factorize` or `compute` are called prior to `solve`. At a high-level, this contract tells us that `compute` (or `analyzePattern().factorize()`) prepares resources for invoking `solve`.

Some method call sequences do not cause errors but have redundancies. For example, we can disallow consecutive calls to `compute` in sequences such as, e.g.,

[2]`https://eigen.tuxfamily.org/dox/classEigen_1_1SparseLU.html`

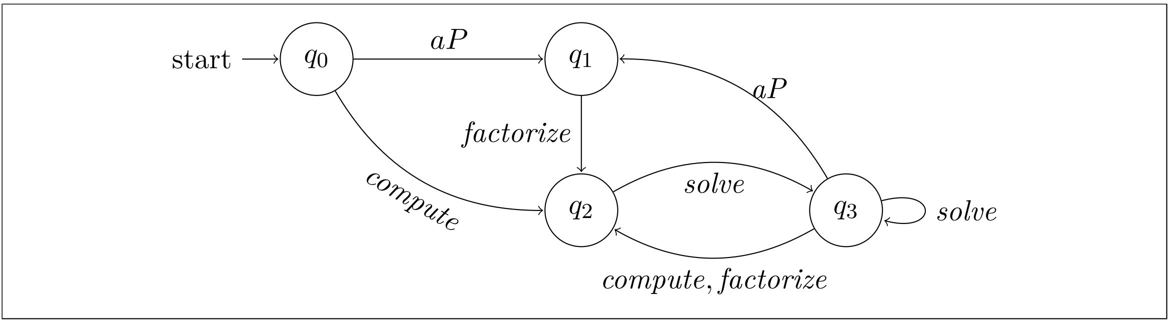


Figure 3.1: SparseLU DFA

```
class SparseLU {
    states q0, q1, q2, q3;
    @Pre(q0) @Post(q1)
    @Pre(q3) @Post(q1)
    void analyzePattern(Mat a);
    @Pre(q1) @Post(q2)
    @Pre(q3) @Post(q2)
    void factorize(Mat a);
    @Pre(q0) @Post(q2)
    @Pre(q3) @Post(q2)
    void compute(Mat a);
    @Pre(q2) @Post(q3)
    @Pre(q3)
    void solve(Mat b);  }
```

Listing (3.1) SparseLU DFA Contract

```
class SparseLU {


    @EnableOnly(factorize)
    void analyzePattern(Mat a);

    @EnableOnly(solve)
    void factorize(Mat a);

    @EnableOnly(solve)
    void compute(Mat a);

    @EnableAll
    void solve(Mat b);  }
```

Listing (3.2) SparseLU BFA Contract

‘`compute().compute().solve()`’

as the result of the first call to `compute` is never used. Also, because `compute` is essentially implemented as ‘`analyzePattern().factorize()`’, it is also redundant to call `factorize` after `compute`.

Figure 3.1 gives the corresponding DFA that substitutes dynamic checks and avoids redundancies. (In the figure, and in the following, we write aP to denote/abbreviate ‘`analyzePattern`’.) Following the literature [19], this DFA can be annotated inside the definition of the class `SparseLU` as in Listing 3.1: states are listed in the class header and transitions are specified by *@Pre* and *@Post* conditions on methods. Already in this small example, this DFA specification is too low-level and presents high annotation overheads, which makes it unreasonable for software engineers to annotate their APIs with.

The entire contract for the `SparseLU` class can be succinctly specified using BFA annotations as in Listing 3.2. In this case, the starting state is unspecified, as it is determined by annotations. In fact, methods that are not *guarded* by other methods (like `solve` is guarded by `compute`) are enabled in the starting state. This condition can be overloaded by specifying annotations on the constructor method. Remarkably, the contract can be specified with only 4 annotations; in contrast, the corresponding DFA requires 8 annotations plus 4 states specified in the class header.

Another difference concerns the treatment of local method dependencies: a small change in BFA annotations can result in a substantial change of the corresponding DFA. To see this, let $\{m_1, m_2, m_3, \ldots, m_n\}$ be methods of some class with an associated DFA (with set of states Q), in which m_1 and m_2 are enabled in each state of Q. Adding an annotation such as ‘`@Enable(m2) m1`’ doubles the number of states of the required DFA, as we need the set of states Q where m_2 is enabled in each state, but also states from Q with m_2 disabled in each state. Accordingly, transitions have to be duplicated for the new states and the remaining methods $(m_3, \dots, m_n)$.

3.2.2 Bit-vector Finite Automata

We define Bit-vector Finite Automata (BFA, in the following): a class of DFAs that captures enabling/disabling dependencies between the methods of a class (cf. Definition 3.2.1) leveraging a bit-vector abstraction on typestates.

Definition 3.2.2 (Sets and Bit-vectors)**.** Let $\mathcal{B}^n$ denote the set of bit-vectors of length $n > 0$. We write $b, b', \dots$ to denote elements of $\mathcal{B}^n$, with $b[i]$ denoting the i-th bit in b. Given a finite set S with $|S| = n$, every $A \subseteq S$ can be represented by a bit-vector $b_A \in \mathcal{B}^n$, obtained via the usual characteristic function.

By a small abuse of notation, given sets $A, A' \subseteq S$, we may write $A \subseteq A'$ to denote the subset operation applied on b_A and $b_{A'}$ (and similarly for $\cup, \cap$, and $\setminus$).

We first define a BFA per class. We assume $c \in \mathit{Classes}$ and $\Sigma_c^\bullet = \{m_1, \dots, m_n\}$ be as described in Notation 3.2.1. Given that c has n methods, we consider states q_b, where, following Definition 3.2.2, the bit-vector $b_A \in \mathcal{B}^n$ denotes the set of methods $A \subseteq \Sigma_c^\bullet$ enabled at that point. We assume that the bit-vector representation of the subset A is consistent with respect to the total ordering on $\Sigma_c^\bullet$, in the sense that bit $b[i]$ corresponds to $m_i \in \Sigma_c^\bullet$. We often write 'b' (and q_b) rather than 'b_A' (and 'q_{b_A}'), for simplicity. As we will see, the intent is that if $m_i \in b$ (resp. $m_i \notin b$), then the i-th method is enabled (resp. disabled) in q_b.

Definition 3.2.3, given next, gives a mapping from methods to triples of bit-vectors, denoted $\mathcal{L}_c$. Given $k > 0$, let us write 1^k (resp. 0^k) to denote a sequence of 1s (resp. 0s) of length k.

The initial state is determined by E^c, the set of enabling annotations on the constructor.

Definition 3.2.3 (Mapping $\mathcal{L}_c$)**.** Given a class c, we define $\mathcal{L}_c$ as a mapping from methods to triples of subsets of Σ_c as follows

$$\mathcal{L}_c : \Sigma_c \to \mathcal{P}(\Sigma_c) \times \mathcal{P}(\Sigma_c) \times \mathcal{P}(\Sigma_c)$$

Given $m_i \in \Sigma_c^\bullet$, we shall write E_i, D_i, and P_i to denote each of the elements of the triple $\mathcal{L}_c(m_i)$. Similarly, we write E^c, D^c, and P^c to denote the elements of the triple $\mathcal{L}_c(m^\uparrow)$. The mapping $\mathcal{L}_c$ is induced by the annotations in class c: for each m_i, the sets E_i and D_i are explicit, and P_i is simply the singleton $\{m_i\}$. This singleton formulation is convenient to define the domain of the compositional analysis in Section 3.3.2: as we will see later, it allows us to uniformly treat method calls and procedure calls which can have more elements in pre-set P_i.

We impose some natural well-formedness conditions on the BFA mapping.

Definition 3.2.4 ($\mathit{well\_formed}(\mathcal{L}_c)$)**.** Let c, Σ_c, and $\mathcal{L}_c$ be a class, its method set, and its BFA mapping, respectively. Then, $\mathsf{well\_formed}(\mathcal{L}_c) = \mathbf{true}$ iff the following conditions hold:

- $\mathcal{L}_c(m^\uparrow) = \langle E^c, D^c, \emptyset \rangle$ such that $E^c \cup D^c = \Sigma_c^\bullet$ and $E^c \cap D^c = \emptyset$;
- for $m_i \in \Sigma_c^\bullet$ we have $\mathcal{L}_c(m_i) = \langle E_i, D_i, \{m_i\} \rangle$ such that $E_i, D_i \subseteq \Sigma_c^\bullet$ and $E_i \cap D_i = \emptyset$.

The first condition says that the constructor's enabling and disabling sets must be disjunctive and complementary with respect to $\Sigma_c^\bullet$; this will be convenient later when defining the compositional analysis algorithm in Section 3.3. The second condition ensures that every method's enabling and disabling sets are disjunctive. Furthermore, by taking $E_i, D_i \in \Sigma_c^\bullet$ we ensure that the annotations of method m_i cannot refer to the constructor nor the destructor (see Notation 3.2.1).

In a BFA, transitions between states $(q_b, q_{b'}, \cdots)$ are determined by $\mathcal{L}_c$. Given $m_i \in \Sigma_c$, we have $j \in E_i$ if and only if m_i enables m_j; similarly, we have $k \in D_i$ if and only if m_i disables m_k. A transition from q_b labeled by method m_i leads to state $q_{b'}$, where b' is determined by $\mathcal{L}_c$ using b. Such a transition is defined only if a pre-condition for m_i is met in state q_b, i.e., $P \subseteq b$. In that case, $b' = (b \cup E_i) \setminus D_i$.

These intuitions should serve to illustrate our approach and, in particular, the local nature of enabling/disabling dependencies between methods. The following definition makes them precise.

Definition 3.2.5 (BFA)**.** Given a $c \in \mathit{Classes}$ with $n > 0$ methods, a BFA for c is defined as a tuple $M = (Q, \Sigma_c^\bullet, \delta, q_{E^c}, \mathcal{L}_c)$ where:

- Q is a finite set of states $q_b, q_{b'}, \ldots$, where $b, b', \ldots \in \mathcal{B}^n$;
- $\Sigma_c^\bullet = \{m_1, \ldots, m_n\}$ is the alphabet (method identities);
- q_{E^c} is the starting state (recall that E^c is enabling set of a constructor);
- $\mathcal{L}_c$ is a BFA mapping (cf. Definition 3.2.3).
- $\delta : Q \times \Sigma_c \to Q$ is the transition function, where

$$\delta(q_b, m_i) = q_{b'}$$

  with $b' = (b \cup E_i) \setminus D_i$, if $P_i \subseteq b$, and is undefined otherwise.

We remark that in a BFA all states in Q are accepting.

Example 3.2.1 (SparseLU)**.** We give the BFA derived from the annotations in the SparseLU example (Listing 3.2). We associate indices to methods:

$$[0 : \mathit{constr}, 1 : \mathit{aP}, 2 : \mathit{compute}, 3 : \mathit{factorize}, 4 : \mathit{solve}]$$

The constructor annotations are implicit: it enables methods that are not guarded by annotations on other methods (in this case, *aP* and *compute*). The mapping $\mathcal{L}_{\text{SparseLU}}$ is as follows:

$$\begin{aligned}\mathcal{L}_{\text{SparseLU}} =\{&0 \mapsto \langle\{1,2\},\{3,4\},\emptyset\rangle,\ 1 \mapsto \langle\{3\},\{1,2,4\},\{1\}\rangle,\\ &2 \mapsto \langle\{4\},\{1,2,3\},\{2\}\rangle,\ 3 \mapsto \langle\{4\},\{1,2,3\},\{3\}\rangle,\ 4 \mapsto \langle\{1,2,3\},\emptyset,\{4\}\rangle\}\end{aligned}$$

The starting state is q_{1100}, as given by the annotations on the constructor. The set of states is

$$Q = \{q_{1100}, q_{0010}, q_{0001} q_{1111}\}$$

Finally, the transition function δ is given by following eight transitions:

$$\begin{array}{lll}\delta(q_{1100}, \mathit{aP}) = q_{0010} & \delta(q_{1100}, \mathit{compute}) = q_{0010} & \delta(q_{0010}, \mathit{factorize}) = q_{0001}\\ \delta(q_{0001}, \mathit{solve}) = q_{1111} & \delta(q_{1111}, \mathit{aP}) = q_{0010} & \delta(q_{1111}, \mathit{compute}) = q_{0001}\\ \delta(q_{1111}, \mathit{factorize}) = q_{0001} & \delta(q_{1111}, \mathit{solve}) = q_{1111} & \end{array}$$

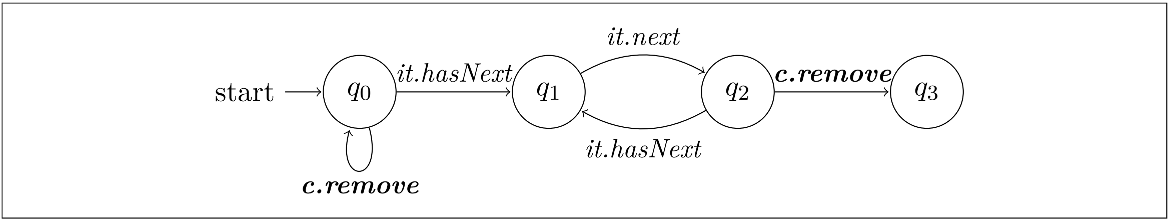


Figure 3.3: State diagram of Iterator DFA

Contrasting BFAs and DFAs We have already seen the differences between BFAs and DFAs in the specification of a representative concrete example. We now compare BFAs and DFAs more formally, by identifying a property that distinguishes the two models.

The property, called *context-independence*, is satisfied by all BFAs but not by all DFAs. To state the property and prove this claim, we need some convenient notations. First, we use $\widetilde{m}$ to denote a finite sequence of method names in Σ. Also, we use '$\cdot$' to denote sequence concatenation, defined as expected. Furthermore, given a BFA M, we write $L(M)$ to denote the language accepted by M, defined as $\{\widetilde{m} : \hat{\delta}(q_{E^c}, \widetilde{m}) = q' \wedge q' \in Q\}$, where $\hat{\delta}(q_b, \widetilde{m})$ is the extension of the one-step transition function $\delta(q_b, m_i)$ to a sequence $\widetilde{m}$ of method calls.

BFAs determine a strict sub-class of DFAs. First, because all states in Q are accepting states, BFA cannot encode the *"must call"* property (cf. Section 3.6). Next, we have the context-independence property:

Theorem 3.2.1 (Context-independence)**.** *Let $M = (Q, \Sigma_c^\bullet, \delta, q_{E^c}, \mathcal{L}_c)$ be a BFA. Then, for $m_n \in \Sigma_c^\bullet$ we have*

1. *If there is $\widetilde{p} \in L(M)$ and $m_{n+1} \in \Sigma_c^\bullet$ such that $\widetilde{p}\cdot m_{n+1} \notin L(M)$ and $\widetilde{p}\cdot m_n\cdot m_{n+1} \in L(M)$ then*
   there is no $\widetilde{m} \in L(M)$ such that $\widetilde{m} \cdot m_n \cdot m_{n+1} \notin L(M)$.
2. *If there is $\widetilde{p} \in L(M)$ and $m_{n+1} \in \Sigma_c^\bullet$ such that $\widetilde{p}\cdot m_{n+1} \in L(M)$ and $\widetilde{p}\cdot m_n\cdot m_{n+1} \notin L(M)$ then*
   there is no $\widetilde{m} \in L(M)$ such that $\widetilde{m} \cdot m_n \cdot m_{n+1} \in L(M)$.

Proof. We only consider the first item, as the second item is shown similarly. By $\widetilde{p} \cdot m_{n+1} \notin L(M)$ and $\widetilde{p} \cdot m_n \cdot m_{n+1} \in L(M)$ and Definition 3.2.5 we know that

$$m_{n+1} \in E_n \tag{3.1}$$

Further, for any $\widetilde{m} \in (\Sigma_c^\bullet)^*$, let q_b be such that $\delta(q_{10^{n-1}}, \widetilde{m}) = q_b$ and $q_{b'}$ such that $\delta(q_b, m_n) = q_{b'}$. Now, by Definition 3.2.5 we have that $\delta(q_{b'}, m_{n+1})$ is defined, as by (3.1) we know $P_{n+1} = \{m_{n+1}\} \subseteq b'$. Thus, for all $\widetilde{m} \in L(M)$ we have $\widetilde{m} \cdot m_n \cdot m_{n+1} \in L(M)$. This concludes the proof. □

Informally, the above theorem says that the effect of a call to m_n to subsequent calls (m_{n+1}) is not influenced by previous calls (i.e., the context) $\widetilde{m}$. That is, Item 1. (resp. Item 2.) says that method m_n enables (resp. disables) the same set of methods in any context.

The context-independence property is not satisfied by DFAs. Consider, for example, a DFA that disallows modifying a collection while iterating is not a BFA (as in Fig. 3 in [10]). Let *it* be a Java Iterator with its usual methods for a collection c. For the sake of illustration, we assume a single DFA relates the iterator and its collection methods; we give the associated state diagram in Figure 3.3. We can see that the sequence

'`c.remove;it.hasNext`'

```java
class Foo {
  SparseLU lu; Matrix a;
  void setupLU1(Matrix b) {
    this.lu.compute(this.a);
    if (?) this.lu.solve(b); }
  void setupLU2() {
    this.lu.analyzePattern(this.a);
    this.lu.factorize(this.a); }
  void solve(Matrix b) {
    this.lu.solve(b); } }
```

Listing (3.3) Class Foo using SparseLU

```java
void wrongUseFoo() {
  Foo foo; Matrix b;
  foo.setupLU1();
  foo.setupLU2();
  foo.solve(b);
}
```

Listing (3.4) Client code for Foo

is accepted, whereas the sequence

‘`it.hasNext;it.next;c.remove;it.hasNext`’

is not. That is, ‘`c.remove`’ disables ‘`it.hasNext`’ *only if* ‘`it.hasNext`’ has been previously called. Thus, the effect of calling ‘`c.remove`’ depends on the calls that precedes it.

BFA subtyping The combination of (i) locally-dependent annotations and (ii) the context-independence property they satisfy enable us to check for contract subtyping by *independently* comparing annotations method-wise; importantly, this comparison boils down to usual set inclusion. Suppose M_1 and M_2 are BFAs for classes c_1 and c_2, respectively, with c_1 being the super-class of c_2. The class inheritance imposes an important question: how to check that c_2 is a proper refinement of c_1. In other words, c_2 must subsume c_1: any valid sequence of calls to methods of c_1 must also be valid for c_2. Using BFAs, we can verify this simply by checking annotations method-wise. We can check whether M_2 subsumes M_1 only by considering their respective annotation mappings $\mathcal{L}_{c_2}$ and $\mathcal{L}_{c_1}$. Then, we have $M_2 \succeq M_1$ iff for all $m_j \in \mathcal{L}_c$ we have $E_1 \subseteq E_2$, $D_1 \supseteq D_2$, and $P_2 \subseteq P_1$ where $\langle E_i, D_i, P_i\rangle = \mathcal{L}_{c_i}(m_j)$ for $i \in \{1,2\}$.

3.3 A Compositional Analysis Algorithm

Since BFAs can be encoded as bit-vectors, standard data-flow analysis frameworks can be employed for the non-compositional case (e.g., intra-procedural analyses) [36]. Here we address the case of member object methods being called: we present a compositional algorithm that is tailored to the INFER compositional static analysis framework.

3.3.1 Key Ideas

We motivate our compositional analysis technique with the example below.

Example 3.3.1. Let `Foo` be a class that has a member `lu` of class `SparseLU` (cf. Listing 3.3). For each method of `Foo` that invokes methods on `lu` we compute a *symbolic summary* that denotes the effect of executing that method on typestates of `lu`. To check against client code, a summary gives us: (i) a pre-condition (i.e., which methods should be allowed before calling a procedure) and (ii) the effect on the *typestate* of an argument when returning from the procedure. A simple instance of a client is `wrongUseFoo` in Listing 3.4.

The central idea of our analysis is to accumulate enabling and disabling annotations. For this, the abstract domain maps object access paths to triples from the definition of $\mathcal{L}_{\text{SparseLU}}$ (cf. Definition 3.2.3). A *transfer function* interprets method calls in this abstract state. We illustrate the transfer function; the evolution of the abstract state is presented as comments in the following code listing.

```
1 void setupLU1(Matrix b) {
2   // s1 = this.lu -> ({}, {}, {})
3   this.lu.compute(this.a);
4   // s2 = this.lu -> ({solve}, {aP, factorize, compute}, {compute})
5   if (?) this.lu.solve(b); }
6   // s3 = this.lu -> ({solve, aP, factorize, compute}, {}, {compute})
7   // join s2 s3 = s4
8   // s4 = sum1 = this.lu -> ({solve}, {aP, factorize, compute}, {compute})
```

At the procedure entry (line 2) we initialize the abstract state as a triple with empty sets (s_1). Next, the abstract state is updated at the invocation of `compute` (line 3): we copy the corresponding tuple from $\mathcal{L}_{\text{SparseLU}}(compute)$ to obtain s_2 (line 4). Notice that `compute` is in the pre-condition set of s_2. Further, given the invocation of `solve` within the if-branch in line 5 we transfer s_2 to s_3 as follows: the enabling set of s_3 is the union of the enabling set from $\mathcal{L}_{\text{SparseLU}}(solve)$ and the enabling set of s_2 with the disabling set from $\mathcal{L}_{\text{SparseLU}}(solve)$ removed (i.e., an empty set in this case). Dually, the disabling set of s_3 is the union of the disabling set of $\mathcal{L}_{\text{SparseLU}}(solve)$ and the disabling set of s_1 with the enabling set of $\mathcal{L}_{\text{SparseLU}}(solve)$ removed. Here we do not have to add `solve` to the pre-condition set, as it is in the enabling set of s_2.

Finally, we *join* the abstract states of two branches (i.e., s_2 and s_3) at line 7. Intuitively, this join operates as follows: (i) a method is enabled only if it is enabled in both branches and not disabled in any branch; (ii) a method is disabled if it is disabled in either branch; (iii) a method called in either branch must be in the pre-condition (cf. Definition 3.3.1). Accordingly, in line 8 we obtain the final state s_4 which is also a summary for `SetupLU1`.

Now, we illustrate the checking of the client code `wrongUseFoo()` (cf. Listing 3.4), with computed summaries:

```
1 void wrongUseFoo() {
2   Foo foo; Matrix b;
3   // d1 = foo.lu -> ({aP, compute}, {solve, factorize}, {})
4   foo.setupLU1(); // apply sum1 to d1
5   // d2 = foo.lu -> ({solve}, {aP, factorize, compute}, {})
6   foo.setupLU2(); // apply sum2 = {this.lu -> ({solve}, {aP, factorize,
      compute}, {aP}) }
7   // warning! ‘analyzePattern’ is in pre of sum2, but not enabled in d2
8   foo.solve(b); }
```

Above, at line 2 the abstract state is initialized with annotations of the constructor `Foo`. Upon invocation of `setupLU1()` (line 4) we apply sum_1 in the same way as user-entered annotations are applied to transfer s_2 to s_3 above. Next, at line 6 we can see that `aP` is in the pre-condition set in the summary for `setupLU2()` (sum_2), which is computed similarly as sum_1; however, it is not in the enabling set of the current abstract state d_2. Thus, a warning is raised: `foo.lu` set up by `foo.setupLU1()` is never used and overridden by `foo.setupLU2()`.

Class Composition In the above example, the allowed orderings of method calls to an object of class `Foo` are imposed by the contracts of its object members (`SparseLU`) and the implementation of its methods. In practice, a class can have multiple members with their own BFA contracts. For instance, a class `Bar` can use two solvers, `SparseLU` and `SparseQR`:

```
1 class Bar {
2   SparseLU lu; SparseQR qr; /* ... */ }
```

where class `SparseQR` has its own BFA contract. The implicit contract of `Bar` depends on the contracts of `lu` and `qr`. Moreover, a class such as `Bar` can be a member of some other class. Thus, we refer to those classes as *composed* and to classes that have declared contracts (as `SparseLU`) as *base classes*.

3.3.2 The Algorithm

We formally define our analysis, which presupposes the control-flow graph (CFG) of a program. Let us write $\mathcal{AP}$ to denote the set of access paths, which enable a field-sensitive data-flow analysis; see, e.g., [9, 54, 46] for more information on this subject. Access paths model heap locations as paths used to access them: a program variable followed by a finite sequence of field accesses (e.g., $foo.a.b$). We use access paths as we would like to explicitly track states of class members; this, in turn, enables a precise compositional analysis. The abstract domain, denoted $\mathbb{D}$, maps access paths $\mathcal{AP}$ to BFA triples; below we write $Cod(\cdot)$ to denote the codomain of a mapping:

$$\mathbb{D} : \mathcal{AP} \to \bigcup_{c \in Classes} Cod(\mathcal{L}_c)$$

As the variables denoted by an access path in $\mathcal{AP}$ can be of any declared class $c \in Classes$, the co-domain of $\mathbb{D}$ is the union of codomains of $\mathcal{L}_c$ for all classes in a program. We remark that $\mathbb{D}$ is sufficient for both checking and summary computation, as we will show in the remaining of the section.

Definition 3.3.1 (Join Operator)**.** We define $\bigsqcup : Cod(\mathcal{L}_c) \times Cod(\mathcal{L}_c) \to Cod(\mathcal{L}_c)$ as follows:

$$\langle E_1, D_1, P_1 \rangle \sqcup \langle E_2, D_2, P_2 \rangle = \langle E_1 \ \cap \ E_2 \setminus (\ D_1 \cup D_2), \ D_1 \cup D_2, \ P_1 \cup \ P_2 \rangle$$

The join operator on $Cod(\mathcal{L}_c)$ is lifted to $\mathbb{D}$ by taking the union of un-matched entries in the mapping.

We now define some useful functions and predicates. First, we remark that our analysis is only concerned with three types of CFG nodes: a method call node, entry, and exit node of a method body; all other node types are irrelevant.

Notation 3.3.1. *We introduce convenient notations for entry and method call nodes:*

- ***`Entry-node`***$[m_j(p_0, \ldots, p_n)]$ *denotes a method entry node where* m_j *is a method name and* $p_0, \ldots, p_n$ *are formal arguments;*
- ***`Call-node`***$[m_j(p_0 : b_0, \ldots, p_n : b_n)]$ *denotes a call to method* m_j *where* $p_0, \ldots, p_n$ *are formal arguments and* $b_0, \ldots, b_n$ *are actual arguments.*

The following definitions concern CFG traversal, predecessor nodes, exit nodes, and actual parameters:

Definition 3.3.2 ($forward$(-))**.** Let G be a CFG. Then, $forward(G)$ enumerates nodes of G by traversing it in a breadth-first manner.

Definition 3.3.3 ($pred$(-))**.** Let G be a CFG and v a node of G. Then, $pred(v)$ denotes a set of predecessor nodes of v. That is, $pred(v) = W$ such that $w \in W$ if and only if there is an edge from w to v in G.

Definition 3.3.4 ($warning$(-))**.** Let G be a CFG and $\mathcal{L}_1, \ldots, \mathcal{L}_k$ be a collection of BFAs. We define $\mathsf{warning}(G, \mathcal{L}_1, \ldots, \mathcal{L}_k) = \textbf{true}$ if there is a path in G that violates some of $\mathcal{L}_i$ for $i \in \{1, \ldots, k\}$.

Definition 3.3.5 ($exit\_node$(-))**.** Let v be a method call node. Then, $\mathsf{exit\_node}(v)$ denotes the exit node w of a method body corresponding to v.

Definition 3.3.6 ($actual\_arg$(-, -))**.** Let $v =$ `Call-node`$[m_j(p_0 : b_0, \ldots, p_n : b_n)]$ be a call node. Suppose $p \in \mathcal{AP}$. We define $\mathsf{actual\_arg}(p, v) = b_i$ if $p = p_i$ for $i \in \{0, \ldots, n\}$; otherwise $\mathsf{actual\_arg}(p, v) = p$.

Algorithm 1: BFA Compositional Analysis

```
  Data: G : A program's CFG, a collection of BFA mappings: $\mathcal{L}_{c_1}, \ldots, \mathcal{L}_{c_k}$ over
        classes $c_1, \ldots c_k$ such that $well\_formed(\mathcal{L}_{c_i})$ for $i \in \{1, \ldots, k\}$
  Result: $warning(G, \mathcal{L}_{c_1}, \ldots, \mathcal{L}_{c_k})$
1 Initialize $NodeMap : Node \to \mathbb{D}$ as an empty map;
2 foreach $v$ in forward(G) do
3     if $pred(v)$ is empty then
4         Initialize $\sigma : \mathbb{D}$ as an empty mapping;
5     else
6         $\sigma = \bigsqcup_{w \in pred(v)} w$;
7     if guard(v, $\sigma$) then $NodeMap[v]$ := transfer($v$,$\sigma$); else return True;
8 return False
```

For convenience, we use a *dot notation* to access elements of BFA triples:

Definition 3.3.7 (Dot notation for BFA triples)**.** Let $\sigma \in \mathbb{D}$ and $p \in \mathcal{AP}$. Further, let $\sigma[p] = \langle E_\sigma, D_\sigma, P_\sigma \rangle$. Then, we have $\sigma[p].E = E_\sigma$, $\sigma[p].D = D_\sigma$, and $\sigma[p].P = P_\sigma$.

The compositional analysis is given in Algorithm 1. It expects a program's CFG and a series of contracts, expressed as BFAs annotation mappings (Definition 3.2.3). If the program violates the BFA contracts, a warning is raised. For the sake of clarity we only return a boolean indicating if a contract is violated (cf. Definition 3.3.4). In the actual implementation we provide more elaborate error reporting.

The algorithm traverses the CFG nodes top-down in a for-loop (lines 2–7) as given by $forward(G)$ (cf. Definition 3.3.2). For each node v, we first check whether v has predecessors: if not, when $\mathsf{pred}(v) = \emptyset$, we initialize domain σ as an empty mapping of type $\mathbb{D}$; otherwise, we collect information from its predecessors (as given by $\mathsf{pred}(v)$) and join them as σ (line 6). Then, it uses predicate guard(-,-) (cf. Algorithm 2) to check whether a method can be called in the given abstract state σ. If the pre-condition is met, then the transfer() function (cf. Algorithm 3) is called on a node. We assume a collection of BFA contracts (given as $\mathcal{L}_{c_1}, \ldots, \mathcal{L}_{c_k}$, the input for Algorithm 1) is accessible in Algorithm 3 to avoid explicit passing.

Guard Predicate Predicate $\mathsf{guard}(v, \sigma)$ checks whether a pre-condition for method call node v in the abstract state σ is met (cf. Algorithm 2). We represent a call node as $m_j(p_0 : b_0, \ldots, p_n : b_n)$ where p_i and b_i (for $i \in \{0, \ldots, n\}$) are formal and actual arguments, respectively. Let σ_w be a post-state of an exit node of method m_j. A pre-condition is satisfied if for all b_i there are no elements in their pre-condition set (i.e., the third element of $\sigma_w[b_i]$) that are also in disabling set of the current abstract state $\sigma[b_i]$. For this predicate we need the property $D = \Sigma_{c_i} \setminus E$, where Σ_{c_i} is a set of methods for class c_i.

This is ensured by condition $well\_formed(\mathcal{L}_{c_i})$ (Definition 3.2.4) and by the definition of transfer() (see below).

The Transfer Function The transfer function, given in Algorithm 3, distinguishes between two types of CFG nodes:

Entry-node: (lines 3–6) This is a function entry node. As described in Notation 3.3.1, for simplicity we represent it as $m_j(p_0, \ldots, p_n)$ where m_j is a method name and $p_0, \ldots, p_n$ are formal arguments. We assume p_0 is a reference to the receiver object (i.e., *this*). If method m_j is defined in a class c_i with user-supplied annotations $\mathcal{L}_{c_i}$, in line 5 we initialize the domain

Algorithm 2: Guard Predicate

```
Data: $v$ : CFG node, $\sigma$ : $\mathbb{D}$
Result: False iff $v$ is a method call that cannot be called in $\sigma$
Procedure guard $(v, \sigma)$
    switch $v$ do
        case Call-node$[m_j(p_0 : b_0, \dots, p_n : b_n)]$ do
            Let $w = exit\_node(v)$;
            for $i \in \{0, \dots, n\}$ do
                if $\sigma_w[p_i].P \cap \sigma[b_i].D \neq \emptyset$ then return False;
            return True
        otherwise do
            return True
```

Algorithm 3: Transfer Function

```
Data: $v$ : CFG node, $\sigma$ : $\mathbb{D}$
Result: Output abstract state $\sigma'$ : $\mathbb{D}$
1  Procedure transfer $(v, \sigma)$
2      switch $v$ do
3          case Entry-node$[m_j(p_0, \dots, p_n)]$ do
4              Let $c_i$ be the class of method $m_j(p_0, \dots, p_n)$;
5              if There is $\mathcal{L}_{c_i}$ then return $\{this \mapsto \mathcal{L}_{c_i}(m_j)\}$;
6              else return EmptyMap ;
7          case Call-node$[m_j(p_0 : b_0, \dots, p_n : b_n)]$ do
8              Let $\sigma_w$ be an abstract state of $exit\_node(v)$;
9              Initialize $\sigma' := \sigma$;
10             if this not in $\sigma'$ then
11                 for $ap$ in $dom(\sigma_w)$ do
12                     $ap' = actual\_arg(ap\{b_0/\texttt{this}\}, v)$;
13                     if $ap'$ in $dom(\sigma)$ then
14                         $E' = (\sigma[ap'].E \cup \sigma_w[ap].E) \setminus \sigma_w[ap].D$;
15                         $D' = (\sigma[ap'].D \cup \sigma_w[ap].D) \setminus \sigma_w[ap].E$;
16                         $P' = \sigma[ap'].P \cup (\sigma_w[ap].P \setminus \sigma[ap'].E)$;
17                         $\sigma'[ap'] = \langle E', D', P' \rangle$;
18                     else
19                         $\sigma'[ap'] := \sigma_w[ap]$;
20             return $\sigma'$
21         otherwise do
22             return $\sigma$
```

to the singleton map (i.e., *this* mapped to $\mathcal{L}_{c_i}(m_j)$). Otherwise, we return an empty map meaning that a summary has to be computed.

Call-node: (lines 7–20) We represent a call node as $m_j(p_0 : b_0, \dots, p_n : b_n)$ (cf. Notation 3.3.1) where we assume actual arguments $b_0, \dots, b_n$ are access paths for objects, with b_0 representing a receiver object.

The analysis is skipped if *this* is in the domain (line 10): this means the method has user-entered annotations. Otherwise, we transfer an abstract state for each argument b_i, but also for each *class member* whose state is updated by m_j. Thus, we consider all access paths in the domain of σ_w, that is $ap \in dom(\sigma_w)$ (line 11). We construct an access path ap' given ap. We distinguish two cases: ap denotes (i) a member and (ii) a formal argument of m_j. By line 12 we handle both cases. In the former case we know ap has form $this.c_1.\dots.c_n$. We construct ap' as ap with $this$ substituted for b_0 ($\mathsf{actual\_arg}(\cdot)$ is the identity in this case, see Definition 3.3.6): e.g., if receiver b_0 is $this.a$ and ap is $this.c_1.\dots.c_n$ then $ap' = this.a.c_1.\dots.c_n$. In the latter case ap denotes the formal argument p_i and $\mathsf{actual\_arg}(\cdot)$ returns corresponding actual argument b_i (as $p_i\{b_0/this\} = p_i$).

Now, as ap' is determined, we construct its BFA triple. If ap' is not in the domain of σ (line 13) we copy a corresponding BFA triple from σ_w (line 19). Otherwise, we transfer elements of an BFA triple at $\sigma[ap']$ as follows. The resulting enabling set is obtained by (i) adding methods that m_j enables ($\sigma_w[ap].E$) to the current enabling set $\sigma[ap'].E$, and (ii) removing methods that m_j disables ($\sigma_w[ap].D$), from it. The disabling set D' is constructed in a complementary way. Finally, the pre-condition set $\sigma[ap'].P$ is expanded with elements of $\sigma_w[ap].P$ that are not in the enabling set $\sigma[ap'].E$. We remark that the property $D = \Sigma_{c_i} \setminus E$ is preserved by the definition of E' and D'. Transfer is the identity on σ for all other types of CFG nodes.

We can see that for each method call we have a constant number of bit-vector operations per argument. That is, our BFA analysis is insensitive to the number of states, as a set of states is abstracted as a single set. Next, we discuss the efficiency of our compositional analysis algorithm by comparing it to the DFA-based approach.

Analysis Complexity: Comparison to DFA-based algorithm As already mentioned, the performance of a compositional DFA-based analysis depends on the number of states.

In DFA-based analyses, the analysis domain is given by $\mathcal{P}(Q)$, where Q is the set of states. In the intraprocedural analysis, at each method call, the transfer function would need to transition each state in the abstract state according to a given DFA. That is, the transfer function is the DFA's transition function lifted to a subset of states (with signature $\mathcal{P}(Q) \mapsto \mathcal{P}(Q)$). Clearly, the intraprocedural analysis depends linearly in the number of DFA states.

Even more prominently, the compositional interprocedural analysis is affected by the number of states. Each procedure has to be analyzed taking *each* state as an entry state: thus, effectively, we would need to run the intraprocedural analysis $|Q|$ times. Now, as a procedure body can contain branches, the analysis can result in *a set of states* for a given input state: the procedure summary is a mapping from a state into a set of states. For a procedure call, the transfer function would need to apply this mapping, thus taking $|Q|^2$ in the worst case. Overall, the compositional analysis takes $|Q|^3$ operations in the worst-case per a procedure call.

To sum up, taking BFAs as the basis for our analysis, an abstract domain is a set of bit-vectors; also, both transfer and join functions are bit-vector operations. The resulting intraprocedural analysis thus requires a *constant* number of operations per method invocation. More importantly, the compositional analysis also has a constant number of operations per method invocation. In fact, the bit-vector abstraction allows a uniform treatment of intraprocedural analysis and procedure summary computation. That is, our compositional analysis is insensitive to the number of states, which is in sharp contrast with DFA-based analyses.

Implementation Note, in our implementation we use several features specific to Infer: (1) Infer's summaries, which allow us to use a single domain for intra and inter procedural analysis; (2) scheduling on CFG top-down traversal, which simplifies the handling of branch statements. In principle, however, BFA can be implemented in other frameworks, such as, e.g., IFDS [52].

Correctness In a BFA, we can abstract a set of states by an *intersection* of states in the set. Let M be a BFA, and Q be its state set. Then, for $S \subseteq Q$ every method call sequence accepted by M *starting* in each state of S is also accepted starting in a state that is an intersection of bits of states in S. Theorem 3.3.1 formalizes this property. First we need an auxiliary definition:

Definition 3.3.8 ($\llbracket\text{-}\rrbracket(\text{-})$)**.** Let $\langle E, D, P\rangle \in Cod(\mathcal{L}_c)$ and $b \in \mathcal{B}^n$. We define $\llbracket\langle E, D, P\rangle\rrbracket(b) = b'$, where $b' = (b \cup E) \setminus D$ if $P \subseteq b$, and is undefined otherwise.

Theorem 3.3.1 (BFA $\cap$-Property)**.** *Suppose $M = (Q, \Sigma_c^\bullet, \delta, q_{E^c}, \mathcal{L}_c)$, $S \subseteq Q$, and $b_* = \bigcap_{q_b \in S} b$. Then we have:*

1. *For $m \in \Sigma_c^\bullet$, it holds: $\delta(q_b, m)$ is defined for all $q_b \in S$ iff $\delta(q_{b_*}, m)$ is defined.*
2. *Let $\sigma = \mathcal{L}_c(m)$. If $S' = \{\delta(q_b, m) : q_b \in S\}$ then $\bigcap_{q_b \in S'} b = \llbracket\sigma\rrbracket(b_*)$.*

Proof. We show two items:

1. By Definition 3.2.5, for all $q_b \in S$ we know $\delta(q_b, m)$ is defined when $P \subseteq b$ with $\langle E, P, D\rangle = \mathcal{L}_c(m)$. So, we have $P \subseteq \bigcap_{q_b \in P} b = b_*$ and $\delta(q_{b_*}, m)$ is defined.
2. By induction on $|S|$.
   - $|S| = 1$. Follows immediately as $\bigcap_{q_b \in \{q_b\}} q_b = q_b$.
   - $|S| > 1$. Let $S = S_0 \cup \{q_b\}$. Let $|S_0| = n$. By IH we know

$$\bigcap_{q_b \in S_0} \llbracket\sigma\rrbracket(b) = \llbracket\sigma\rrbracket(\bigcap_{q_b \in S_0} b) \tag{3.2}$$

We should show

$$\bigcap_{q_b \in (S_0 \cup \{q_{b'}\})} \llbracket\sigma\rrbracket(b) = \llbracket\sigma\rrbracket(\bigcap_{q_b \in (S_0 \cup \{q_{b'}\})} b)$$

We have

$$\begin{aligned}
\bigcap_{q_b \in (S_0 \cup \{q_{b'}\})} \llbracket\sigma\rrbracket(b) &= \bigcap_{q_b \in S_0} \llbracket\sigma\rrbracket(b) \cap \llbracket\sigma\rrbracket(b') \\
&= \llbracket\sigma\rrbracket(b_*) \cap \llbracket\sigma\rrbracket(b') && \text{(by (3.2))} \\
&= ((b_* \cup E) \setminus D) \cap ((b' \cup E) \setminus D) \\
&= ((b_* \cap b') \cup E) \setminus D && \text{(by set laws)} \\
&= \llbracket\sigma\rrbracket(b_* \ \cap \ b') = \llbracket\sigma\rrbracket(\bigcap_{q_b \in (S_0 \cup \{q_{b'}\})} b)
\end{aligned}$$

where $b_* = \llbracket\sigma\rrbracket(\bigcap_{q_b \in S_0} b)$. This concludes the proof.

□

Our BFA-based algorithm (Algorithm 1) interprets method call sequences in the abstract state and joins them (using the join operator from Definition 3.3.1) following the control-flow of the program. Thus, we can prove its correctness by separately establishing: (1) the correctness of the interpretation of call sequences using a *declarative* representation of the transfer function (Definition 3.3.9) and (2) the soundness of join operator (Definition 3.3.1). For brevity, we consider a single program object, as method call sequences for distinct objects are analyzed independently.

We define the *declarative* transfer function as follows:

Definition 3.3.9 ($\mathsf{dtransfer}_c$(-))**.** Let $c \in \mathit{Classes}$ be a class, Σ_c be a set of methods of c, and $\mathcal{L}_c$ be a BFA mapping. Furthermore, let $m \in \Sigma_c$ be a method, $\langle E^m, D^m, P^m \rangle = \mathcal{L}_c(m)$, and $\langle E, D, P \rangle \in \mathit{Cod}(\mathcal{L}_c)$. Then, we define

$$\mathsf{dtransfer}_c(m, \langle E, D, P \rangle) = \langle E', D', P' \rangle$$

where

- $E' = (E \ \cup \ E^m) \setminus D^m$,
- $D' = (D \ \cup \ D^m) \setminus E^m$, and
- $P' = P \ \cup \ (P^m \ \setminus \ E)$, if $P^m \cap D = \emptyset$, and is undefined otherwise.

Let $m_1, \ldots, m_n, m_{n+1}$ be a method sequence and $\phi = \langle E, D, P \rangle$, then

$$\mathsf{dtransfer}_c(m_1, \ldots, m_n, m_{n+1}, \phi) = \mathsf{dtransfer}_c(m_{n+1}, \mathsf{dtransfer}_c(m_1, \ldots, m_n, \phi))$$

Relying on Theorem 3.3.1, we state the soundness of join:

Theorem 3.3.2 (Soundness of $\sqcup$)**.** *Let $q_b \in Q$ and $\phi_i = \langle E_i, D_i, P_i \rangle$ for $i \in \{1, 2\}$. Then,*

$$[\![\phi_1]\!](b) \cap [\![\phi_2]\!](b) = [\![\phi_1 \sqcup \phi_2]\!](b)$$

Proof. By Definition 3.3.8, Definition 3.3.1, and set laws we have:

$$\begin{aligned}
[\![\phi_1]\!](b) \cap [\![\phi_2]\!](b) &= ((b \cup E_1) \setminus D_1) \cap ((b \cup E_2) \setminus D_2) \\
&= ((b \cup E_1) \cap (b \cup E_2)) \setminus (D_1 \cup D_2) \\
&= (b \cup (E_1 \cap E_2)) \setminus (D_1 \cup D_2) \\
&= (b \cup (E_1 \cap E_2 \setminus (D_1 \cup D_2)) \setminus (D_1 \cup D_2) \\
&= [\![\phi_1 \sqcup \phi_2]\!](b)
\end{aligned}$$

This concludes the proof. □

With these auxiliary notions in place, we show the correctness of the transfer function (i.e., summary computation that is specialized for the code checking):

Theorem 3.3.3 (Correctness of $\mathsf{dtransfer}_c(\cdot)$)**.** *Let $M = (Q, \Sigma_c, \delta, q_{E^c}, \mathcal{L}_c)$. Let $q_b \in Q$ and $\widetilde{m} = m_1, \ldots, m_n \in \Sigma_c^*$. Then*

$$\mathsf{dtransfer}_c(m_1, \ldots, m_n, \langle \emptyset, \emptyset, \emptyset \rangle) = \langle E', D', P' \rangle \iff \hat{\delta}(q_b, m_1, \ldots, m_n) = q_{b'}$$

such that $b' = [\![\langle E', D', P' \rangle]\!](b)$.

Proof. We show the two directions of the equivalence:

- ($\Rightarrow$, Soundness): By induction on the length of $\widetilde{m} = m_1, \ldots, m_n$.
  - Case $n = 1$. In this case we have $\widetilde{m} = m_1$. Let $\langle E^m, D^m, \{m_1\} \rangle = \mathcal{L}_c(m_1)$. By Definition 3.3.9 we have $E' = (\emptyset \cup E^m) \setminus D^m = E^m$ and $D' = (\emptyset \cup D^m) \setminus E^m = D^m$, as E^m and D^m are disjoint, and $P' = \emptyset \cup (\{m\} \setminus \emptyset)$. So, we have $b' = (b \cup E^m) \setminus D^m$. Further, we have $P' \subseteq b$. Finally, by the definition of $\delta(\cdot)$ (from Definition 3.2.5) we have $\hat{\delta}(q_b, m_1, \ldots, m_n) = q_{b'}$.

- Case $n > 1$. Let $\widetilde{m} = m_1, \ldots, m_n, m_{n+1}$. By IH we know

$$\mathsf{dtransfer}_c(m_1, \ldots, m_n, \langle \emptyset, \emptyset, \emptyset \rangle) = \langle E', D', P' \rangle \implies \hat{\delta}(q_b, m_1, \ldots, m_n) = q_{b'} \quad (3.3)$$

such that $b' = (b \cup E') \setminus D'$ and $P' \subseteq b$. Now, we assume $P'' \subseteq b$ and

$$\mathsf{dtransfer}_c(m_1, \ldots, m_n, m_{n+1}, \langle \emptyset, \emptyset, \emptyset \rangle) = \langle E'', D'', P'' \rangle$$

Then, we should show

$$\hat{\delta}(q_b, m_1, \ldots, m_n, m_{n+1}) = q_{b''} \quad (3.4)$$

where $b'' = (b \cup E'') \setminus D''$.

Let $\mathcal{L}_c(m_{n+1}) = \langle E^m, D^m, P^m \rangle$. We know $P^m = \{m_{n+1}\}$. By Definition 3.3.9 we have

$$\begin{aligned} &\mathsf{dtransfer}_c(m_1, \ldots, m_n, m_{n+1}, \langle \emptyset, \emptyset, \emptyset \rangle) = \\ &\quad \mathsf{dtransfer}_c(m_{n+1}, \mathsf{dtransfer}_c(m_1, \ldots, m_n, \langle \emptyset, \emptyset, \emptyset \rangle)\rangle) = \\ &\quad \mathsf{dtransfer}_c(m_{n+1}, \langle E', D', P' \rangle) \end{aligned}$$

Further, we have

$$E'' = (E' \cup E^m) \setminus D^m \qquad D'' = (D' \cup D^m) \setminus E^m \qquad P'' = P' \cup (P^m \setminus E') \quad (3.5)$$

Now, by substitution and De Morgan's laws we have:

$$\begin{aligned} b'' &= (b \cup E'') \setminus D'' \\ &= (b \cup ((E' \cup E^m) \setminus D^m)) \setminus ((D' \cup D^m) \setminus E^m) \\ &= ((b \cup (E' \cup E^m)) \setminus (D' \setminus E^m)) \setminus D^m \\ &= (((b \cup E') \setminus D') \cup E^m) \setminus D^m \\ &= (b' \cup E^m) \setminus D^m \end{aligned}$$

Now, by $P'' \subseteq b$, $P'' = P' \cup (P^m \setminus E')$, and $P^m \cap D' = \emptyset$, we have $P^m \subseteq (b \cup E') \setminus D' = b'$ (by (3.3)). Further, by Definition 3.2.5 we have

$$\delta(q_{b'}, m_{n+1}) = q_{b''} \quad (3.6)$$

Now, by the definition of $\hat{\delta}(\cdot)$ we have

$$\hat{\delta}(q_b, m_1, \ldots, m_{n+1}) = \delta(\hat{\delta}(q_b, m_1, \ldots, m_n), m_{n+1})$$

By this, (3.3), and (3.6) the goal (3.4) follows. This concludes this case.

- ($\Leftarrow$, Completeness): By induction on the length of $\widetilde{m} = m_1, \ldots, m_n$.
  - $n = 1$. In this case $\widetilde{m} = m_1$. Let $\langle E^m, D^m, \{m_1\} \rangle = \mathcal{L}_c(m_1)$. By Definition 3.2.5 we have $b' = (b \cup E^m) \setminus D^m$ and $\{m_1\} \subseteq b$. By Definition 3.3.9 we have $E' = E^m$, $D' = D^m$, and $P' = \{m_1\}$. Thus, as $\{m_1\} \cap \emptyset = \emptyset$ we have $b' = [\![\langle E', D', P' \rangle]\!](b)$.
  - $n > 1$. Let $\widetilde{m} = m_1, \ldots, m_n, m_{n+1}$. By IH we know

$$\hat{\delta}(q_b, m_1, \ldots, m_n) = q_b' \Rightarrow \mathsf{dtransfer}_c(m_1, \ldots, m_n, \langle \emptyset, \emptyset, \emptyset \rangle) = \langle E', D', P' \rangle \quad (3.7)$$

where $b' = (b \cup E') \setminus D'$ and $P' \subseteq b$. Now, we assume

$$\hat{\delta}(q_b, m_1, \dots, m_n, m_{n+1}) = q_{b''} \tag{3.8}$$

We should show that

$$\mathsf{dtransfer}_c(m_1, \dots, m_n, m_{n+1}, \langle \emptyset, \emptyset, \emptyset \rangle) = \langle E'', D'', P'' \rangle$$

such that $b'' = (b \cup E'') \setminus D''$ and $P'' \subseteq b$. We know

$$\mathsf{dtransfer}_c(m_1, \dots, m_n, m_{n+1}, \langle \emptyset, \emptyset, \emptyset \rangle) = \mathsf{dtransfer}_c(m_{n+1}, \langle E', D', P' \rangle)$$

By Definition 3.2.5 we have:

$$\hat{\delta}(q_b, m_1, \dots, m_n, m_{n+1}) = \delta(\hat{\delta}(q_b, m_1, \dots, m_n), m_{n+1}) = q_{b''}$$

So by (3.7) and (3.8) we have $\{m_{n+1}\} \subseteq b'$ and $b' = (b \cup E') \setminus D'$. It follows $\{m_{n+1}\} \cap D' = \emptyset$. That is, $\mathsf{dtransfer}_c(m_{n+1}, \langle E', D', P' \rangle)$ is defined. Finally, showing that $b'' = (b \cup E'') \setminus D''$ follows by the substitution and De Morgan's laws as in the previous case. This concludes the proof.

□

Let us discuss the specialization of Theorem 3.3.3 for the code checking. In this case, we know that a method sequence starts with the constructor method (i.e., the sequence is of the form $m^{\uparrow}, m_1, \dots, m_n$) and q_{E^c} is the input state. By *well_formed*$(\mathcal{L}_c)$ (Definition 3.2.4) we know that if $\delta(q_{E^c}, m^{\uparrow}) = q_b$ and

$$\mathsf{dtransfer}_c(m^{\uparrow}, m_1, \dots, m_n, \langle \emptyset, \emptyset, \emptyset \rangle) = \sigma$$

then methods not enabled in q_b are in the disabling set of σ. Thus, for any sequence $m_1, \dots, m_{k-1}, m_k$ such that m_k is disabled by the constructor and not enabled in substring $m_1, \dots, m_{k-1}$, the condition $P \cap D_i \neq \emptyset$ correctly checks that a method is disabled. If *well_formed*$(\mathcal{L}_c)$ did not hold, the algorithm would fail to detect an error as it would put m_k in P since $m_k \notin E$.

Aliasing We discuss how *aliasing information* can be integrated into our approach. In Example 3.3.1 member `lu` of object `foo` can be aliased. Thus, we keep track of BFA triples for all base members instead of constructing an explicit BFA contract for a composed class (e.g., `Foo`). Further, we would need to generalize an abstract state to a mapping of *alias sets* to BFA triples. That is, given a set of access paths $\{a_1, \dots, a_n\}$, the elements of the abstract state would be $\{a_1, \dots, a_n\} \mapsto \langle E, D, P \rangle$. For example, when invoking method `setupLU1` we would need to apply its summary (sum_1) to triples of each alias set that contains '`foo.lu`' as an element. Let $d_1 = \{S_1 \mapsto t_1, S_2 \mapsto t_2, \dots\}$ be an abstract state where S_1 and S_2 are the only keys such that `foo.lu` $\in S_i$ (for $i \in \{1, 2\}$) and t_1 and t_2 are some BFA triples.

```
// d1 = S1 -> t1, S2 -> t2, ...
foo.setupLU1(); // apply sum1 = {this.lu -> t3}
// d2 = S1 -> apply t3 to t1, S2 -> apply t3 to t2, ...
```

Above, at line 2 we would need to update the bindings of S_1 and S_2 by applying a BFA triple for `this.foo` from sum_1 (that is t_3) to t_1 and t_2. The resulting abstract state d_2 is given at line 3. We remark that if a procedure does not alter aliases, we can soundly compute and apply summaries, as shown above.

3.4 Analysis of 'Must call' Properties

Up to here, we have considered the specification of so-called *'may call'* properties—our BFA abstraction contains states that represent methods that *may* be called at some program point. It is natural to also consider *must call* properties, in which a method *requires* another method to be invoked in a code continuation. In this section, we show how the main ideas of our approach can be accommodated to support the analysis of contracts with 'must call' properties, by relying on a *conservative* extension of our BFA formalism.

Interestingly, we note that local contracts involving only 'must call' method dependencies also suffer from the state explosion problem. To illustrate this, consider a class that contains n pairs of methods such that one method requires another one to be invoked in a code continuation. Depending on the call history, at any given program point, any subset of n methods is required to be called in a code continuation. As this information must be encoded in states, the corresponding DFA would have 2^n reachable states.

Now we discuss how we refine our abstraction of states (set of states) in the presence of *require annotations*. In the case of enabling/disabling annotations, we showed that states only differ in a set of output edges. We leveraged this fact to abstract *a set of states* into a set of output edges. However, by having an additional 'require' annotations there could be two *distinct* states with the same set of output edges where incoming paths of one state can satisfy the 'require' annotation, whereas paths of the other state cannot. Furthermore, only states whose incoming paths satisfy all 'require' conditions can be accepting. Therefore, our abstraction of states must include information of required methods in addition to enabled methods. We remark that this refined abstraction still allow us to represent a set of states as a single state.

3.4.1 Annotation Language Extension

First, we extend the BFA specification language given in Section 3.2.1 with the following base annotation:

$$\texttt{@Require}(R_i)\ m_i$$

which asserts that invoking method m_i requires invocations of methods in R_i in a code continuation. In other words, a method call sequence starting with m_i is only valid if all methods in R_i are present in the sequence.

We extend the definition of annotation language from Definition 3.2.1 as follows:

Definition 3.4.1 (Annotation Language, Extended). Let $\Sigma_c = \{m^{\uparrow}, m_1, \ldots, m_n, m^{\downarrow}\}$ be a set of method names, where we have

- The constructor method $m^{\uparrow}$ is annotated by

  $$\texttt{@Enable}(E^c)\ \texttt{@Disable}(D^c)\ \texttt{@Require}(R^c)\ m^{\uparrow}$$

  where $E^c \cup D^c = \Sigma_c^{\bullet}$, $E^c \cap D^c = \emptyset$, and $R^c \subseteq E^c$;

- Each m_i for $m_i \in \Sigma_c^{\bullet}$ is annotated by

  $$\texttt{@Enable}(E_i)\ \texttt{@Disable}(D_i)\ \texttt{@Require}(R_i)\ m_i$$

  where $E_i \subseteq \Sigma_c^{\bullet}$, $D_i \subseteq \Sigma_c^{\bullet}$, $E_i \cap D_i = \emptyset$, and $R_i \subseteq E_i$.

Let $\tilde{x} = m^{\uparrow}, x_0, x_1, x_2, \ldots$ be a sequence where each $x_i \in \Sigma_c^{\bullet}$. We say that $\tilde{x}$ is *valid (w.r.t. annotations)* if the following holds:

```
class SparseLU {

    @EnableOnly(factorize)
    void analyzePattern(Mat a);

    @EnableOnly(solve)
    void factorize(Mat a);

    @EnableOnly(solve)
    void compute(Mat a);

    @EnableAll
    void solve(Mat b);  }
```

Listing (3.5) `SparseLU` BFA *May*-Contract

```
class SparseLU {

    @RequireOnly(factorize)
    void analyzePattern(Mat a);

    @RequireOnly(solve)
    void factorize(Mat a);

    @RequireOnly(solve)
    void compute(Mat a);

    @EnableAll
    void solve(Mat b);  }
```

Listing (3.6) `SparseLU` BFA* *Must*-Contract

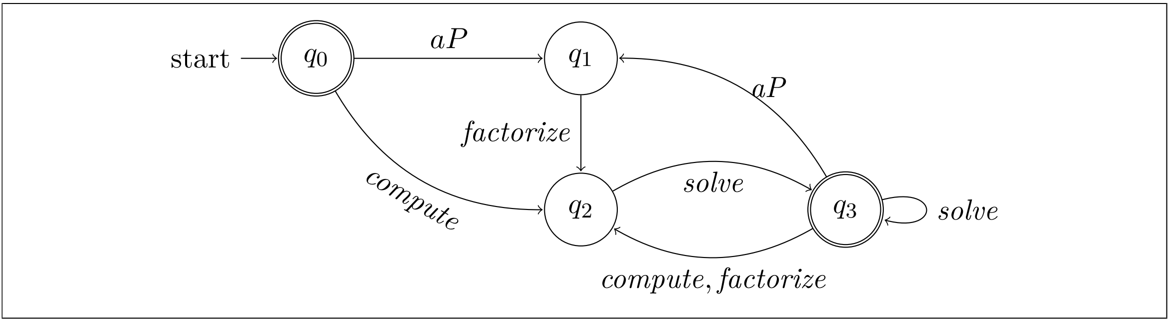


Figure 3.6: SparseLU BFA* with `Require` annotation

- For all substrings $\tilde{x}' = x_i, \dots, x_k$ of $\tilde{x}$ such that $x_k \in D_i$ there is j $(i < j \leq k)$ such that $x_k \in E_j$;
- If $\tilde{x}' = x_i, \dots$ is a substring of $\tilde{x}$ then for each $x_j \in R_i$ there is substring $x_i, \dots, x_j$ in $\tilde{x}'$.

Analogously to `@EnableOnly`(E_i) m_i we can derive `@RequiresOnly`(R_i) m_i as follows:

$$\texttt{@RequireOnly}(R_i)\ m_i \overset{\text{def}}{=} \texttt{@Require}(R_i)\ \texttt{@Disable}(C \setminus R_i)\ m_i$$

We illustrate the semantics of '`@Require`(R_i) m_i' by appealing to our running example from Figure 3.1. We may wish to refine the contract for class `SparseLU` in such a way that all computed resources *must* be used. For example, a call to method `compute()` has to be followed by at least one invocation of method `solve()`. The contract in Listing 3.6 makes use of '`@RequireOnly`(R_i) m_i' to enforce that all computed resources are properly consumed. Compare this 'must call' contract to its 'may call' counterpart in Listing 3.5: the only difference is that '`@EnableOnly`(E_i) m_i' are substituted by '`@RequireOnly`(R_i) m_i'.

Observe that the 'must call' contract induces an *extended* BFA (abbreviated BFA* in the following) in which *not all states are accepting* as they were in Figure 3.1. Such a BFA* is given in Figure 3.6: there, for instance, state q_2 is not an accepting state: calling `compute()` in q_1 cannot lead to an accepting state as it imposes a requirement to call `solve()`. Hence, in order to reach an accepting state from q_2 this requirement must be satisfied. In this case, a simple call to `solve()` in q_2 leads to the accepting state q_3.

Our insight is that every state q should record the accumulated requirements for its outgoing paths, i.e., methods that must be invoked to reach accepting states. For example, the abstraction of state q_2 should contain information that method `solve()` must be an element of a path to an accepting state. Therefore, only states without any such requirements are accepting states. As we have seen, we abstract a state by a bit-vector b, which records enabled methods in a state. Now, our abstraction of a state should also include another bit-vector f that records the accumulated requirements of a state. We now proceed to make these intuitions formal.

3.4.2 Formalizing the 'Must call' property

Extended BFA (BFA*)

Following the intuition that a state must record requirements for outgoing paths, we extend the state bit-vector representation as follows:

$$q_{b,f}$$

where $b, f \in \mathcal{B}^n$ with n being the number of methods in a class. Here, b represents the enabled methods in a state, as before, and f accumulates require annotations: methods that must be elements of *every* path from $q_{b,f}$ to some accepting state.

Now, we define $\mathcal{L}_c^*$ as the extension of the mapping $\mathcal{L}_c$ from Definition 3.2.3 as follows:

Definition 3.4.2 (Mapping $\mathcal{L}_c^*$)**.** Given a class c, we define $\mathcal{L}_c^*$ as a mapping from methods to tuple of subsets of Σ_c:

$$\mathcal{L}_c^* : \Sigma_c \to (\mathcal{P}(\Sigma_c) \times \mathcal{P}(\Sigma_c) \times \mathcal{P}(\Sigma_c)) \times (\mathcal{P}(\Sigma_c) \times \mathcal{P}(\Sigma_c))$$

Above, the first triple is as before: given $m_i \in \Sigma_c$ we write E_i, D_i, and P_i to denote first three elements of $\mathcal{L}_c^*(m_i)$. There is an additional pair in $\mathcal{L}_c^*(m_i)$, which collects information needed to encode the 'must call' property. We shall write R_i and C_i to denote its elements.

Similarly as before, transitions between states $q_{b,f}, q_{b',f'}, \cdots$ are determined by $\mathcal{L}_c^*$. In addition to the semantics of E_i, D_i, and P_i on transitions, we define the following meanings for R_i and C_i. The set of methods R_i adds the following requirements for subsequent transitions: given $m_i \in \Sigma_c^\bullet$ we have $l \in R_i$ if and only if m_l must be called after m_i. Dually, C_i records the fulfillment of requirements for a transition. Similarly to P_i, C_i is a singleton set containing method m_i. Again, we define this is as a set to ease the definition of the domain of the compositional analysis algorithm in Section 3.4.3. We formalize these intuitions as an extension of BFA (Definition 3.2.5).

Well-formed mapping We identify some natural well-formedness conditions on the mapping $\mathcal{L}_c^*$. First, we remark that a method cannot require a call to itself, as this would make a self-loop of requirements which cannot be satisfied by any finite sequence. Further, in order to be able to satisfy requirements (i.e. to reach accepting states) we need a condition that require annotations are subset of enabling annotations. We incorporate these conditions in the extension of predicate $well\_formed(\cdot)$ (Definition 3.2.4):

Definition 3.4.3 ($well\_formed(\mathcal{L}_c^*)$)**.** Let c, Σ_c, and $\mathcal{L}_c^*$ be a class, its method set, and its mapping, respectively. Then, $\mathsf{well\_formed}(\mathcal{L}_c^*) = \mathbf{true}$ iff the following conditions hold:

- $\mathcal{L}_c^*(m^\uparrow) = \langle E^c, D^c, \emptyset, R^c, \emptyset\rangle$ such that $E^c \cup D^c = \Sigma_c^\bullet$, $E^c \cap D^c = \emptyset$, and $R^c \subseteq E^c$;
- For $m_i \in \Sigma_c^\bullet$ we have $\mathcal{L}_c^*(m_i) = \langle E_i, D_i, \{m_i\}, R_i, \{m_i\}\rangle$ such that

$$E_i, D_i \subseteq \Sigma_c^\bullet,\ \ E_i \cap D_i = \emptyset,\ \ m_i \notin R_i,\ \text{and}\ R_i \subseteq E_i.$$

We are now ready to extend the definition of BFA from Definition 3.2.5:

Definition 3.4.4 (BFA*)**.** Given a $c \in \mathit{Classes}$ with $n > 0$ methods, an *extended BFA* (BFA*) for c is defined as a tuple $M = (Q, \Sigma_c, \delta, q_{E^c,R^c}, \mathcal{L}_c^*, F)$ where:

- Q is a finite set of states $q_{b,f}, q_{b',f'}, \ldots$, where $b, b', \ldots, f, f', \ldots \in \mathcal{B}^n$
- $\Sigma_c^\bullet = \{m_1, \ldots, m_n\}$ is the alphabet (method identities);
- q_{E^c,R^c} is the starting state;
- $\delta : Q \times \Sigma_c^\bullet \to Q$ is the transition function, where
$$\delta(q_{b,f}, m_i) = q_{b',f'}$$
with $b' = (b \cup E_i) \setminus D_i$ if $P_i \subseteq b$, and is undefined otherwise. Also, $f' = f \setminus C_i \cup R_i$;
- $\mathcal{L}_c^*$ is an extended BFA mapping (cf. Definition 3.4.2) such that $\mathit{well\_formed}(\mathcal{L}_c^*)$ (cf. Definition 3.4.3);
- The set of accepting states F is defined as
$$F = \{q_{b,0^n} : q_{b,0^n} \in Q\}$$

The definition of F follows the intuition that a state is accepting only if it has no outstanding requirements, i.e., its bit-vector f is zero-vector.

We now need to show that a well-formed $\mathcal{L}_c^*$ ensures that its induced BFA* has reachable accepting states. This boils down to showing that in each state the required bit set f is contained in the enabled bit set b:

Lemma 3.4.1. *Let $M = (Q, \Sigma_c, \delta, q_{E^c,R^c}, \mathcal{L}_c^*, F)$ be a BFA*. Then, for $q_{b,f} \in Q$ we have $f \subseteq b$.*

Proof. First, we can see that initial state q_{E^c,R^c} trivially satisfies $f \subseteq b$. Further, let $q_{b,f} \in Q$ such that $f \subseteq b$. Then, for $m_i \in \Sigma$ we have

$$\delta(q_{b,f}, m_i) = q_{b',f'}$$

with $b' = (b \cup E_i) \setminus D_i$ if $P_i \subseteq b$, and is undefined otherwise. Also, $f' = f \setminus C_i \cup R_i$. Now, the goal $f' \subseteq b'$ follows by this and conditions $E_i \cap D_i = \emptyset$ and $R_i \subseteq E_i$ ensured by well_formed($\mathcal{L}_c^*$) (Definition 3.4.3). □

We illustrate states and transitions of a BFA* given in Figure 3.6 in the following example:

Example 3.4.1 (SparseLU must-contract)**.** The mapping $\mathcal{L}_{\text{SparseLU}}^*$ that corresponds to the contract given in Listing 3.6 is as follows:

$$\begin{aligned}\mathcal{L}_{\text{SparseLU}}^* = &\{0 \mapsto \langle\{1,2\},\{3,4\},\emptyset\rangle,\langle\emptyset,\emptyset\rangle,\ 1 \mapsto \langle\{3\},\{1,2,4\},\{1\}\rangle,\langle\{3\},\{1\}\rangle,\\ &2 \mapsto \langle\{4\},\{1,2,3\},\{2\}\rangle,\langle\{4\},\{2\}\rangle,\ 3 \mapsto \langle\{4\},\{1,2,3\},\{3\}\rangle,\langle\{4\},\{3\}\rangle,\\ &4 \mapsto \langle\{1,2,3\},\emptyset,\{4\}\rangle,\langle\emptyset,\{4\}\rangle\}\end{aligned}$$

The starting state is $q_{1100,0000}$. The set of states is:

$$Q = \{q_{1100,00000}, q_{0010,00010}, q_{0001,00001}, q_{1111,0000}\}$$

Differently from the contract given in Example 3.2.1, in which all states were accepting, here we have an explicit set of accepting states:

$$F = \{q_{1100,0000}, q_{1111,0000}\}.$$

The corresponding transition function $\delta(\cdot)$ is as follows:

```
class Bar {
    SparseLU lu; Matrix a, b;
    void solveLU_must() {
        this.lu.solve(this.b); }
    void setupLU_must() {
        if (?) {
            this.lu.analyzePattern();
            this.lu.factorize();
        } else {
            this.lu.compute(); } } }
```

Listing (3.7) Class Bar using SparseLU

```
void useBar() {
    Bar bar;
    bar.setupLU_must();
    bar.solveLU_must();
}
```

Listing (3.8) Client code for Bar

$$\begin{array}{ll} \delta(q_{1100,0000}, aP) = q_{0010,0010} & \delta(q_{1100,0000}, compute) = q_{0010,0010} \\ \delta(q_{0010,0010}, factorize) = q_{0001,0001} & \delta(q_{0001,0001}, solve) = q_{1111,0000} \\ \delta(q_{1111,0000}, aP) = q_{0010,0010} & \delta(q_{1111,0000}, compute) = q_{0001,0001} \\ \delta(q_{1111,0000}, factorize) = q_{0001,0001} & \delta(q_{1111,0000}, solve) = q_{1111,0000} \end{array}$$

Notice that the transformations of b-bits of states are as in Example 3.2.1. Additionally, transitions operate on f-bits to determine the accepting states. For example, the transition

$$\delta(q_{1111,0000}, compute) = q_{0001,0001}$$

adds the requirement to call `solve` by f-bits 0001. This is satisfied in transition $\delta(q_{0001,0001}, solve) = q_{1111,0000}$. As the f-bits of $q_{1111,0000}$ are all zeros, this state is accepting.

BFA* subtyping We now discuss the extension of the subtyping relation given in Section 3.2.2. In order to check that c_1 is a superclass of c_2, that is that M_2 subsumes M_1 ($M_2 \succeq M_1$), additionally to checking respective E, D, and P sets of $\mathcal{L}^*_{c_1}$ and $\mathcal{L}^*_{c_2}$ for each method, as given in Section 3.2.2, we need the following checks: $R_2 \subseteq R_1$ and $C_1 \subseteq C_2$. This follows the intuition that a superclass must be at least permissive as its subclasses: the subclass methods can only have more requirements.

3.4.3 An Extended Algorithm

We now present the extension of the compositional analysis algorithm to account for BFAs*. We illustrate the key ideas of the required extensions with an example.

Example 3.4.2. In Listing 3.7 we give class `Bar` that has a member `lu` of `SparseLU` and implements two methods that make calls on `lu`; Listing 3.8 contains a client code for class `Bar`. Now we illustrate how a summary is computed in the presence of a 'require' annotation for `setupLU_must()` and `solveLU_must()`.

Analogously to how our original algorithm accumulates enabling annotations by traversing a program's CFG, in the extension we will accumulate require annotations. We extend the abstract domain with a pair $\langle R, C\rangle$, where R and C are sets of methods in which we will appropriately accumulate require annotations. Intuitively, we use R to record call requirements for a code continuation and C to track methods that have been called up to a current code point.

First, we compute a summary for `solveLU_must()` as follows:

```
void solveLU_must() {
    // s1 = this.lu -> ({}, {})
    this.lu.solve();
    // s2 = this.lu -> ({}, {solve})
    // sum_solveLU = s2
}
```

At procedure entry we initialize the abstract state as an empty pair (s_1). Next, on the invocation of `solve()` we simply copy the corresponding annotations from $\mathcal{L}^*_{SparseLU}(solve)$. Therefore, the summary `sum_solveLU` essentially only records that `solve` is called within this procedure.

Next, we compute a summary for `setupLU_must`:

```
1  void setupLU_must(Matrix b) {
2      // s1 = this.lu -> ({}, {})
3      if (?) {
4          this.lu.analyzePattern();
5          // s2 = this.lu -> ({factorize}, {})
6          this.lu.factorize();
7          // s3 = this.lu -> ({solve}, {})
8      } else {
9          this.lu.compute();
10         // s4 = this.lu -> ({solve}, {})
11     }
12     // join s3 s4 = s5
13     // s5 = this.lu -> ({solve}, {})
14     // sum_setupLU = s5
15 }
```

In the first if-branch, on line 5, we copy the corresponding annotations from $\mathcal{L}^*_{SparseLU}(aP)$ to obtain s_2. Here, we remark that `factorize` is in the require set of s_2. Next, on line 6 on the invocation of `factorize()` we remove `factorize` from the require set of s_2 and add its requirements, i.e., `solve` to the require set of s_2 to obtain s_3. Similarly, we construct s_4 on line 9.

Now, on line 12 we should join the resulting sets of the two branches, that is, s_3 and s_4. For this we take the union of the require sets and the intersection of the called sets: this follows the intuition that a method must be called in a continuation if it is required within *any* branch; dually, a method call required prior to branching is satisfied only if it is invoked in *both* branches.

Once summaries for `solveLU_must()` and `setupLU_must()` are computed, we can check the client code `useBar`:

```
1  void useBar() {
2      Bar bar;
3      // b1 = bar -> ({}, {})
4      bar.setupLU_must();
5      // copy sum_setupL1 to get b2
6      // b2 = bar -> ({solve}, {})
7      bar.solveLU_must();
8      // apply sum_solveLU to b2 to get b3
9      // b3 = bar -> ({}, {})
10     bar.destructor(); // explicit call to a destructor
11     // bar can be destructed here as there are no requirements for it
12     // that is, b3[bar].R is the empty set
13 }
```

Here, on line 4 we simply copy the summary computed for method `setupLU_must()`. Next, on line 7 we apply the summary of `solveLU_must()` to the current abstract state b_1 to obtain b_2: the resulting require set of b_3 is obtained by taking an union of the current require set and the summary's require set (the first component of `sum_solveLU`) and by removing elements of the called set (the second component of `sum_solveLU`) from it. The resulting called set is the union of the current called set and the called set of the summary. Finally, when `destructor` method is called (line 10) we check if there are any outstanding requirements for object `bar`: i.e. if a require set of the current abstract state is empty. As the require set in b_3 is empty, there no warning is raised.

We show to extend our compositional analysis algorithm from Section 3.3 to incorporate analysis of 'must call' properties.

Abstract Domain First we recall that our abstract domain $\mathbb{D}$ is a mapping from access paths to elements of mapping $\mathcal{L}_c$. Given the extended mapping $\mathcal{L}_c^*$, this is reflected on the abstract domain as follows:

$$\mathbb{D} : \mathcal{AP} \rightarrow \bigcup_{c \in Classes} Cod(\mathcal{L}_c^*)$$

The elements of the co-domain have now the following form:

$$\langle \langle E, D, P \rangle, \langle R, C \rangle \rangle$$

where $R, C \subseteq \Sigma_c$. Intuitively, M is a set of methods that must be called in a code continuation, and C is a set of methods that have been called up to the current program point.

Algorithm We modify the algorithm to work with an abstract domain extended with the pair $\langle R, C \rangle$. To this end, we extend (i) the join operator, (ii) the guard predicate (Algorithm 2), and (iii) the transfer function (Algorithm 3). Next, we discuss these extensions.

Join operator The modified join operator has the following signature:

$$\bigsqcup : Cod(\mathcal{L}_c^*) \times Cod(\mathcal{L}_c^*) \rightarrow Cod(\mathcal{L}_c^*)$$

Its definition is conservatively extended as follows:

$$\begin{aligned}&\langle \langle E_1, D_1, P_1 \rangle, \langle R_1, C_1 \rangle \rangle \sqcup \langle \langle E_2, D_2, P_2 \rangle, \langle R_2, C_2 \rangle \rangle = \\ &\quad \langle \langle E_1 \ \cap \ E_2 \setminus (\ D_1 \cup D_2), \ D_1 \cup D_2, \ P_1 \cup \ P_2 \rangle, \ \langle R_1 \cup R_2, \ C_1 \cap C_2 \rangle \rangle\end{aligned}$$

Guard predicate In Algorithm 2, in the body of case Call-node$[m_j(p_0 : b_0, \ldots, p_n : b_n)]$ we add the following check after line 4:

$$\textbf{if } m_j == destructor \textbf{ and } \sigma_w[p_0].R \neq \emptyset \textbf{ then return } \boldsymbol{False};$$

In the case m_j is destructor, we additionally check whether its requirements are empty; if not we raise a warning.

Transfer function In Algorithm 3 we add the following lines after line 16 to transfer the new elements $\langle R, C \rangle$:

$$\begin{aligned}R' &= (\sigma(ap).R \cup \sigma_w(ap).R) \setminus \sigma_w(ap).C \\ C' &= (\sigma(ap).C \cup \sigma_w(ap).C) \setminus \sigma_w(ap).R\end{aligned}$$

Then, the output abstract state σ' is constructed as follows:

$$\sigma'(ap') = \langle \langle E', D', P' \rangle, \langle R', C' \rangle \rangle$$

where E', D', and P' are constructed as in Algorithm 3.

3.4.4 Extended Proofs of Correctness

Here we present the correctness guarantees for BFAs*. We describe the needed extensions to the definitions, theorems, and proofs we discussed in the case of BFA. As we will see, all the correctness properties that hold for 'may call' contracts, hold for 'must call' contracts as well. Hence, we confirm that the main ideas of our bit-vector abstraction of DFAs are not limited to 'may call' properties that we initially focused on: the principles of our abstraction can be applied to 'must call' properties too.

Context-independence Here, we characterize context-independence property for require annotations. Recall that context-independence states that the effects of annotations on subsequent calls do not depend on previous calls. Similarly, in the case of enabling/disabling annotations, this property directly follows from the idempotence of the operation on f-bits in the extended definition of $\delta(\cdot)$, that is, $f' = (f \setminus C_i) \cup R_i$. The effect of this operation is independent of bits in f, which are accumulated by preceding calls (i.e., they represent a context).

Now, we formalize the extension of the statement and proof. First, as not all states in a BFA* are accepting, the definition of $L(M)$ that denotes strings accepted by M is now as follows:

$$L(M) = \{\widetilde{m} : \hat{\delta}(q_{E^c,R^c}, \widetilde{m}) = q' \wedge q' \in F\}$$

Consequently, we need to reformulate statements of first two items to preserve their meanings, and add the item concerning require annotations. Thus, we extend theorem Theorem 3.2.1 as follows:

Theorem 3.4.2 (Context-independence, Extended)**.** *Let $M = (Q, \Sigma_c^\bullet, \delta, q_{E^c,R^c}, \mathcal{L}_c^*, F)$ be a BFA*. Then, for $m_n \in \Sigma_c^\bullet$ we have*

1. *If there is $\widetilde{p}_1 \in \Sigma_c^{\bullet *}$ and $m_{n+1} \in \Sigma_c^\bullet$ such that for any $\widetilde{s}_2 \in \Sigma_c^{\bullet *}$ we have $\widetilde{p}_1 \cdot m_{n+1} \cdot \widetilde{s}_2 \notin L(M)$ and there is $\widetilde{p}_2 \in \Sigma_c^{\bullet *}$ such that $\widetilde{p}_1 \cdot m_n \cdot m_{n+1} \cdot \widetilde{p}_2 \in L(M)$ then there is no $\widetilde{m} \in \Sigma_c^{\bullet *}$ such that $\widetilde{m} \cdot m_n \cdot m_{n+1} \cdot \widetilde{s}_2 \notin L(M)$ for all $\widetilde{s}_2 \in \Sigma_c^{\bullet *}$.*
2. *If there is $\widetilde{p}_1, \widetilde{p}_2 \in \Sigma_c^{\bullet *}$ and $m_{n+1} \in \Sigma_c^\bullet$ such that $\widetilde{p}_1 \cdot m_{n+1} \cdot \widetilde{p}_2 \in L(M)$ and $\widetilde{p} \cdot m_n \cdot m_{n+1} \cdot \widetilde{s}_2 \notin L(M)$ for all $\widetilde{s}_2 \in \Sigma_c^{\bullet *}$ then there is no $\widetilde{m}_1, \widetilde{m}_2 \in \Sigma_c^{\bullet *}$ such that $\widetilde{m}_1 \cdot m_n \cdot m_{n+1} \cdot \widetilde{m}_2 \in L(M)$.*
3. *If there are $\widetilde{p}_1, \widetilde{p}_2 \in L(M)$ and $m_n \in \Sigma_c^\bullet$ such that $\widetilde{p}_1 \cdot m_n \cdot \widetilde{p}_2 \in L(M)$ then there is no $\widetilde{m} \in L(M)$ with $\mathcal{L}_c^*(m).R = \emptyset$ for $m \in \widetilde{m}$ such that $\widetilde{m} \cdot m_n \cdot \widetilde{p}_2 \notin L(M)$.*

Proof. This property follows directly by the definition of transition function $\delta(\cdot)$ from the definition of BFAs* (Definition 3.4.4): that is, by the idempotence of b and f-bits transformation. More precisely, the effects of transformations $b' = (b \cup E_i) \setminus D_i$ (resp. $f' = (f \setminus C_i) \cup R_i$) do not depend on input bits b (resp. f).

The first two items are shown similarly as in the proof of Theorem 3.2.1. We remark that additional sequences are only introduced in order to properly use the definition of $L(M)$ for BFAs*.

Now we show item (3). We prove directly by the extended definition of the transition function $\delta(\cdot)$, that is by $f' = (f \setminus C_i) \cup R_i$. First, let $q_{b,f}$ be defined as follows

$$\delta(q_{E^c,R^c}, \widetilde{p}_1 \cdot m_n) = q_{b,f}$$

Further, by $\widetilde{p}_1 \cdot m_n \cdot \widetilde{p}_2 \in L(M)$ we have $\widetilde{p}_2 \supseteq f$, as $q_{b',0^n} \in F$ by the definition of F. By this we have $\widetilde{p}_2 \supseteq R_n$ as $R_n \subseteq f$. Finally, as $\mathcal{L}_c^*(m).R = \emptyset$ for $m \in \widetilde{m}$ we have

$$\delta(q_{E^c,R^c}, \widetilde{m} \cdot m_n) = q_{b',R_n}$$

Using this and $\widetilde{p}_2 \supseteq R_n$ we have $\widetilde{m} \cdot m_n \cdot \widetilde{p}_2 \in L(M)$. □

BFA $\cap$-Property We first extend $[\![\cdot]\!](\cdot)$ from Definition 3.3.8 to operate on both b-bits and f-bits:

Definition 3.4.5 ($[\![\cdot]\!](\cdot)$ Extended)**.** Let $\langle\langle E, D, P\rangle, \langle R, C\rangle\rangle \in Cod(\mathcal{L}_c^*)$, $b, f \in \mathcal{B}^n$. We define

$$[\![\langle\langle E, D, P\rangle, \langle R, C\rangle\rangle]\!](b, f) = b', f'$$

where $b' = (b \cup E) \setminus D$ if $P \subseteq b$, and is undefined otherwise; and $f' = (f \setminus C) \cup R$.

Now, to abstract the set of states of a BFA* we also need to handle f-bits of states. Complementary to b^*, we define f^* as *the union* of f-bits. Now, we extend Theorem 3.3.1 by incorporating f-bits in states and also item (3), which shows that a union of f-bits is the right way to abstract set of states P into a single state: intuitively, set of states P can be abstracted into the accepting state only if all states in P are accepting.

Theorem 3.4.3 (BFA* $\cap$-Property)**.** *Suppose $M = (Q, \Sigma_c^\bullet, \delta, q_{E^c,R^c}, \mathcal{L}_c^*, F)$, $S \subseteq Q$, $b_* = \bigcap_{q_b \in P} b$, and $f^* = \bigcup_{q_{b,f} \in P} f$. Then we have:*

1. *For $m \in \Sigma_c$, it holds: $\delta(q_{b,f}, m)$ is defined for all $q_{b,f} \in S$ iff $\delta(q_{b_*,f_*}, m)$ is defined.*
2. *Let $\sigma = \mathcal{L}_c^*(m)$. If $S' = \{\delta(q_{b,f}, m) : q_{b,f} \in S\}$ then $\bigcap_{q_{b,f} \in S'} b, \bigcup_{q_{b,f} \in S'} f = [\![\sigma]\!](b_*, f_*)$.*
3. *$S \subseteq F$ if and only if $f^* = 0^n$.*

Proof. The first item is only concerned with b-bits, thus it is shown as in Theorem 3.3.1.

Now, we discuss the proof for item (2). Here, we can separately prove the part for b-bits and for f-bits. The former proof is the same as in the corresponding case of Theorem 3.3.1. Moreover, the proof concerning f-bits follows the same lines as for b-bits (by induction on the cardinality of S and set laws): it again directly follows by the idempotence of transformation of f-bits (i.e., $f' = (f \setminus C_i) \cup R_i$); we remark that difference here is that we use the union (in the definition of f^* bits) instead of the intersection.

Finally, the proof of item (3) follows directly from the definition of accepting states F, that is $F = \{q_{b,0^n} : q_{b,0^n} \in Q\}$. Thus we know $S \subseteq F$ if and only if for all $q_{b,f} \in S$ we have $f = 0^n$. The right-hand side is equivalent to $f^* = 0^n$. □

Soundness of join operator We extend Theorem 3.3.2 with f-bits in the state representation, $\langle R_i, C_i\rangle$ in ϕ_i, and using the extended $[\![\cdot]\!](\cdot)$ from Definition 3.4.5. We note that this theorem again relies on Theorem 3.4.3: we abstract set of reachable states by the union of f-bits.

For convenience, we will use "projections" of $[\![\cdot]\!](\cdot)$ to b and f-bits. Let $\phi = \langle\langle E, D, P\rangle, \langle R, C\rangle\rangle$, then will use $[\![\phi]\!]_b(b) = b'$ and $[\![\phi]\!]_f(f) = f'$, where b' and f' are defined as in Definition 3.4.5.

Theorem 3.4.4 (Soundness of extended $\sqcup$)**.** *Let $q_{b,f} \in Q$ and $\phi_i = \langle\langle E_i, D_i, P_i\rangle, \langle R_i, C_i\rangle\rangle$ for $i \in \{1, 2\}$. Then, $[\![\phi_1]\!]_b(b) \cap [\![\phi_2]\!]_b(b) = [\![\phi_1 \sqcup \phi_2]\!]_b(b)$ and $[\![\phi_1]\!]_f(f) \cup [\![\phi_2]\!]_f(f) = [\![\phi_1 \sqcup \phi_2]\!]_f(f)$.*

Proof. The proof concerning b-bits is the same as in Theorem 3.3.2. Now, we show the part concerning f-bits, that is

$$[\![\phi_1]\!]_f(f) \cup [\![\phi_2]\!]_f(f) = [\![\phi_1 \sqcup \phi_2]\!]_f(f)$$

The proof follows by the extended definition of $[\![\cdot]\!](\cdot)$ from Definition 3.4.5 and set laws as follows:

$$\begin{aligned}[\![\phi_1]\!](f) \cup [\![\phi_2]\!](f) &= ((f \setminus C_1) \cup R_1) \cup ((f \setminus C_2) \cup R_2)\\ &= (f \setminus (C_1 \cap C_2)) \cup (R_1 \cup R_2) = [\![\phi_1 \sqcup \phi_2]\!](f)\end{aligned}$$

□

Correctness of $\mathsf{dtransfer}_c(\cdot)$ We extend $\mathsf{dtransfer}_c(\cdot)$ from Definition 3.4.6 to account for the extended transfer function as follows:

Definition 3.4.6 ($\mathsf{dtransfer}_c(\cdot)$)**.** Let $c \in \mathit{Classes}$ be a class, Σ_c be a set of methods of c, and $\mathcal{L}_c^*$ be a BFA^*. Further, let $m \in \Sigma_c$ be a method, $\langle E^m, D^m, P^m\rangle = \mathcal{L}_c^*(m)$, and $\langle E, D, P\rangle \in \mathit{Cod}(\mathcal{L}_c^*)$. Then,

$$\mathsf{dtransfer}_c(m, \langle\langle E, D, P\rangle,\ \langle R, C\rangle\rangle) = \langle\langle E', D', P'\rangle,\ \langle R', C'\rangle\rangle$$

where $E' = (E \cup E^m) \setminus D^m$, $D' = (D \cup D^m) \setminus E^m$, and $P' = P \cup (P^m \setminus E)$, if $P^m \cap D = \emptyset$, and is undefined otherwise. Also, $R' = (R \cup R^m) \setminus C^m$ and $C' = (C \cup C^m) \setminus R^m$.

Let $m_1, \ldots, m_n, m_{n+1}$ be a method sequence and $\phi = \langle\langle E, D, P\rangle, \langle R, C\rangle\rangle$, then

$$\mathsf{dtransfer}_c(m_1, \ldots, m_n, m_{n+1}, \phi) = \mathsf{dtransfer}_c(m_{n+1}, \mathsf{dtransfer}_c(m_1, \ldots, m_n, \phi))$$

We now extend Theorem 3.3.3 to show the correctness of the extended $\mathsf{dtransfer}_c(\cdot)$ as follows:

Theorem 3.4.5 (Correctness of $\mathsf{dtransfer}_c(\cdot)$)**.** *Let $M = (Q, \Sigma_c^\bullet, \delta, q_{E^c,R^c}, \mathcal{L}_c^*, F)$. Let $q_{b,f} \in Q$ and $m_1, \ldots, m_n \in \Sigma_c^{\bullet *}$. Then*

$$\mathsf{dtransfer}_c(m_1, \ldots, m_n, \langle\langle \emptyset, \emptyset, \emptyset\rangle,\ \langle \emptyset, \emptyset\rangle\rangle) = \phi' \iff \hat{\delta}(q_{b,f}, m_1, \ldots, m_n) = q_{b',f'}$$

such that $b', f' = [\![\phi']\!](b, f)$ where $\phi' = \langle\langle E', D', P'\rangle,\ \langle R', C'\rangle\rangle$.

Proof. The proof concerning b-bits is as in Theorem 3.3.3. Now, we will prove part concerning transformation of f-bits.

We show Soundness ($\Rightarrow$) direction, as the other direction is shown similarly. The proof is by induction. We need strengthen the induction hypothesis with the following invariant: $R' \cap C' = \emptyset$.

- Case $n = 1$. We have $\widetilde{m} = m_1$. Let $R^m = \mathcal{L}_c^*(m_1).R$ and $C^m = \mathcal{L}_c^*(m_1).C$. First by the definition of $\mathsf{dtransfer}_c(\cdot)$ we have $R' = (\emptyset \cup R^m) \setminus C^m = R^m$ and $C' = (\emptyset \cup C^m) \setminus R^m = C^m$. Thus, we have $f' = [\![\phi']\!]_f(f) = (f \setminus C^m) \cup R^m$. Thus, directly by the definition of $\delta(\cdot)$ we have $\delta(q_{b,f}, m_1) = q_{b',f'}$.

- Case $n > 1$. Let $\widetilde{m} = m_1, \ldots, m_n, m_{n+1}$. By IH we know

$$\mathsf{dtransfer}_c(m_1, \ldots, m_n, \langle\langle \emptyset, \emptyset, \emptyset\rangle,\ \langle \emptyset, \emptyset\rangle\rangle) = \phi' \Rightarrow \hat{\delta}(q_{b,f}, m_1, \ldots, m_n) = q_{b',f'} \tag{3.9}$$

  such that $b', f' = [\![\phi']\!](b, f)$ and $f' \subseteq b'$ where $\phi' = \langle\langle E', D', P'\rangle,\ \langle R', C'\rangle\rangle$. As we focus only on f' bits, we can infer $f' = (f \setminus C') \cup R'$. Now, we assume

$$\mathsf{dtransfer}_c(m_1, \ldots, m_n, m_{n+1}, \langle\langle \emptyset, \emptyset, \emptyset\rangle,\ \langle \emptyset, \emptyset\rangle\rangle) = \phi'' \tag{3.10}$$

  such that $\phi'' = \langle\langle E'', D'', P''\rangle,\ \langle R'', C''\rangle\rangle$. We should show

$$\hat{\delta}(q_{b,f}, m_1, \ldots, m_n, m_{n+1}) = q_{b'',f''} \tag{3.11}$$

  such that $f'' = (f \setminus C'') \cup R''$ and $f'' \subseteq b''$. Let $\mathcal{L}_c^*(m_{n+1}) = \langle R^m, C^m\rangle$. We know $C^m = \{m_{n+1}\}$. By Definition 3.3.9 we have

$$\mathsf{dtransfer}_c(m_1, \ldots, m_n, m_{n+1}, \langle\langle \emptyset, \emptyset, \emptyset\rangle,\ \langle \emptyset, \emptyset\rangle\rangle) = \mathsf{dtransfer}_c(m_{n+1}, \phi')$$

Further, by (3.9), (3.10) and Definition 3.3.9 we have:

$$R'' = (R' \cup R^m) \setminus C^m$$
$$C'' = (C' \cup C^m) \setminus R^m$$

Here we remark that the invariant $R'' \cap C'' = \emptyset$ holds as $R^m \cap C^m = \emptyset$ by $well\_formed(\mathcal{L}_c^*)$ (Definition 3.4.3). Now, by substitution and De Morgan's laws we have:

$$\begin{aligned} f'' &= (f \setminus C'') \cup R'' \\ &= (f \setminus (C' \cup C^m)) \cup ((R' \cup R^m) \setminus C^m) \\ &= (((f \setminus C') \cup R') \setminus C^m) \cup R^m \\ &= ((f' \setminus C^m) \cup R^m \end{aligned}$$

where third equivalence holds by invariant $R' \cap C' = \emptyset$ and $R^m \cap C^m = \emptyset$. Further, by the definition of $\delta(\cdot)$ (from Definition 3.2.5) we have $\delta(q_{b',f'}, m_{n+1}) = q_{b'',f''}$. This concludes this case.

□

Summing up, we presented BFAs*, the extension to BFAs that allow us to specify both 'may call' and 'must call' properties, while enabling the bit-vector representation of states and transitions in the underlying BFA*. The bit-vector abstraction provides noticeable scalability benefits in terms of both specification and the code analysis. Next, we present the usability and performance evaluations that substantiate the claim of the smaller annotation overhead, as well as theoretical discussions of the algorithm performance improvements over DFA-based techniques.

3.5 Evaluation

To evaluate our technique, we implement two analyses in INFER, namely BFA* and DFA, and use the default INFER typestate analysis TOPL as a baseline comparison. More in details:

1. BFA*: The INFER implementation of the analysis technique introduced in this chapter.
2. DFA: A lightweight, DFA-based typestate analyzer implemented in INFER. We translate BFA* annotations to a minimal DFA and perform the analysis.
3. TOPL: An industrial typestate analyzer, implemented in INFER [2].

We remark that TOPL is designed for high precision and not for low-latency environments. It uses PULSE, an INFER memory safety analysis, which provides it with alias information. We include it in our evaluation as a baseline state-of-the-art typestate analysis, i.e., an off-the-shelf industrial strength tool we could hypothetically use. We note our benchmarks do not require aliasing and in theory PULSE is not required.

Our evaluation aims to validate the following two claims:

Claim-I: Reduced annotation overhead. The BFA* contract annotation overheads are smaller in terms of atomic annotations (e.g., @Post(...), @Enable(...)) than both competing analyses.

Claim-II: Improved scalability on large code and contracts. Our analysis scales better than the competing analyzers for our use case on two dimensions, namely, caller code size and contract size.

We analyze a benchmark of 22 contracts that specify common patterns of locally dependent contract annotations for a class. Of these, 18 are *may* contracts and 4 are *must* contracts. We identified common patterns of locally dependent contracts, such as setter/getter example given in Figure 1.3, and generate variants of them (e.g., by varying annotations and number of methods) such that we have contract samples that are *almost* linearly distributed in the number of (DFA) states. This can be seen in Figure 3.10, which outlines key features of these contracts (such as number of methods and number of states). The annotations for BFA* are varied; from them, we generate minimal DFA representations in DFA annotation format and Topl annotation format. This allows us to clearly show how the performance of the analyzers under consideration is impacted by the increase of the state space.

Moreover, we self-generate 122 client programs that follow the compositional patterns we described in Example 3.3.1 (this kind of patterns are also considered in, e.g., [34]). The pattern defines a composed class, as the class `Bar` illustrated at the end of Section 3.3.1, that has an object member of classes that have declared contracts (recall that we refer to those as base classes). Each of the methods of the composed class invokes methods on its members. Thus, a compositional analysis computes procedure summaries of these methods; this way, it effectively infers a contract of the composed class based on those of its class members. We remark that a composed class can itself be a member of another composed class, as expected. This pattern depends on important parameters, namely, number of composed classes, lines of code (i.e., number of method invocations), if-branches, and loops. The self-generation that follows this pattern allows us to precisely vary those parameters and measure their impact on the analysis performance. Note, the code is such that it does not invoke the need for aliasing (as we do not support it yet in our BFA* implementation).

Experimental Setup We used an Intel(R) Core(TM) i9-9880H CPU at 2.3 GHz with 16GB of physical RAM running macOS 11.6 on the bare-metal. The experiments were conducted in isolation without virtualization so that runtime results are robust. All experiments shown here are run in single-thread for Infer 1.1.0 running with OCaml 4.11.1.

Our use case is to integrate static analyses in interactive IDEs e.g., Microsoft Visual Studio Code [56], so that code can be analyzed at coding time. For this reason, our use case requires low-latency execution of the static analysis. Our SLA is based on the RAIL user-centric performance model [3].

Usability Evaluation Figure 3.10 outlines the key features of the 22 contracts we considered, called CR-1 – CR-22. Among these, CR-12, CR-14, CR-17, and CR-22 are *must* contracts.

In Figure 3.8 and Figure 3.9 we detail CR-4 as an example. For each contract, we specify the number of methods, the number of DFA states the contract corresponds to, and number of atomic annotation terms in BFA*, DFA, and Topl. An atomic annotation term is a standalone annotation in the given annotation language. We can observe that as the contract sizes increase in number of states, the annotation overhead for DFA and Topl increase significantly. On the other hand, the annotation overhead for BFA* remain largely constant wrt. state increase and increases rather proportionally with the number of methods in a contract. Observe that for contracts on classes with 4 or more methods, a manual specification using DFA or Topl annotations becomes impractical. Overall, we validate Claim-I by the fact that BFA* requires less annotation overhead on all of the contracts, making contract specification more practical.

Performance Evaluation Recall that we distinguish between *base* and *composed* classes: the former have a user-entered contract, and the latter have contracts that are implicitly inferred based on those of their members (that could be either base or composed classes themselves).

```
class SparseLU {
    states q0, q1, q2, q3, q4;
    @Pre(q0) @Post(q1)
    @Pre(q3) @Post(q1)
    void analyzePattern(Mat a);
    @Pre(q1) @Post(q2)
    @Pre(q3) @Post(q2)
    void factorize(Mat a);
    @Pre(q0) @Post(q2)
    @Pre(q3) @Post(q2)
    void compute(Mat a);
    @Pre(q2) @Post(q3)
    @Pre(q3)
    @Pre(q4) @Post(q3)
    void solve(Mat b);
    @Pre(q2) @Post(q4)
    @Pre(q3) @Post(q4)
    void transpose();}
```

Listing (3.9) DFA specification of SparseLU CR4 contract

```
class SparseLU {
  SparseLU();

  @EnableOnly(factorize)
  void analyzePattern(Mat a);

  @EnableOnly(solve, transpose)
  void factorize(Mat a);

  @EnableOnly(solve, transpose)
  void compute(Mat a);

  @EnableAll
  void solve(Mat b);

  @Disable(transpose)
  void transpose(); }
```

Listing (3.10) BFA specification of SparseLU CR4 contract

```
property SparseLU
  prefix "SparseLU"
  start -> start: *
  start -> q0: SparseLU() => x := RetFoo
  q1 -> q2: analyzePattern(SparseLU, IgnoreRet) when SparseLU == x
  q3 -> q2: analyzePattern(SparseLU, IgnoreRet) when SparseLU == x
  q1 -> q2: factorize(SparseLU, IgnoreRet) when SparseLU == x
  q3 -> q2: factorize(SparseLU, IgnoreRet) when SparseLU == x
  q0 -> q2: compute(SparseLU, IgnoreRet) when SparseLU == x
  q3 -> q2: compute(SparseLU, IgnoreRet) when SparseLU == x
  q2 -> q3: solve(SparseLU, IgnoreRet) when SparseLU == x
  q2 -> q4: transpose(SparseLU, IgnoreRet) when SparseLU == x
  q4 -> q2: transpose(SparseLU, IgnoreRet) when SparseLU == x
  q2 -> error: analyzePattern(SparseLU, IgnoreRet) when SparseLU == x
  q3 -> error: analyzePattern(SparseLU, IgnoreRet) when SparseLU == x
  q4 -> error: analyzePattern(SparseLU, IgnoreRet) when SparseLU == x
  q0 -> error: factorize(SparseLU, IgnoreRet) when SparseLU == x
  q2 -> error: factorize(SparseLU, IgnoreRet) when SparseLU == x
  q4 -> error: factorize(SparseLU, IgnoreRet) when SparseLU == x
  q1 -> error: compute(SparseLU, IgnoreRet) when SparseLU == x
  q2 -> error: compute(SparseLU, IgnoreRet) when SparseLU == x
  q4 -> error: compute(SparseLU, IgnoreRet) when SparseLU == x
  q1 -> error: solve(SparseLU, IgnoreRet) when SparseLU == x
  q4 -> error: solve(SparseLU, IgnoreRet) when SparseLU == x
  q4 -> error: solve(SparseLU, IgnoreRet) when SparseLU == x
  q0 -> error: transpose(SparseLU, IgnoreRet) when SparseLU == x
  q1 -> error: transpose(SparseLU, IgnoreRet) when SparseLU == x
  q4 -> error: transpose(SparseLU, IgnoreRet) when SparseLU == x
```

Listing (3.11) TOPL specification of SparseLU CR4 contract

Figure 3.8: DFA, BFA*, and TOPL specifications of CR4 contract for SparseLU class .

The total number of base classes in a composed class and contract size (i.e., the number of states in a minimal DFA that is a translation of a BFA* contract) play the most significant roles in execution-time. In Figure 3.11 we present a comparison of analyzer execution-times (y-axis) with contract size (x-axis), where each line in the graph represents a different number of base classes composed in a given class (given in legends).

Comparing BFA* analysis against DFA analysis **Figure 3.11a** compares various class compositions (with contracts) specified in the legend, for client programs of 500-1K LoC. The DFA implementation sharply increases in execution-time as the number of states increases.

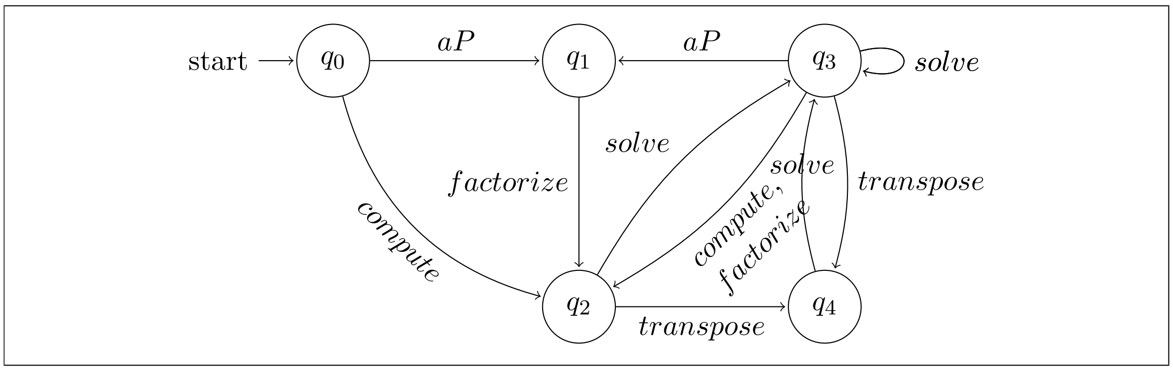


Figure 3.9: State diagram of DFA-based SparseLU CR-4 contract . This contract extends the `SparseLU` contract from Example 3.2.1 with an additional method (`transpose`). The intention is to capture the fact that consecutive calls to `transpose` are redundant. In Figure 3.8 we can see how this extension is specified in the three specification languages under consideration (DFA, TOPL, and BFA*).

Contract	#methods	#states	#BFA	#DFA	#TOPL
CR-1	3	2	3	5	9
CR-2	3	3	5	5	14
CR-3	3	5	4	8	25
CR-4	5	5	5	10	24
CR-5	5	9	8	29	71
CR-6	5	14	9	36	116
CR-7	7	18	12	85	213
CR-8	7	30	10	120	323
CR-9	7	41	12	157	460
CR-10	10	85	18	568	1407
CR-11	14	100	17	940	1884

Contract	#methods	#states	#BFA	#DFA	#TOPL
CR-12*	14	998	24	8185	20295
CR-13	14	1044	32	7766	20704
CR-14*	14	1358	24	10669	27265
CR-15	14	1628	21	13558	33740
CR-16	14	2104	25	17092	43305
CR-17*	14	2322	21	15529	47068
CR-18	14	2644	24	26014	61846
CR-19	16	3138	29	38345	88134
CR-20	18	3638	23	39423	91120
CR-21	18	4000	27	41092	101185
CR-22*	14	4510	27	36947	93615

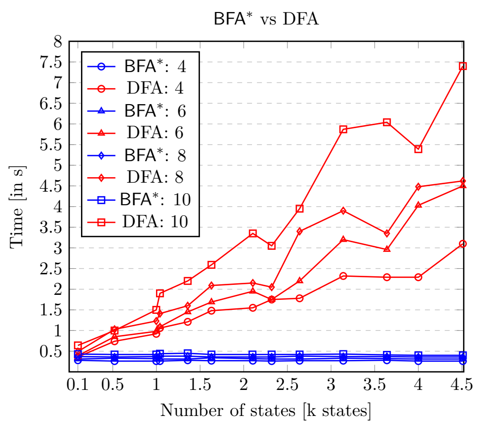
Figure 3.10: Details of the 22 contracts in our evaluation. Contracts marked with '*' include `Require` annotations.

The BFA* implementation remains rather constant, always under the SLA of 1 seconds. Overall, BFA* produces a geometric mean speedup over DFA of 5.7×.

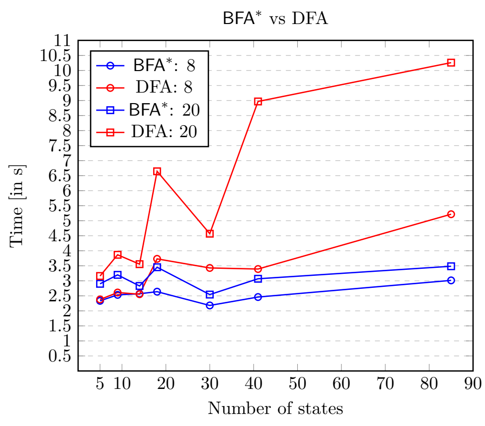
Figure 3.11b compares various class compositions for client programs of 15K LoC. Both implementations fail to meet the SLA; however, the BFA* is close and exhibits constant behaviour regardless of the number of states in the contract. The DFA implementation is rather erratic, tending to sharply increase in execution-time as the number of states increases. Overall, BFA* produces a geometric mean speedup over DFA of 1.5×. We note that *must* contracts do not have noticeable performance differences from *may* contracts.

Comparing BFA*-based analysis vs TOPL typestate implementations (Execution time) Here again client programs do not require aliasing. **Figure 3.11c** compares various class compositions for client programs of 500-1K LoC. The TOPL implementation sharply increases in execution-time as the number of states increases, quickly missing the SLA. In contrast, the BFA* implementation remains constant always under the SLA. Overall, BFA* produces a geometric mean speedup over TOPL of 6.59×. **Figure 3.11d** compares various class compositions for client programs of 15K LoC. Both implementations fail to meet the SLA. The TOPL implementation remains constant until ∼30 states and then rapidly increases in execution time. Overall, BFA* produces a geometric mean speedup over TOPL of 301.65×.

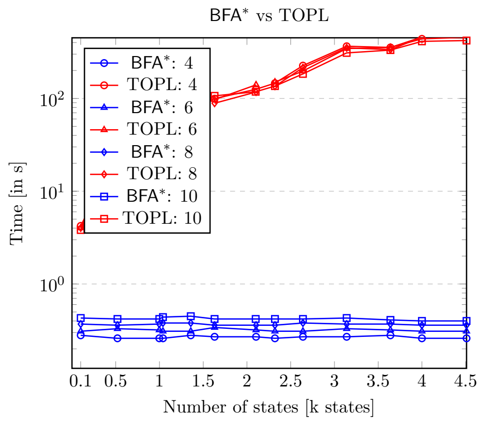
Overall, we validate Claim-II by showing that our technique removes state as a factor of performance degradation at the expense of limited but suffice contract expressively. Even when using client programs of 15K LoC, we remain close to our SLA and with potential



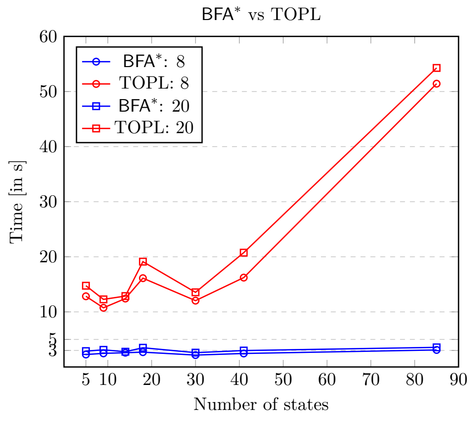
(a) DFA vs BFA* execution comparison on composed contracts (500-1k LoC)



(b) DFA vs BFA* execution comparison on composed contracts (15k LoC)



(c) TOPL vs BFA* comparison on composed contracts (500-1k LoC)



(d) TOPL vs BFA* comparison on composed contracts (15k LoC)

Figure 3.11: Runtime comparisons. Each line represents a different number of base classes composed in a client code.

to achieve it with further optimizations. Again, we note that *must* contracts do not have noticeable performance differences from *may* contracts.

3.6 Related Work

We focus on comparisons with restricted forms of typestate contracts. We refer to the typestate literature [55, 44, 19, 11, 18] for a more general treatment. The work [35] proposes restricted form of typestates tailored for use-case of the object construction using the builder pattern. This approach is restricted in that it only accumulates called methods in an abstract (monotonic) state, and it does not require aliasing for supported contracts. Compared to our approach, we share the idea of specifying typestate without explicitly mentioning states. On the other hand, their technique is less expressive than our annotations. They cannot express various properties we can (e.g., the property "cannot call a method"). Similarly, [23] defines heap-monotonic typestates where monotonicity can be seen as a restriction. It can be performed without an alias analysis.

Recent work on the RAPID analyzer [22] aims to verify cloud-based APIs usage. It combines *local* type-state with global value-flow analysis. Locality of type-state checking in their work is related to aliasing, not to type-state specification as in our work. Their type-state approach is DFA-based. They also highlight the state explosion problem for usual contracts found in practice, where the set of methods has to be invoked prior to some event. In comparison, we allow more granular contract specifications with a very large number of states while avoiding an explicit DFA. The FUGUE tool [18] allows DFA-based specifications, but also annotations for describing specific *resource protocols* contracts. These annotations have a *locality* flavor—annotations on one method do not refer to other methods. Moreover, we share the idea of specifying typestate without explicitly mentioning states. The annotations in FUGUE can specify “must call” properties (e.g. “must call a release method”). Our extended BFA formalism supports must logic that can express similar contracts. JaTyC [48] is a recent tool that supports Java inheritance. Our formalism can also handle inheritance, which we discuss in this chapter as BFA subsumption (cf. Section 3.2).

Our annotations could be mimicked by having a local DFA attached to each method. In this case, the DFAs would have the same restrictions as our annotation language. We are not aware of prior work in this direction. We also note that while our technique is implemented in INFER using the algorithm in Section 3.2, the fact that we can translate typestates to bit-vectors allows typestate analysis for local contracts to be used in distributive dataflow frameworks, such as IFDS [52].

3.7 Concluding Remarks

In this chapter, we have tackled the problem of analyzing code contracts in low-latency environments by developing a novel lightweight typestate analysis. Our technique is based on BFAs, a sub-class of contracts that can be encoded as bit-vectors. We believe BFAs are a simple and effective abstraction. They allow for succinct annotations that can describe a range of may and must logic contracts and on the other hand, they exhibit more scalable performance compared to DFA based approaches. We have implemented our typestate analysis in the industrial strength static analyzer INFER which is publicly available and open source.

Future Work There are several interesting research directions for the future work. First, it is worth investigating how BFA and DFA-based analyses can be bundled into a single analysis, thus inhering the benefits of both. Furthermore, we plan integrate aliasing in our approach, leveraging the fact that INFER already comes with aliasing checkers. This would enable an investigation to verify our conjecture that our BFA-based analysis performance gains will be preserved, or perhaps more prominently displayed, in the presence of aliasing information.

Moreover, it would be interesting to explore whether our BFA formalism can be effectively used in settings where BFA-based methods are typically used, such as, for example, automata learning, code synthesis, and automatic program repair. Finally, understanding the usability gains of moving from DFAs to BFAs is definitely interesting and it deserves a separate user study investigation.

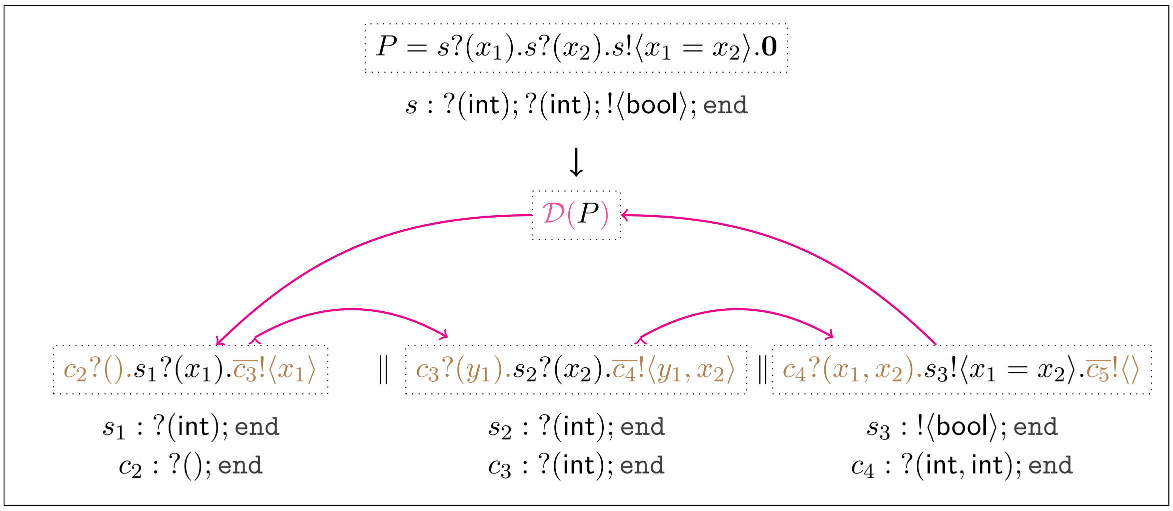
4 Minimal Session Types for HO

4.1 Introduction

This chapter identifies and studies the properties of an elementary formulation of session types, which we call *minimal session types*. Minimal session types are session types without sequencing. That is, in session types such as '$!\langle U\rangle;S$' and '$?(U);S$', we stipulate that S can only correspond to end, the type of the terminated protocol.

Adopting minimal session types entails dispensing with sequencing, which is arguably the most distinctive feature of session types. While this may appear as a far too drastic restriction, it turns out that it is not: we show that for every process P typable under standard (non minimal) session types, there is a *decomposition* of P, denoted $\mathcal{D}(P)$, a process that codifies the sequencing information given by the session types (protocols) of P using additional synchronizations, extracted from its protocols. Figure 4.1 illustrates the key idea of the decomposition using the process P and session type S motivated above. Because P contains three actions in sequence (as stipulated by S), its decomposition $\mathcal{D}(P)$ consists of three processes in parallel—each of them implementing one action of P—as well as of mechanisms for orchestrating these parallel processes: the synchronizations on names $c_2, \ldots, c_5$ ensure that the sequencing in P is preserved and that received names are properly propagated. These three parallel processes are typable with minimal session types (in the figure, they are given below each process), which are obtained by suitably "slicing" S.

Our main finding is that $\mathcal{D}(P)$ satisfies two important properties: first, it is well-typed using minimal session types (*static correctness*); second, it is behaviorally equivalent to P (*dynamic correctness*). These properties ensure that having sequencing in both types and processes is convenient but *redundant*: only sequencing at the level of processes is truly indispensable.



$P = s?(x_1).s?(x_2).s!\langle x_1 = x_2\rangle.\mathbf{0}$
$s : ?(\mathsf{int}); ?(\mathsf{int}); !\langle \mathsf{bool}\rangle; \mathtt{end}$
$\mathcal{D}(P)$
$c_2?().s_1?(x_1).\overline{c_3}!\langle x_1\rangle \;\|\; c_3?(y_1).s_2?(x_2).\overline{c_4}!\langle y_1, x_2\rangle \;\|\; c_4?(x_1, x_2).s_3!\langle x_1 = x_2\rangle.\overline{c_5}!\langle\rangle$
$s_1 : ?(\mathsf{int}); \mathtt{end}$
$c_2 : ?(); \mathtt{end}$
$s_2 : ?(\mathsf{int}); \mathtt{end}$
$c_3 : ?(\mathsf{int}); \mathtt{end}$
$s_3 : !\langle \mathsf{bool}\rangle; \mathtt{end}$
$c_4 : ?(\mathsf{int}, \mathsf{int}); \mathtt{end}$

Figure 4.1: The process decomposition, illustrated. Arrows in magenta indicate synchronizations orchestrated by the decomposition $\mathcal{D}(P)$.

The definition of $\mathcal{D}(P)$ is interesting on its own, as it draws inspiration from a known result by Parrow [50], who showed that any *untyped* π-calculus process can be decomposed as a collection of *trio processes*, i.e., processes with at most three nested prefixes [50].

The question of how to relate session types with other type systems has attracted interest in the past. Session types have been encoded into, for instance, generic types [27] and linear types [20, 15, 16]. As such, these prior studies concern the *relative expressiveness* of session types, where the expressivity of session types stands with respect to that of some other type system. In sharp contrast, we study the *absolute expressiveness* of session types: how session types can be explained in terms of themselves. To our knowledge, this is the first study of its kind.

Session types have been developed on top of different process languages (mainly, dialects of the π-calculus), and so choosing the target language for minimal session types is an important decision in our developments. In this chapter, our target language is HO, the core process calculus for session-based concurrency studied by Kouzapas et al. [39, 40]. HO is a very small language, which only supports passing of abstractions (i.e., functions from names to processes) and lacks name-passing and recursion. Nonetheless, HO is very expressive, because both features can be encoded in HO in a fully abstract way. Moreover, HO has a well-developed theory of behavioral equivalences [40]. The combination of minimal number of features and expressivity makes HO an excellent candidate for studying a minimal formulation of session types. Indeed, as we will see, several aspects of our decomposition take advantage of the higher-order nature of HO. Being a higher-order language, HO is very different from the (untyped, first-order) π-calculus considered by Parrow [50]. Therefore, our technical results arise in a context very different from Parrow's.

Contributions & Outline. In summary, in this chapter we present the following contributions:

1. We identify the class of *minimal session types* (MST) as a simple fragment of standard session types for HO without sequencing that retains its absolute expressiveness (Definition 4.3.1).

2. We show how to decompose standard session types into minimal session types, and how to decompose processes typable with standard session types into processes typable with minimal session types. This is a result of static correctness (Theorem 4.3.1).

3. We show that the decomposition of a process is behaviorally equivalent to the original process. This is a result of *dynamic correctness*, formalized in terms of *MST bisimulations*, a typed behavioral equivalence that we introduce here (Theorem 4.4.1).

4. We develop optimizations and extensions of our decomposition that bear witness to its robustness.

The rest of the chapter is organized as follows. In Section 4.3 we present *minimal session types*, and the decomposition of well-typed HO processes into minimal session types processes, accompanied by explanations and examples. In Section 4.4 we show the correctness of the decomposition, by establishing an *MST bisimulation* between an HO process and its decomposition. In Section 4.5 we examine two optimizations of the decomposition that are enabled by the higher-order nature of our setting. In Section 4.6 we discuss extensions of our approach to consider constructs for branching and selection. Finally, in Section 4.7 we elaborate further on related work and in Section 4.8 we present some closing remarks. The appendix contains omitted definitions and proofs.

Colors. Throughout the chapter we use different colors (such as pink and green) to improve readability. However, the usage of colors is not indispensable, and the chapter can be followed in black-and-white.

4.2 Examples

The following simple examples illustrate how HO can implement certain mechanisms that shall come in handy later, when defining the process decomposition in Section 4.3.

We first illustrate an HO implementation of a kind of recursive server that is able to repeatedly fulfil a certain request.

Example 4.2.1 (A Server of a Kind)**.** Let S_a denote the process $a?(x).(x\, r)$, which receives an abstraction on the shared name a and then applies it to r. Consider the following process composition:

$$\begin{aligned} P &= (\nu\, r)\,(\nu\, a)\,(a!\langle V\rangle.\mathbf{0} \mid a!\langle W\rangle.\mathbf{0} \mid S_a \mid \overline{r}?(x_1).\overline{r}?(x_2).Q) \\ V &= \lambda y.\,(y!\langle V'\rangle.S_a\{{}^{y}\!/r\}) \\ W &= \lambda z.\,(z!\langle W'\rangle.S_a\{{}^{z}\!/r\}) \end{aligned}$$

where V' and W' are some unspecified shared values. In P, process S_a operates as a server that provides r upon an invocation on a. Dually, the outputs on a are requests to this server. One possible reduction sequence for P is the following:

$$\begin{aligned} P &\longrightarrow (\nu\, r)\,(\nu\, a)\,(a!\langle W\rangle.\mathbf{0} \mid V\, r \mid \overline{r}?(x_1).\overline{r}?(x_2).Q) \\ &\longrightarrow (\nu\, r)\,(\nu\, a)\,(a!\langle W\rangle.\mathbf{0} \mid r!\langle V'\rangle.S_a \mid \overline{r}?(x_1).\overline{r}?(x_2).Q) \\ &\longrightarrow (\nu\, r)\,(\nu\, a)\,(a!\langle W\rangle.\mathbf{0} \mid S_a \mid \overline{r}?(x_2).Q\{{}^{V'}\!/x_1\}) = P' \end{aligned}$$

In this reduction sequence, the value V in the first request is instantiated with the name r by consuming a copy of S_a available in P. However, a copy of the server S_a is restored through the value V, after an communication on r. This way, in P' the exchange of W' on r can take place:

$$P' \longrightarrow^{*} (\nu\, r)\,(\nu\, a)\,(S_a \mid Q\{{}^{V'}\!/x_1\}\{{}^{W'}\!/x_2\})$$

Now, we consider an example that illustrate a mechanism for a form of the partial instantiation.

Example 4.2.2 (Partial Instantiation)**.** Let S_a and S_b be servers as defined in the previous example:

$$S_a = a?(x).(x\, r) \qquad\qquad S_b = b?(x).(x\, v)$$

Further, let R be a process in which requests to S_a and S_b are nested within abstractions:

$$R = a!\langle \lambda y.\, b!\langle \lambda z.\, V\,(y, z)\rangle\rangle$$

Notice how the polyadic application '$V\,(y, z)$' is enclosed in the innermost abstraction. Now consider the following composition:

$$P = (\nu\, a, b)\, R \mid S_a \mid S_b$$

The structure of R induces a form of partial instantiation for y, z, implemented by combining synchronizations and β-reductions. To see this, let us inspect one possible reduction chain for P:

$$\begin{aligned} P &\longrightarrow (\nu\, b)\;(\lambda y.\, b!\langle \lambda z.\, V\,(y, z)\rangle\, r)\;\mid S_b \\ &\longrightarrow (\nu\, b)\, b!\langle \lambda z.\, V\,(r, z)\rangle \mid S_b = P' \end{aligned}$$

The first request of R, aimed to obtain name r, is realized by the first reduction, i.e., the communication with S_a on name a: the result is the application of the top-level abstraction to r. Subsequently, the application step substitutes y with r. Hence, in P', names in the nested application are only *partially instantiated*: at this point, we have '$V\,(r,z)$'.

Process P' can then execute the same steps to instantiate z with name v by interacting with S_b. After two reductions, we obtain the fully instantiated application $V\,(r,v)$:

$$P' \longrightarrow \lambda z.\, V\,(r,z)\,v \longrightarrow V\,(r,v)$$

Next, we show how processes emerging from previous two examples are typed.

Example 4.2.3 (Typing Recursive Servers)**.** Here we show how to type the processes from Example 4.2.1. Let us define:

$$T = \mu \mathsf{t}.!\langle U\rangle;\mathsf{t} \qquad C = \langle T \multimap \diamond\rangle$$

where U is some value type. We recall process P from Example 4.2.1 with the additional typing information on bound names r and a:

$$\begin{aligned} P &= (\nu\, r : T)\,(\nu\, a : C)\,(a!\langle V\rangle.\mathbf{0} \mid a!\langle W\rangle.\mathbf{0} \mid S_a \mid \overline{r}?(x_1).\overline{r}?(x_2).Q)\\ V &= \lambda y.\,(y!\langle V'\rangle.S_a\{y/r\})\\ W &= \lambda z.\,(z!\langle W'\rangle.S_a\{z/r\}) \end{aligned}$$

where S_a stands for $a?(x).(x\,r)$. Let us assume there is a shared environment Γ under which V' and W' implement type U:

$$\Gamma;\emptyset;\emptyset \vdash V' \triangleright U \tag{4.1}$$
$$\Gamma;\emptyset;\emptyset \vdash W' \triangleright U \tag{4.2}$$

Also, we assume that process $\overline{r}?(x_1).\overline{r}?(x_2).Q$ is well-typed under the following environments:

$$\Gamma, a : C;\emptyset;\overline{r} : \overline{T} \vdash \overline{r}?(x_1).\overline{r}?(x_2).Q \triangleright \diamond \tag{4.3}$$

Under these assumptions, it holds that the body of process P correctly implements name a with type C, i.e.,

$$\Gamma;\emptyset;r : T \vdash (\nu\, a : C)\,(a!\langle V\rangle.\mathbf{0} \mid a!\langle W\rangle.\mathbf{0} \mid S_a) \triangleright \diamond$$

We detail the corresponding typing derivations:

$$\dfrac{\dfrac{}{\Gamma, a : C; x : T\multimap\diamond;\emptyset \vdash x \triangleright T\multimap\diamond}\,(\text{LVar}) \qquad \dfrac{}{\Gamma, a : C;\emptyset;y : T \vdash y \triangleright T}\,(\text{Sess})}{\Gamma, a : C; x : T\multimap\diamond; y : T \vdash x\,y \triangleright \diamond}\,(\text{App}) \tag{4.4}$$

$$\dfrac{(4.4) \qquad \dfrac{}{\Gamma, a : C;\emptyset;\emptyset \vdash a \triangleright \langle T\multimap\diamond\rangle}\,(\text{Sh}) \qquad \dfrac{}{\Gamma, a : C; x : T\multimap\diamond;\emptyset \vdash x \triangleright T\multimap\diamond}\,(\text{LVar})}{\Gamma, a : C;\emptyset; y : T \vdash a?(x).(x\,y) \triangleright \diamond}\,(\text{Acc}) \tag{4.5}$$

$$\dfrac{\dfrac{(4.5) \qquad (4.1)}{\Gamma, a : C;\emptyset; y : T \vdash y!\langle V'\rangle.S_a\{y/r\} \triangleright \diamond}\,(\text{Send}) \qquad \dfrac{}{\Gamma, a : C;\emptyset;y : T \vdash y \triangleright T}\,(\text{Sess})}{\Gamma, a : C;\emptyset;\emptyset \vdash V \triangleright T\multimap\diamond}\,(\text{Abs}) \tag{4.6}$$

$$\dfrac{\dfrac{}{\Gamma, a : C;\emptyset;\emptyset \vdash \mathbf{0} \triangleright \diamond}\,(\text{Nil}) \qquad \dfrac{}{\Gamma, a : C;\emptyset;\emptyset \vdash a \triangleright \langle T\multimap\diamond\rangle}\,(\text{Sh}) \qquad (4.6)}{\Gamma, a : C;\emptyset;\emptyset \vdash a!\langle V\rangle.\mathbf{0} \triangleright \diamond}\,(\text{Req}) \tag{4.7}$$

In the following derivation tree the right-hand side is shown similarly to (4.7) using assumption (4.2) instead of (4.1):

$$\textrm{(Par)}\ \dfrac{(4.7) \qquad \dfrac{}{\Gamma, a : C; \emptyset; \emptyset \vdash a!\langle W\rangle.\mathbf{0} \triangleright \diamond}}{\Gamma, a : C; \emptyset; \emptyset \vdash a!\langle V\rangle.\mathbf{0} \mid a!\langle W\rangle.\mathbf{0} \triangleright \diamond} \tag{4.8}$$

Finally we have:

$$\textrm{(ResS)}\ \dfrac{\textrm{(Res)}\ \dfrac{\textrm{(Par)}\ \dfrac{\textrm{(Par)}\ \dfrac{(4.8) \qquad \dfrac{}{\Gamma, a : C; \emptyset; r : T \vdash S_a \triangleright \diamond}}{\Gamma, a : C; \emptyset; r : T \vdash a!\langle V\rangle.\mathbf{0} \mid a!\langle W\rangle.\mathbf{0} \mid S_a \mid \overline{r}?(x_1).\overline{r}?(x_2).Q \triangleright \diamond} \qquad (4.3)}{\Gamma, a : C; \emptyset; r : T, \overline{r} : \overline{T} \vdash a!\langle V\rangle.\mathbf{0} \mid a!\langle W\rangle.\mathbf{0} \mid S_a \mid \overline{r}?(x_1).\overline{r}?(x_2).Q \triangleright \diamond}}{\Gamma; \emptyset; r : T, \overline{r} : \overline{T} \vdash (\nu\, a : C)\, (a!\langle V\rangle.\mathbf{0} \mid a!\langle W\rangle.\mathbf{0} \mid S_a \mid \overline{r}?(x_1).\overline{r}?(x_2).Q) \triangleright \diamond}}{\Gamma; \emptyset; \emptyset \vdash (\nu\, r : T)\, (\nu\, a : C)\, (a!\langle V\rangle.\mathbf{0} \mid a!\langle W\rangle.\mathbf{0} \mid S_a \mid \overline{r}?(x_1).\overline{r}?(x_2).Q) \triangleright \diamond}$$

Above, we have omitted details of the right-hand side derivation; it the same as (4.5) with name y substituted with r.

Example 4.2.4 (Typing Nested Abstractions)**.** Here we show how to type process P from Example 4.2.2. Let types C_1 and C_2 be defined as $C_i = \langle S_i \multimap \diamond\rangle$ where $i \in \{1, 2\}$ and S_i stands for a tail-recursive type. For simplicity, we assume that value V has the following typing:

$$a : C_1, b : C_2; \emptyset; \emptyset \vdash V \triangleright (S_1, S_2) \multimap \diamond \tag{4.9}$$

The following holds:

$$\emptyset; \emptyset; \emptyset \vdash (\nu\, a : C_1)\, (\nu\, b : C_2)\, R \mid S_a \mid S_b \triangleright \diamond \tag{4.10}$$

In the following typing derivations, we rely on the following two typing rules for polyadic elements; they can be derived from monadic typing rules from Figure 2.5 (see Remark 2 for details):

$$\textrm{(PolySess)}\ \dfrac{}{\Gamma; \emptyset; \widetilde{u} : \widetilde{S} \vdash \widetilde{u} \triangleright \widetilde{S}}$$

$$\textrm{(PolyApp)}\ \dfrac{\rightsquigarrow \in \{\multimap, \to\} \qquad \Gamma; \Lambda; \Delta_1 \vdash V \triangleright \widetilde{C} \rightsquigarrow \diamond \qquad \Gamma; \emptyset; \Delta_2 \vdash \widetilde{u} \triangleright \widetilde{C}}{\Gamma; \Lambda; \Delta_1, \Delta_2 \vdash V\, \widetilde{u}}$$

Now, we detail the typing derivations that show (4.10):

$$\textrm{(PolyApp)}\ \dfrac{(4.9) \qquad \textrm{(PolySess)}\ \dfrac{}{a : C_1, b : C_2; \emptyset; y : S_1, z : S_2 \vdash y, z \triangleright S_1, S_2}}{a : C_1, b : C_2; \emptyset; y : S_1, z : S_2 \vdash V\, (y, z) \triangleright \diamond} \tag{4.11}$$

$$\textrm{(Abs)}\ \dfrac{(4.11) \qquad \textrm{(Sess)}\ \dfrac{}{a : C_1, b : C_2; \emptyset; z : S_2 \vdash z \triangleright S_2}}{a : C_1, b : C_2; \emptyset; y : S_1 \vdash \lambda z.\, V\, (y, z) \triangleright S_2 \multimap \diamond} \tag{4.12}$$

$$\textrm{(Req)}\ \dfrac{\textrm{(Nil)}\ \dfrac{}{a : C_1, b : C_2; \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond} \qquad \textrm{(Sh)}\ \dfrac{}{a : C_1, b : C_2; \emptyset; \emptyset \vdash b \triangleright \langle S_2 \multimap \diamond\rangle} \qquad (4.12)}{a : C_1, b : C_2; \emptyset; y : S_1 \vdash b!\langle \lambda z.\, V\, (y, z)\rangle \triangleright \diamond} \tag{4.13}$$

$$\text{(Abs)}\ \dfrac{(4.13) \qquad \text{(Sess)}\ \dfrac{}{a : C_1, b : C_2; \emptyset; y : S_1 \vdash y \triangleright S_1}}{a : C_1, b : C_2; \emptyset; \emptyset \vdash \lambda y.\, b!\langle \lambda z.\, V\,(y, z)\rangle \triangleright S_1 \multimap \diamond} \tag{4.14}$$

$$\text{(Req)}\ \dfrac{a : C_1, b : C_2; \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond \qquad \text{(Sh)}\ \dfrac{}{a : C_1, b : C_2; \emptyset; \emptyset \vdash a \triangleright \langle S_1 \multimap \diamond\rangle} \qquad (4.14)}{a : C_1, b : C_2; \emptyset; \emptyset \vdash a!\langle \lambda y.\, b!\langle \lambda z.\, V\,(y, z)\rangle\rangle \triangleright \diamond} \tag{4.15}$$

Finally we have:

$$\text{(Res)}\ \dfrac{\text{(Res)}\ \dfrac{\text{(Par)}\ \dfrac{\text{(Par)}\ \dfrac{(4.15) \qquad \dfrac{}{a : C_1, b : C_2; \emptyset; \emptyset \vdash S_a \triangleright \diamond}}{a : C_1, b : C_2; \emptyset; \emptyset \vdash R \mid S_a \triangleright \diamond} \qquad \dfrac{}{a : C_1, b : C_2; \emptyset; \emptyset \vdash S_b \triangleright \diamond}}{a : C_1, b : C_2; \emptyset; \emptyset \vdash R \mid S_a \mid S_b \triangleright \diamond}}{a : C_1; \emptyset; \emptyset \vdash (\nu\, b : C_2)\, R \mid S_a \mid S_b \triangleright \diamond}}{\emptyset; \emptyset; \emptyset \vdash (\nu\, a : C_1)\, (\nu\, b : C_2)\, R \mid S_a \mid S_b \triangleright \diamond}$$

In the above typing derivation, we remark that the judgments

$$a : C_1, b : C_2; \emptyset; \emptyset \vdash S_a \triangleright \diamond \tag{4.16}$$
$$a : C_1, b : C_2; \emptyset; \emptyset \vdash S_b \triangleright \diamond \tag{4.17}$$

are shown similarly as in (4.5) from Example 4.2.3. Indeed, to derive (4.16) reusing the derivation tree from (4.5) we need to substitute y with r and then weaken the shared environment (4.5) with $b : C_2$ (see Lemma 2.2.2). Similarly, by substituting a with b and y with r in the derivation tree (4.5) and then by weakening the shared environment with $a : C_1$ in its conclusion we can obtain (4.17).

4.3 Decomposing Session-Typed Processes

In this section we define minimal session types and present a *decomposition* of well-typed processes: given a process P typable with (standard) session types, our decomposition returns a process denoted $\mathcal{D}(P)$, typable with minimal session types.

The definition of $\mathcal{D}(P)$ follows Parrow's *trio processes* for the π-calculus [50]. A trio process is a process with at most three sequential prefixes. Roughly speaking, if P is a process with k sequential actions, then $\mathcal{D}(P)$ will contain k trios running in parallel: each of them will enact exactly one action from P. The decomposition is carefully designed to ensure that trios trigger each other by preserving the sequencing in P.

This section is organized as follows. First, in Section 4.3.1, we use examples to discuss some key ideas of the decomposition. Then, in Section 4.3.2, we give the full definitions of minimal session types and the decomposition functions. We define decomposition functions for types $\mathcal{G}(-)$ and for processes $\mathcal{D}(-)$. The former "slices" a session type S and returns a list of minimal session types, corresponding to individual actions in S; the latter breaks down an HO process into a parallel composition of processes. We demonstrate these notions on a number of examples in Section 4.3.3. Finally, in Section 4.3.4 we establish the *static correctness* result (Theorem 4.3.1): if P is well-typed under session types $S_1, \ldots, S_n$, then $\mathcal{D}(P)$ is typable using the minimal session types $\mathcal{G}(S_1), \ldots, \mathcal{G}(S_n)$. The issue of dynamic correctness, i.e., the operational correspondence between P and $\mathcal{D}(P)$, is treated separately in Section 4.4.

Remark 3 (Color Convention)**.** We use colors to differentiate the operations on processes (in pink) and on types (in green). The usage of the colors is for visual aid only, and is not important for the mathematical content of the presented material.

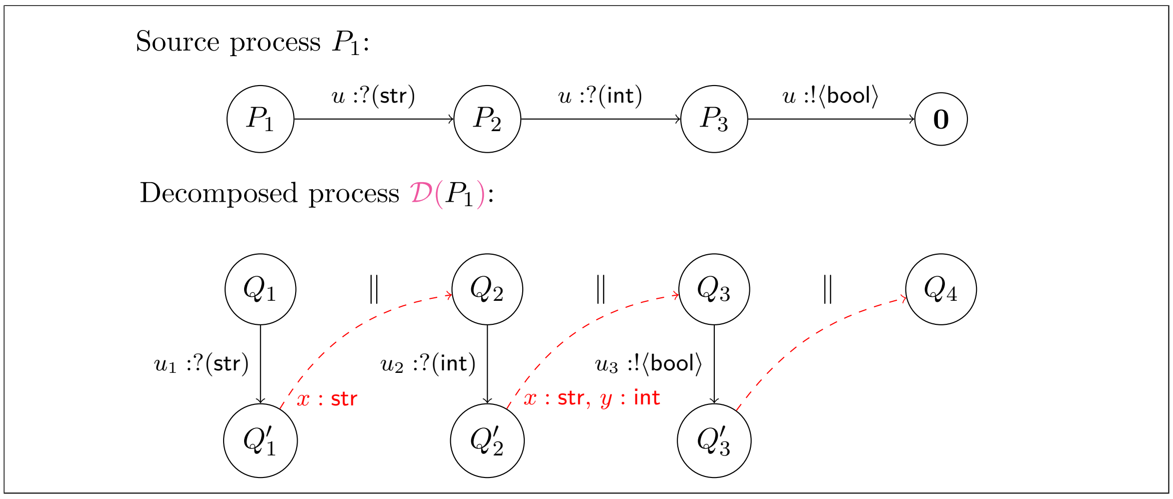


Figure 4.2: Our decomposition function $\mathcal{D}(-)$, illustrated. Nodes represent process states, '∥' represents parallel composition of processes, black arrows stand for actions, and red arrows indicate synchronizations that preserve the sequentiality of the source process by activating trios and propagating (bound) values.

4.3.1 Key Ideas

Consider a process P_1 that implements the (standard) session type $S = ?(\mathsf{str});?(\mathsf{int});!\langle\mathsf{bool}\rangle;\mathtt{end}$ along name u. In process P_1, name u is not a single-use resource; rather, it is used several times to implement the communication actions in S; Figure 4.2 (top) graphically depicts the actions and the corresponding states.

The decomposition $\mathcal{D}(P_1)$ is illustrated in the bottom part of Figure 4.2: it is defined as the parallel composition of four processes Q_i (for $i \in \{1, \ldots, 4\}$). Each process Q_1, Q_2, and Q_3 mimic one action of P_1 on an indexed name u_i, while Q_4 simulates the termination of the session. This way, a single name u in P_1 is decomposed into a sequence of names u_1, u_2, u_3 in $\mathcal{D}(P_1)$.

The processes Q_1, Q_2, Q_3, and Q_4 are composed in parallel, but we would like to retain the same sequentiality of actions on the channels u_i as we have on the channel u. To that end, each process Q_i, with the exception of Q_1, does not perform its designated action on u_i until it gets activated by the previous process. In turn, after Q_i performs an action on u_i it evolves to a state Q_i', which is responsible for activating the next process Q_{i+1}. In Figure 4.2, the activations are indicated by red arrows. In general, the decomposition orchestrates the activation of sub-processes, following the sequencing prescribed by the session types of the given process. Therefore, assuming a well-typed source process, our decomposition codifies the sequentiality in session types into the process level.

The activation mechanism includes the *propagation* of values across sub-processes (cf. the labels on red arrows). This establishes a flow of values from sub-processes binding them to those that use them (i.e., it makes variable bindings explicit). For example, in P_1, the Boolean value being sent over as part of the session S might depend on the previously received string and integer values. Therefore, both of those values have to be propagated to the process Q_3, which is responsible for sending out the Boolean.

In this example a single name $u : S$ is decomposed into a sequence $\widetilde{u} = (u_1, \ldots, u_n)$: each $u_i \in \widetilde{u}$ is a single-use resource, as prescribed by its minimal session type. Such is the case for non-recursive types S. When S is recursive, the situation is more interesting: each action of S can be repeated many times, and therefore the names $\widetilde{u}$ should be propagated across trios to enable potentially many uses. As an example, consider the recursive session type

$S = \mu t.?(\mathsf{int});!\langle \mathsf{int}\rangle;t$, in which an input and an output actions are repeated indefinitely. Consider the following process

$$R_1 = \underbrace{r?(z).}_{T_1}\underbrace{r!\langle -z\rangle.}_{T_2}\underbrace{r?(z).}_{T_3}\underbrace{r!\langle z\rangle.}_{T_4}\underbrace{V\,r}_{T_5}$$

which makes use of the channel $r : S$ and where V has type $S \rightarrow \diamond$. Figure 4.3 (top) gives the first four actions of R_1 and the corresponding sates: the body of type S prescribes two actions on name r, performed sequentially in R_1 and R_2; subsequent actions (enabled in R_3 and R_4) correspond to a "new instance" of the body of S.

The decomposition $\mathcal{D}(R_1)$, depicted in Figure 4.3 (bottom), generates a trio process for each prefix in R_1; we denote prefixes with their corresponding trios $T_1, \dots, T_5$. The type decomposition function on types, $\mathcal{G}(-)$, slices S into two *minimal tail-recursive types*: $M_1 = \mu t.?(\mathsf{int});t$ and $M_2 = \mu t.!\langle \mathsf{int}\rangle;t$.

In the recursive case, a key idea is that trios that mimic actions prescribed by a recursive session types should reuse names, which should be propagated across trios. This way, for instance, trios T_1 and T_3 mimic the same (input) action, and so they both should use the same name (r_1). To achieve this, we devise a mechanism that propagates names with tail-recursive types (such as (r_1, r_2)) through the trios. These propagation actions are represented by blue arrows in Figure 4.3 (bottom). In our example, T_3 gathers the complete decomposition of names from preceding trios (r_1, r_2); it mimics an input action on r_1 and makes (r_1, r_2) available to future trios (i.e., T_4 and T_5).

Since the same tail-recursive names can be (re)used infinitely often, we propagate tail-recursive names through the following process. All the names $\widetilde{r}$ corresponding to the decomposition of a tail-recursive name r are bound in the process

$$c^r?(x).x\,\widetilde{r},$$

which is similar to the servers discussed in Example 4.2.1. We call these processes *recursive propagators*, and each tail-recursive name in the original process P has a dedicated propagator in $\mathcal{D}(P)$ on the channel c^r. Whenever a trio has to perform an action $\alpha(r_i)$ on one of the decomposed tail-recursive names (i.e., a decomposition of an input action '$r?(y)$.' or an output action '$r!\langle V\rangle$.' on the name r), it first has to request the name from the corresponding recursive propagator by performing an output action $c^r!\langle N\rangle$, where value N is the abstraction

$$N = \lambda\widetilde{z}.\,\alpha(z_i).(\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid c^r?(x).x\,\widetilde{z}).$$

A synchronization on c^r will result in the reduction:

$$c^r?(x).x\,\widetilde{r} \mid c^r!\langle N\rangle \longrightarrow \alpha(r_i).(\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid c^r?(x).x\,\widetilde{r}).$$

The resulting process first simulates $\alpha(r)$ and subsequently reinstates the recursive propagator on c^r, for the benefit of the other trios requiring access to the names $\widetilde{r}$. See Examples 4.3.9 and 4.3.10 below (Page 82) for further illustration of this method.

This decomposition strategy handles HO processes with recursive types which are *simple* and *contractive*. That is, recursive types of the form $\mu t.S$, where the body $S \neq t$ does not itself contain recursive types. Unless stated otherwise, we consider *tail-recursive* session types such as, e.g., $S = \mu t.?(\mathsf{int});?(\mathsf{bool});!\langle\mathsf{bool}\rangle;t$. Non-tail-recursive session types such as $\mu t.?((\widetilde{T}, t) \rightarrow \diamond);\mathtt{end}$, used in the fully-abstract encoding of HOπ into HO [39], can also be accommodated; see Example 4.3.3 below.

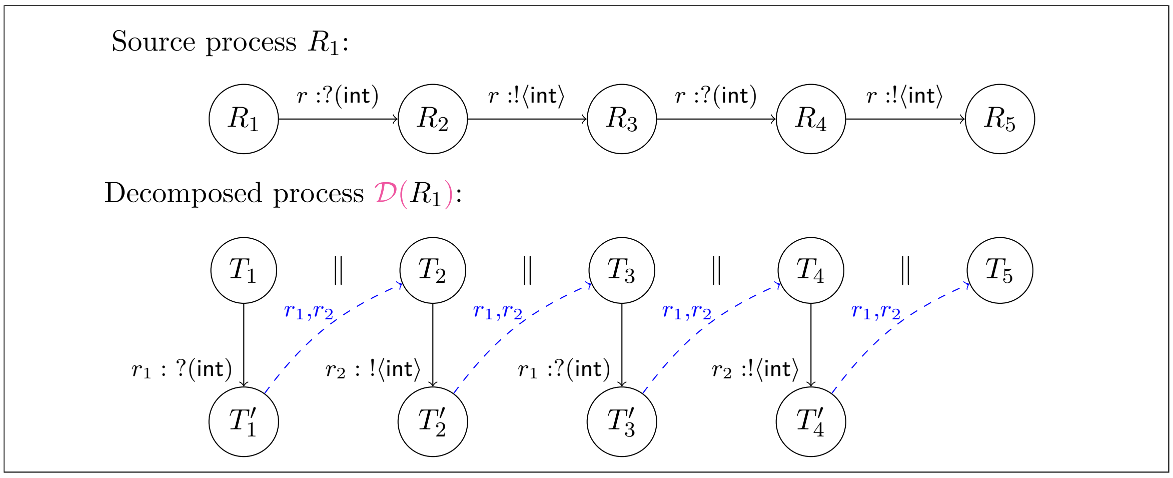

Source process R_1:
R_1 r :?(int) R_2 r :!⟨int⟩ R_3 r :?(int) R_4 r :!⟨int⟩ R_5
Decomposed process $\mathcal{D}(R_1)$:
T_1 ∥ T_2 ∥ T_3 ∥ T_4 ∥ T_5
r_1,r_2
r_1 : ?(int)
r_2 : !⟨int⟩
r_1 :?(int)
r_2 :!⟨int⟩
T_1' T_2' T_3' T_4'

Figure 4.3: Decomposition of processes with recursive session types, illustrated. Dashed blue arrows represent the propagation of tail-recursive names (r_1,r_2) across trios.

4.3.2 The Decomposition

Here we formally present the decomposition of HO processes. We start introducing some preliminary definitions, including the definition of an auxiliary function, called the *breakdown function*.

Following Parrow [50] we adopt some useful terminology and notation on trios. The *context* of a trio is a tuple of variables $\widetilde{x}$, possibly empty, which makes variable bindings explicit. We use a reserved set of *propagator names* (or simply *propagators*), denoted with $c_k, c_{k+1}, \ldots$, to carry contexts and trigger the subsequent trio. A process with less than three sequential prefixes is called a *degenerate trio*. Also, a *leading trio* is the one that receives a context, performs an action, and triggers the next trio; a *control trio* only activates other trios.

The breakdown function works on both processes and values. The breakdown of process P is denoted by $\mathcal{B}^k_{\tilde{x}}(P)$, where k is the index for the propagators c_k, and $\widetilde{x}$ is the context to be received by the previous trio. Similarly, the breakdown of a value V is denoted by $\mathcal{V}_{\tilde{x}}(V)$.

Minimal Session Types and Decomposing Types

We start by introducing minimal session types as a fragment of Figure 2.4:

Definition 4.3.1 (Minimal Session Types)**.** The syntax of *minimal session types* for HO is defined as follows:

$$\begin{array}{rcl} U & ::= & \widetilde{C}\rightarrow\diamond \mid \widetilde{C}\multimap\diamond \\ C & ::= & M \mid \langle U\rangle \\ M & ::= & \gamma \mid\ !\langle\widetilde{U}\rangle;\gamma \mid\ ?(\widetilde{U});\gamma \mid \mu\mathsf{t}.M \\ \gamma & ::= & \mathsf{end} \mid \mathsf{t} \end{array}$$

The above definition is minimal in its use of sequencing, which is only present in recursive session types such as $\mu\mathsf{t}.!\langle U\rangle;\mathsf{t}$ and $\mu\mathsf{t}.?(U);\mathsf{t}$—these are tail-recursive session types with exactly one session prefix. Clearly, this minimal type structure induces a reduced set of typable HO processes. A type system for HO based on minimal session types can be straightforwardly obtained by specializing the definitions, typing rules, and results summarized in Section 2.2. We refer to HO processes and terms typeable with minimal session types as *MST processes and terms*, respectively.

$$
\begin{aligned}
\mathcal{G}(!\langle U\rangle;S) &= \begin{cases} !\langle \mathcal{G}(U)\rangle & \text{if } S = \mathsf{end} \\ !\langle \mathcal{G}(U)\rangle\,, \mathcal{G}(S) & \text{otherwise} \end{cases} \\
\mathcal{G}(?(U);S) &= \begin{cases} ?(\mathcal{G}(U)) & \text{if } S = \mathsf{end} \\ ?(\mathcal{G}(U))\,, \mathcal{G}(S) & \text{otherwise} \end{cases} \\
\mathcal{G}(\mu \mathsf{t}.S) &= \begin{cases} \mathcal{R}(S) & \text{if } \mathsf{tr}(\mu \mathsf{t}.S) \\ \mu \mathsf{t}.\mathcal{G}(S) & \text{if } \neg\mathsf{tr}(\mu \mathsf{t}.S) \text{ and } \mathcal{G}(S) \text{ is a singleton} \end{cases} \\
\mathcal{G}(\mathsf{end}) &= \mathsf{end} \\
\mathcal{G}(\mathsf{t}) &= \mathsf{t} \\
\mathcal{G}(C \multimap \diamond) &= \mathcal{G}(C) \multimap \diamond \\
\mathcal{G}(C \rightarrow \diamond) &= \mathcal{G}(C) \rightarrow \diamond \\
\mathcal{G}(\langle U\rangle) &= \langle \mathcal{G}(U)\rangle \\
\mathcal{R}(\mathsf{t}) &= \epsilon \\
\mathcal{R}(!\langle U\rangle;S) &= \mu \mathsf{t}.!\langle \mathcal{G}(U)\rangle;\mathsf{t}, \mathcal{R}(S) \\
\mathcal{R}(?(U);S) &= \mu \mathsf{t}.?(\mathcal{G}(U));\mathsf{t}, \mathcal{R}(S)
\end{aligned}
$$

Figure 4.4: Decomposing session types into minimal session types (Definition 4.3.3)

We now define how to "slice" a standard session type into a *list* of minimal session types. We need the following auxiliary definition.

Definition 4.3.2 (Predicates on Types and Names)**.** Let C be a channel type.

- We write $\mathsf{tr}(C)$ to indicate that C is a tail-recursive session type.
- Given $u : C$, we write $\mathtt{lin}(u)$ if a session type (i.e. $C = S$ for some S) that is not tail recursive.

With a slight abuse of notation, we write $\mathsf{tr}(u)$ to mean $u : C$ and $\mathsf{tr}(C)$ (and similarly for $\neg\mathsf{tr}(u)$).

Definition 4.3.3 (Decomposing Session Types)**.** Given the session, higher-order, and shared types of Figure 2.4, the *type decomposition function* $\mathcal{G}(-)$ is defined using the auxiliary function $\mathcal{R}(-)$ as in Figure 4.4. We write $|\mathcal{G}(S)|$ to denote the length of $\mathcal{G}(S)$ (and similarly for $\mathcal{R}(-)$).

The decomposition is self-explanatory; intuitively, if a session type S contains k input/output actions, the list $\mathcal{G}(S)$ will contain k minimal session types. For a tail recursive $\mu \mathsf{t}.S$, $\mathcal{G}(\mu \mathsf{t}.S)$ is a list of minimal recursive session types, obtained using the auxiliary function $\mathcal{R}(-)$ on S: if S has k prefixes then the list $\mathcal{G}(\mu \mathsf{t}.S)$ will contain k minimal recursive session types.

We illustrate Definition 4.3.3 with three examples.

Example 4.3.1 (Decomposition a Non-recursive Type)**.** Let $S = ?(\mathsf{int});?(\mathsf{int});!\langle\mathsf{bool}\rangle;\mathsf{end}$ be the session type given in Section 4.1. Then $\mathcal{G}(S)$ denotes the list $?(\mathsf{int})\,, ?(\mathsf{int})\,, !\langle\mathsf{bool}\rangle$. ◁

Example 4.3.2 (Decomposing a Recursive Type)**.** Let $S = \mu \mathsf{t}.S'$ be a recursive session type, with $S' = ?(\mathsf{int});?(\mathsf{bool});!\langle\mathsf{bool}\rangle;\mathsf{t}$. By Definition 4.3.3, since S is tail-recursive, $\mathcal{G}(S) = \mathcal{R}(S')$. Further, $\mathcal{R}(S') = \mu \mathsf{t}.?(\mathcal{G}(\mathsf{int}));\mathsf{t}, \mathcal{R}(?(\mathsf{bool});!\langle\mathsf{bool}\rangle;\mathsf{t})$. By definition of $\mathcal{R}(-)$, we obtain

$$\mathcal{G}(S) = \mu \mathsf{t}.?(\mathsf{int});\mathsf{t},\ \mu \mathsf{t}.?(\mathsf{bool});\mathsf{t},\ \mu \mathsf{t}.!\langle\mathsf{bool}\rangle;\mathsf{t}, \mathcal{R}(t)$$

(using $\mathcal{G}(\mathsf{int}) = \mathsf{int}$ and $\mathcal{G}(\mathsf{bool}) = \mathsf{bool}$). Since $\mathcal{R}(\mathsf{t}) = \epsilon$, we obtain

$$\mathcal{G}(S) = \mu\mathsf{t}.?(\mathsf{int});\mathsf{t},\ \mu\mathsf{t}.?(\mathsf{bool});\mathsf{t},\ \mu\mathsf{t}.!\langle\mathsf{bool}\rangle;\mathsf{t}$$

$\triangleleft$

In addition to tail-recursive types that are handled by $\mathcal{R}(-)$, we need to support non-tail-recursive types of form $\mu\mathsf{t}.?((\widetilde{T},\mathsf{t})\to\diamond);\mathtt{end}$ that are essential for the encoding of recursion in HOπ into HO. The following example illustrates such a decomposition.

Example 4.3.3 (Decomposing a Non-tail-recursive Type)**.** Let $S = \mu\mathsf{t}.?((?(\mathsf{str});!\langle\mathsf{str}\rangle;\mathtt{end},\mathsf{t})\to\diamond);\mathtt{end}$ be a non-tail-recursive type. We obtain the following decomposition:

$$\begin{aligned}\mathcal{G}(S) &= \mu\mathsf{t}.\mathcal{G}(?((?(\mathsf{str});!\langle\mathsf{str}\rangle;\mathtt{end},\mathsf{t})\to\diamond);\mathtt{end})\\ &= \mu\mathsf{t}.?(\mathcal{G}((?(\mathsf{str});!\langle\mathsf{str}\rangle;\mathtt{end},\mathsf{t})\to\diamond))\\ &= \mu\mathsf{t}.?((?(\mathsf{str}),!\langle\mathsf{str}\rangle,\mathsf{t})\to\diamond) = M\end{aligned}$$

We can see that we have generated minimal non-tail-recursive type M. $\triangleleft$

Now, we illustrate the encoding of HOπ recursive processes into HO from [39] using the non-tail-recursive type S given in the above example.

Example 4.3.4 (Encoding Recursion)**.** Consider the process $P = \mu X.a?(m).a!\langle m\rangle.X$, which contains recursion and so it is not an HO process. Still, P can be encoded into HO as follows [39]:

$$[\![P]\!] = a?(m).a!\langle m\rangle.(\nu\, s)\,(V\,(a,s) \mid \overline{s}!\langle V\rangle)$$

where the value V is an abstraction that potentially reduces to $[\![P]\!]$:

$$V = \lambda(x_a,y_1).\,y_1?(z_x).x_a?(m).x_a!\langle m\rangle.(\nu\, s)\,(z_x\,(x_a,s) \mid \overline{s}!\langle z_x\rangle.\mathbf{0})$$

As detailed in [39], this encoding relies on non-tail-recursive types. In particular, the bound name s in $[\![P]\!]$ is typed with the following type, discussed above in Example 4.3.3:

$$S = \mu\mathsf{t}.?((?(\mathsf{str});!\langle\mathsf{str}\rangle;\mathtt{end},\mathsf{t})\to\diamond);\mathtt{end}$$

We compose $[\![P]\!]$ with an appropriate client process to illustrate the encoding of recursion. Below R stands for some unspecified process such that $a \in \mathtt{rn}(R)$:

$$\begin{aligned}[\![P]\!] \mid a!\langle W\rangle.a?(b).R &\longrightarrow^2 (\nu\, s)\,(V\,(a,s) \mid \overline{s}!\langle V\rangle) \mid R\\ &\longrightarrow (\nu\, s)\,(s?(z_x).a?(m).a!\langle m\rangle.(\nu\, s')\,(z_x\,(a,s') \mid \overline{s'}!\langle z_x\rangle) \mid \overline{s}!\langle V\rangle) \mid R\\ &\longrightarrow a?(m).a!\langle m\rangle.(\nu\, s')\,(V\,(a,s') \mid \overline{s'}!\langle V\rangle) \mid R\\ &= [\![P]\!] \mid R\end{aligned}$$

Decomposing Processes

As we have seen, each session type S is decomposed into $\mathcal{G}(S)$, a list of minimal session types. Accordingly, given an assignment $s : S$, we decompose s into a series of names, one for each action in S. We use *indexed names* to formalize the names used by minimally typed processes. Formally, an indexed name is a pair (n,i) with $i \in \mathbb{N}$, which we denote as n_i. We refer to processes with indexed names as *indexed processes*.

The decomposition of processes is defined in Definition 4.3.9, and it relies on a breakdown function, denoted $\mathcal{B}^k_{\tilde{x}}(-)$, which operates on indexed processes. Before we dive into those functions we present some auxiliary definitions.

Preliminaries. To handle the unfolding of recursive types, we shall use the following auxiliary function, which decomposes guarded recursive types, by first ignoring all the actions until the recursion.

Definition 4.3.4 (Decomposing an Unfolded Recursive Type)**.** Let S be a session type. The function $\mathcal{R}^{\star}(-)$: is defined as follows

$$\begin{aligned}\mathcal{R}^{\star}(\mu \mathsf{t}.S) &= \mathcal{R}(S)\\ \mathcal{R}^{\star}(?(U);S) &= \mathcal{R}^{\star}(S)\\ \mathcal{R}^{\star}(!\langle U\rangle;S) &= \mathcal{R}^{\star}(S)\end{aligned}$$

Example 4.3.5. Let $T = ?(\mathsf{bool});!\langle\mathsf{bool}\rangle;S$ be a derived unfolding of S from Example 4.3.2. Then, by Definition 4.3.3, $\mathcal{R}^{\star}(T)$ is the list of minimal recursive types obtained as follows: first, $\mathcal{R}^{\star}(T) = \mathcal{R}^{\star}(!\langle\mathsf{bool}\rangle;\mu\mathsf{t}.S')$ and after one more step, $\mathcal{R}^{\star}(!\langle\mathsf{bool}\rangle;\mu\mathsf{t}.S') = \mathcal{R}^{\star}(\mu\mathsf{t}.S')$. Finally, we have $\mathcal{R}^{\star}(\mu\mathsf{t}.S') = \mathcal{R}(S')$. We get the same list of minimal types as in Example 4.3.2: $\mathcal{R}^{\star}(T) = \mu\mathsf{t}.?(\mathsf{int});\mathsf{t}, \mu\mathsf{t}.?(\mathsf{bool});\mathsf{t}, \mu\mathsf{t}.!\langle\mathsf{bool}\rangle;\mathsf{t}$. $\lhd$

Given an unfolded recursive session type S, the auxiliary function $[S\rangle$ returns the position of the top-most prefix of S within its body.

Definition 4.3.5 (Index function)**.** Let S be an (unfolded) recursive session type. The function $[S\rangle$ is defined as follows:

$$[S\rangle = \begin{cases}[S'\{^{S}/_{\mathsf{t}}\}\rangle^{\star}_{0} & \text{if } S = \mu\mathsf{t}.S'\\ [S\rangle^{\star}_{0} & \text{otherwise}\end{cases}$$

where $[S\rangle^{\star}_{l}$:

$$\begin{aligned}[\mu\mathsf{t}.S\rangle^{\star}_{l} &= |\mathcal{R}(S)| - l + 1\\ [!\langle U\rangle;S\rangle^{\star}_{l} &= [S\rangle^{\star}_{l+1}\\ [?(U);S\rangle^{\star}_{l} &= [S\rangle^{\star}_{l+1}\end{aligned}$$

Example 4.3.6. Let $S' = ?(\mathsf{bool});!\langle\mathsf{bool}\rangle;S$ where S is as in Example 4.3.2. Then $[S'\rangle = 2$ since the top-most prefix of S' ('$?(\mathsf{bool});$') is the second prefix in the body of S. $\lhd$

In order to determine the required number of propagators $(c_k, c_{k+1}, \ldots)$ required in the breakdown of processes and values, we define the *degree* of a process:

Definition 4.3.6 (Degree of a Process)**.** Let P be an HO process. The *degree* of P, denoted $\wr P \int$, is defined as follows:

$$\wr P \int = \begin{cases}\wr Q \int + 1 & \text{if } P = u_i!\langle V\rangle.Q \text{ or } P = u_i?(y).Q\\ \wr P' \int & \text{if } P = (\nu\, s : S)\, P'\\ \wr Q \int + \wr R \int + 1 & \text{if } P = Q \mid R\\ 1 & \text{if } P = V\, u_i \text{ or } P = \mathbf{0}\end{cases}$$

We define an auxiliary function that "initializes" the indices of a tuple of names, for turning a regular process into an indexed process.

Definition 4.3.7 (Initializing an indexed process)**.** Let $\widetilde{u} = (a, b, s, s', r, r', \ldots)$ be a finite tuple of names. We shall write $\mathsf{init}(\widetilde{u})$ to denote the tuple of indexed names $(a_1, b_1, s_1, s'_1, r_1, r'_1, \ldots)$.

Definition 4.3.8 (Subsequent index substitution)**.** Let n_i be an indexed name. We define $\mathsf{next}(n_i) = (\mathtt{lin}(n_i))\,?\,\{^{n_{i+1}}/_{n_i}\}\!:\{\}$.

Remark 4. Recall that we write '$c_k?()$' and '$\overline{c_k}!\langle\rangle$' to denote input and output prefixes in which the value communicated along c_k is not relevant. While '$c_k?()$' stands for '$c_k?(x)$', '$\overline{c_k}!\langle\rangle$' stands for '$\overline{c_k}!\langle\lambda x.\, \mathbf{0}\rangle$'. Their corresponding minimal types are $?(\mathtt{end} \to \diamond)$ and $!\langle\mathtt{end} \to \diamond\rangle$, which are denoted by $?(-)$ and $!\langle-\rangle$, respectively.

Given a typed process P, we write $\mathtt{rn}(P)$ to denote the set of free names of P whose types are recursive. As mentioned above, for each $r \in \mathtt{rn}(P)$ with $r : S$ we shall rely on a control trio of the form $c^r?(x).x\,\widetilde{r}$, where $\widetilde{r} = r_1, \ldots, r_{|\mathcal{G}(S)|}$.

Definition 4.3.9 (Decomposition of a Process)**.** Let P be a closed HO process with $\widetilde{u} = \mathtt{fn}(P)$ and $\widetilde{v} = \mathtt{rn}(P)$. The *decomposition* of P, denoted $\mathcal{D}(P)$, is defined as:

$$\mathcal{D}(P) = (\nu\,\widetilde{c})\,(\nu\,\widetilde{c}_r)\left(\overline{c_k}!\langle\rangle.\mathbf{0} \mid \mathcal{B}^k_\epsilon(P\sigma) \mid \prod_{r\in\widetilde{v}} c^r?(x).x\,\widetilde{r}\right)$$

where: $k > 0$; $\widetilde{c} = (c_k, \ldots, c_{k+\wr P \wr -1})$; $\widetilde{c_r} = \bigcup_{r\in\widetilde{v}} c^r$; $\sigma = \{{}^{\mathsf{init}(\widetilde{u})}/_{\widetilde{u}}\}$.

Notice that when $\mathtt{rn}(P) = \emptyset$, then $\mathcal{D}(P) = (\nu\,\widetilde{c})\,(\overline{c_k}!\langle\rangle.\mathbf{0} \mid \mathcal{B}^k_\epsilon(P\sigma))$. We now discuss the breakdown of process P, denoted $\mathcal{B}^k_{\widetilde{x}}(P)$.

The Breakdown Function. Given a context $\widetilde{x}$ and a $k > 0$, the breakdown of an indexed process P, denoted $\mathcal{B}^k_{\widetilde{x}}(P)$, is defined recursively on the structure of processes. The definition of $\mathcal{B}^k_{\widetilde{x}}(-)$ relies on an auxiliary breakdown function on values, denoted $\mathcal{V}_{\widetilde{x}}(-)$. When $V = y$, then the breakdown function is simply the identity: $\mathcal{V}_{\widetilde{x}}(y) = y$.

The breakdown function relies on type information, in two ways. First, names are decomposed based on their session types. Second, for most constructs the shape of decomposed process depends on whether the associated session type is tail-recursive or not. The definition of the breakdown function is given in Table 4.1. Next, we describe each of the cases of the definition. In Section 4.3.3 (Page 79) we develop several examples.

Output: The decomposition of $u_i!\langle V\rangle.Q$ is arguably the most interesting case, as both the sent value V and the continuation Q have to be decomposed. We distinguish two cases:

- If $\neg\mathsf{tr}(u_i)$ then u_i is linear or shared, and then we have:

$$\mathcal{B}^k_{\widetilde{x}}(u_i!\langle V\rangle.Q) = c_k?(\widetilde{x}).u_i!\langle\mathcal{V}_{\widetilde{y}}(V\sigma)\rangle.\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(Q\sigma)$$

  This decomposition consists of a leading trio that mimics an output action in parallel with the breakdown of Q. The context $\widetilde{x}$ must include the free variables of V and Q, which are denoted $\widetilde{y}$ and $\widetilde{w}$, respectively. These tuples are not necessarily disjoint: variables with shared types can appear free in both V and Q. The value V is then broken down with parameters $\widetilde{y}$ and $k+1$; the latter serves to consistently generate propagators for the trios in the breakdown of V, denoted $\mathcal{V}_{\widetilde{y}}(V\sigma)$ (see below). The substitution σ increments the index of session names; it is applied to both V and Q before they are broken down. By taking $\sigma = \mathsf{next}(u_i)$ we distinguish two cases (see Definition 4.3.8):

  - If name u_i is linear (i.e., it has a session type) then its future occurrences are renamed into u_{i+1}, and $\sigma = \{{}^{u_{i+1}}/_{u_i}\}$;
  - Otherwise, if u_i is shared, then $\sigma = \{\}$.

P	$\mathcal{B}^{k}_{\tilde{x}}(P)$	
$u_i!\langle V\rangle.Q$	• $\neg\mathsf{tr}(S)$: $c_k?(\widetilde{x}).u_i!\langle \mathcal{V}_{\tilde{y}}(V\sigma)\rangle.\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(Q\sigma)$ • $\mathsf{tr}(S)$: $c_k?(\widetilde{x}).c^u!\langle N_V\rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(Q)$ where: $N_V = \lambda\widetilde{z}.\, z_{[S\rangle}!\langle \mathcal{V}_{\tilde{y}}(V)\rangle.(\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid c^u?(x).x\,\widetilde{z})$	$u_i : S$ $\widetilde{y} = \mathtt{fv}(V)$ $\widetilde{w} = \mathtt{fv}(Q)$ $\sigma = \mathsf{next}(u_i)$ $\widetilde{z} = (z_1, \dots, z_{\lvert\mathcal{R}^\star(S)\rvert})$
$u_i?(y).Q$	• $\mathsf{tr}(S)$: $c_k?(\widetilde{x}).u_i?(y).\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(Q\sigma)$ • $\neg\mathsf{tr}(S)$: $c_k?(\widetilde{x}).c^u!\langle N_y\rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(Q)$ where: $N_y = \lambda\widetilde{z}.\, z_{[S\rangle}?(y).(\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid c^u?(x).x\,\widetilde{z})$	$u_i : S$ $\widetilde{w} = \mathtt{fv}(Q)$ $\sigma = \mathsf{next}(u_i)$ $\widetilde{z} = (z_1, \dots, z_{\lvert\mathcal{R}^\star(S)\rvert})$
$V\,(\widetilde{r}, u_i)$	$c_k?(\widetilde{x}).\,\overbrace{c^{r_1}!\langle\lambda\widetilde{z}_1.c^{r_2}!\langle\lambda\widetilde{z}_2.\cdots.c^{r_n}!\langle\lambda\widetilde{z}_n.}^{n=\lvert\tilde{r}\rvert}\,Q\rangle\,\rangle\rangle$ where: $Q = \mathcal{V}_{\tilde{x}}(V)\,(\widetilde{z}_1, \dots, \widetilde{z}_n, \widetilde{m})$	$u_i : C$ $\forall r_i \in \widetilde{r}.(r_i : S_i \wedge \mathsf{tr}(S_i) \wedge$ $\widetilde{z}_i = (z^i_1, \dots, z^i_{\lvert\mathcal{R}^\star(S_i)\rvert}))$ $\widetilde{m} = (u_i, \dots, u_{i+\lvert\mathcal{G}(C)\rvert-1})$
$(\nu\, s : C)\, P'$	•$\neg\mathsf{tr}(C)$: $(\nu\,\widetilde{s} : \mathcal{G}(C))\ \mathcal{B}^{k}_{\tilde{x}}(P'\sigma)$ • $\mathsf{tr}(C)$: $(\nu\,\widetilde{s} : \mathcal{G}(C))\,(\nu\, c^s)\, c^s?(x).x\,\widetilde{s} \mid$ $(\nu\, c^{\bar{s}})\, c^{\bar{s}}?(x).x\,\widetilde{\bar{s}} \mid\ \mathcal{B}^{k}_{\tilde{x}}(P'\sigma)$	$\widetilde{x} = \mathtt{fv}(P')$ $\widetilde{s} = (s_1, \dots, s_{\lvert\mathcal{G}(C)\rvert})$ $\widetilde{\bar{s}} = (\overline{s_1}, \dots, \overline{s_{\lvert\mathcal{G}(C)\rvert}})$ $\sigma = \{{}^{s_1\overline{s_1}}/_{s\bar{s}}\}$
$Q \mid R$	$c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle\widetilde{y}\rangle.\overline{c_{k+l+1}}!\langle\widetilde{w}\rangle \mid \mathcal{B}^{k+1}_{\tilde{y}}(Q) \mid \mathcal{B}^{k+l+1}_{\tilde{w}}(R)$	$\widetilde{y} = \mathtt{fv}(Q)$ $\widetilde{w} = \mathtt{fv}(R)$ $l = \wr Q \int$
$\mathbf{0}$	$c_k?().\mathbf{0}$	
V	$\mathcal{V}_{\tilde{x}}(V)$	
y	y	
$\lambda(\widetilde{y}z).\, P$	$\lambda(\widetilde{y^1}, \dots, \widetilde{y^n}, \widetilde{z}) : (\widetilde{M})^{\leadsto}.\, N$ where: $\widetilde{M} = (\mathcal{G}(S_1), \dots, \mathcal{G}(S_n), \mathcal{G}(C))$ $N = (\nu\,\widetilde{c})\,(\nu\,\widetilde{c}_r)\,\prod_{i\in\lvert\widetilde{y}\rvert}(c^{y_i}?(x).x\,\widetilde{y}^i) \mid \overline{c_1}!\langle\widetilde{x}\rangle \mid$ $\mathcal{B}^{1}_{\tilde{x}}(P\{{}^{z_1}/_{z}\})$	$\widetilde{y}z : \widetilde{S}C$ $\forall y_i \in \widetilde{y}.(y_i : S_i \wedge \mathsf{tr}(S_i) \wedge$ $\widetilde{y^i} = (y^i_1, \dots, y^i_{\lvert\mathcal{G}(S_i)\rvert}))$ $\widetilde{z} = (z_1, \dots, z_{\lvert\mathcal{G}(C)\rvert})$ $\widetilde{c} = (c_1, \dots, c_{\wr P \int})$ $\widetilde{c}_r = \bigcup_{r\in\tilde{y}} c^r$

Table 4.1: The breakdown function for processes and values.

Note that if u_i is linear then it appears either in V or Q and σ affects only one of them. The last prefix activates the breakdown of Q with its corresponding context $\widetilde{w}$.

In case $V = y$, the same strategy applies; because $\mathcal{V}_{\tilde{y}}(y\sigma) = y$, we have:

$$\mathcal{B}^{k}_{\tilde{x}}(u_i!\langle y\rangle.Q) = c_k?(\widetilde{x}).u_i!\langle y\rangle.\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(Q\sigma)$$

Notice that variable y is not propagated further if it does not appear free in Q.

- If $\mathtt{tr}(u_i)$ then u_i is tail-recursive and then we have:

$$\begin{aligned}\mathcal{B}^{k}_{\tilde{x}}(u_i!\langle V\rangle.Q) =&\ \ c_k?(\widetilde{x}).c^u!\langle N_V\rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(Q)\\ \text{where: }\ N_V =&\ \ \lambda\widetilde{z}.\, z_{[S\rangle}!\langle \mathcal{V}_{\tilde{y}}(V)\rangle.(\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid c^u?(x).x\,\widetilde{z})\end{aligned}$$

The decomposition consists of a leading trio that mimics the output action running in parallel with the breakdown of Q. After receiving the context $\widetilde{x}$, the leading trio sends an abstraction N_V along c^u, which performs several tasks. First, N_V collects the sequence of names $\tilde{u}$; then, it mimics the output action of P along one of such names ($u_{[S\rangle}$) and triggers the next trio, with context $\widetilde{w}$; finally, it reinstates the server on c^u for the next trio that uses u. Notice that indexing is not relevant in this case.

In case $V = y$, we have $\mathcal{V}_{\tilde{y}}(y\sigma) = y$ and $\wr y \int = 0$, hence:

$$\mathcal{B}^{k}_{\tilde{x}}(u_i!\langle y\rangle.Q) = c_k?(\widetilde{x}).c^u!\langle\lambda\widetilde{z}.\, z_{[S\rangle}!\langle y\rangle.(\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid c^u?(x).x\,\widetilde{z})\rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(Q)$$

Input: To decompose a process $u_i?(y).Q$ we distinguish two cases, as before: (i) name u_i is linear or shared or (ii) tail-recursive. In case (i), the breakdown is defined as follows:

$$\mathcal{B}^{k}_{\tilde{x}}(u_i?(y).Q) = c_k?(\widetilde{x}).u_i?(y).\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(Q\sigma)$$

where $\widetilde{w} = \mathtt{fv}(Q)$. A leading trio mimics the input action and possibly extends the context with the received variable y. The substitution σ is defined as in the output case.

In case (ii), when u_i has tail-recursive session type S, the decomposition is as in the output case:

$$\mathcal{B}^{k}_{\tilde{x}}(u_i?(y).Q) = c_k?(\widetilde{x}).c^u!\langle\lambda\widetilde{z}.\, z_{[S\rangle}?(y).(\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid c^u?(x).x\,\widetilde{z})\rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(Q)$$

Application: For simplicity we consider the breakdown of applications of the form $V\,(\widetilde{r}, u_i)$, where every $r_i \in \widetilde{r}$ is such that $\mathtt{tr}(r_i)$ and only u_i is such that $\neg\mathtt{tr}(u_i)$. The general case (involving different orders in names and multiple names with non-recursive types) is similar. We have:

$$\mathcal{B}^{k}_{\tilde{x}}(V\,(\widetilde{r}, u_i)) = c_k?(\widetilde{x}).\,\overbrace{c^{r_1}!\langle\lambda\widetilde{z}_1.c^{r_2}!\langle\lambda\widetilde{z}_2.\cdots.c^{r_n}!\langle\lambda\widetilde{z}_n.}^{n=|\tilde{r}|}\,\mathcal{V}_{\tilde{x}}(V)\,(\widetilde{z}_1,\ldots,\widetilde{z}_n,\widetilde{m})\rangle\,\rangle\rangle$$

Let us first discuss how names in $(\widetilde{r}, u_i)$ are decomposed using types. Letting $|\widetilde{r}| = n$ and $i \in \{1,\ldots,n\}$, for each $r_i \in \widetilde{r}$ (with $r_i : S_i$) we generate a sequence $\widetilde{z}_i = (z^i_1,\ldots,z^i_{|\mathcal{R}^\star(S_i)|})$ as in the output case. We decompose name u_i (with $u_i : C$) as $\widetilde{m} = (u_i,\ldots,u_{i+|\mathcal{G}(C)|-1})$.

The decomposition first receives a context $\widetilde{x}$ for value V: we break down V with $\widetilde{x}$ as a context since these variables need to be propagated to the abstracted process. Subsequently, an output on c^{r_1} sends a value containing n abstractions that occur nested within output prefixes—this is similar to the mechanism for partial instantiation shown in Example 4.2.2. For each $j \in \{1,\ldots,n-1\}$, each abstraction binds $\widetilde{z}_j$ and sends the next abstraction along $c^{r_{j+1}}$. The innermost abstraction abstracts over $\widetilde{z}_n$ and encloses the process $\mathcal{V}_{\tilde{x}}(V)\,(\widetilde{z}_1,\ldots,\widetilde{z}_n,\widetilde{m})$, which effectively mimics the application. This abstraction nesting binds all variables $\widetilde{z}_i$, the decompositions of all tail-recursive names ($\widetilde{r}$).

The breakdown of a value application of the form $y\,(\widetilde{r}, u_i)$ results into the following specific case:

$$\mathcal{B}^{k}_{\tilde{x}}(y\,(\widetilde{r}, u_i)) = c_k?(\widetilde{x}).\,\overbrace{c^{r_1}!\langle\lambda\widetilde{z}_1.c^{r_2}!\langle\lambda\widetilde{z}_2.\cdots.c^{r_n}!\langle\lambda\widetilde{z}_n.}^{n=|\tilde{r}|}\,y\,(\widetilde{z}_1,\ldots,\widetilde{z}_n,\widetilde{m})\rangle\,\rangle\rangle$$

Restriction: The decomposition of $(\nu\, s : C)\, P'$ depends on C:

- If $\neg\mathtt{tr}(C)$ then

$$\mathcal{B}^k_{\tilde{x}}((\nu\, s : C)\, P') = (\nu\, \tilde{s} : \mathcal{G}(C))\ \mathcal{B}^k_{\tilde{x}}(P'\sigma)$$

  By construction, $\tilde{x} = \mathtt{fv}(P')$. Similarly as in the decomposition of u_i into $\widetilde{m}$ discussed above, we use the type C of s to obtain the tuple $\tilde{s}$ of length $|\mathcal{G}(C)|$. We initialize the index of s in P' by applying the substitution σ. This substitution depends on C: if it is a shared type then $\sigma = \{^{s_1}/_{s}\}$; otherwise, if C is a session type, then $\sigma = \{^{s_1 \overline{s_1}}/_{s\overline{s}}\}$.

- Otherwise, if $\mathtt{tr}(C)$ then we have:

$$\mathcal{B}^k_{\tilde{x}}((\nu\, s : C)\, P') = (\nu\, \tilde{s} : \mathcal{R}(S))\, (\nu\, c^s)\, c^s?(x).x\, \tilde{s} \mid (\nu\, c^{\overline{s}})\, c^{\overline{s}}?(x).x\, \tilde{\overline{s}} \mid \mathcal{B}^k_{\tilde{x}}(P')$$

  We decompose s into $\tilde{s} = (s_1, \ldots, s_{|\mathcal{G}(S)|})$ and $\overline{s}$ into $\tilde{\overline{s}} = (\overline{s_1}, \ldots, \overline{s_{|\mathcal{G}(S)|}})$. Notice that as $\mathtt{tr}(C)$ we have $C \equiv \mu\mathtt{t}.S$, therefore $\mathcal{G}(C) = \mathcal{R}(S)$. The breakdown introduces two servers in parallel with the breakdown of P'; they provide names for s and $\overline{s}$ along c^s and $c^{\overline{s}}$, respectively. The server on c^s (resp. $c^{\overline{s}}$) receives a value and applies it to the sequence $\tilde{s}$ (resp. $\tilde{\overline{s}}$). We restrict over $\tilde{s}$ and propagators c^s and $c^{\overline{s}}$.

Composition: The breakdown of a process $Q \mid R$ is as follows:

$$\mathcal{B}^k_{\tilde{x}}(Q \mid R) = c_k?(\tilde{x}).\overline{c_{k+1}}!\langle\tilde{y}\rangle.\overline{c_{k+l+1}}!\langle\tilde{w}\rangle \mid \mathcal{B}^{k+1}_{\tilde{y}}(Q) \mid \mathcal{B}^{k+l+1}_{\tilde{w}}(R)$$

A control trio triggers the breakdowns of Q and R; it does not mimic any action of the source process. The tuple $\tilde{y} \subseteq \tilde{x}$ (resp. $\tilde{w} \subseteq \tilde{x}$) collects the free variables in Q (resp. R). To avoid name conflicts, the trigger for the breakdown of R is $\overline{c_{k+l+1}}$, with $l = \lfloor Q \rfloor$.

Inaction: To breakdown $\mathbf{0}$, we define a degenerate trio with only one input prefix that receives a context that by construction will always be empty (i.e., $\tilde{x} = \epsilon$, cf. Remark 4):

$$\mathcal{B}^k_{\tilde{x}}(\mathbf{0}) = c_k?().\mathbf{0}$$

Value: For simplicity, let us consider values of the form $V = \lambda(\tilde{y}, z) : (\tilde{S}, C)^{\leadsto}.\, P$, where $\mathtt{tr}(y_i)$ holds for every $y_i \in \tilde{y}$ and $\neg\mathtt{tr}(z)$, and $\leadsto \in \{\multimap, \rightarrow\}$. The general case is defined similarly. We have:

$$\begin{aligned}
\mathcal{V}_{\tilde{x}}(\lambda(\tilde{y}, z) : (\tilde{S}, C)^{\leadsto}.\, P) &= \lambda(\widetilde{y^1}, \ldots, \widetilde{y^n}, \tilde{z}) : (\widetilde{M})^{\leadsto}.\, N \quad \text{where:} \\
\widetilde{M} &= \mathcal{G}(S_1), \ldots, \mathcal{G}(S_n), \mathcal{G}(C) \\
N &= (\nu\, \tilde{c})\, (\nu\, \tilde{c}_r) \prod_{i \in |\tilde{y}|} c^{y_i}?(x).x\, \widetilde{y^i} \mid \overline{c_1}!\langle\tilde{x}\rangle \mid \mathcal{B}^1_{\tilde{x}}(P\{^{z_1}/_{z}\})
\end{aligned}$$

Every y_i (with $y_i : S_i$) is decomposed into $\widetilde{y^i} = (y_1, \ldots, y_{|\mathcal{G}(S_i)|})$. We use C to decompose z into $\tilde{z}$. We abstract over $\widetilde{y^1}, \ldots, \widetilde{y^n}, \tilde{z}$; the body of the abstraction (i.e. N) is the composition of recursive names propagators, the control trio, and the breakdown of P, with name index initialized with the substitution $\{^{z_1}/_{z}\}$. For every $y_i \in \tilde{y}$ there is a server $c^{y_i}?(x).x\, \widetilde{y^i}$ as a subprocess in the abstracted composition—the rationale for these servers is as in previous cases. We restrict the propagators $\tilde{c} = (c_1, \ldots, c_{\lfloor P \rfloor})$: this enables us to type the value in a shared environment when $\leadsto = \rightarrow$. Also, we restrict special propagator names $\tilde{c}_r = \bigcup_{r \in \tilde{v}} c^r$.

4.3.3 The Decomposition by Example

We illustrate the decompositions by means of several examples.

Decomposing Processes with Non-Recursive Names

Example 4.3.7. Consider process $P = (\nu\, u)\,(Q \mid R)$ whose body implements end-points of channel u with session type $S = ?(U);?(\mathsf{bool});\mathtt{end}$, with $U = (?(\mathsf{bool});\mathtt{end}) \multimap \diamond$, where:

$$\begin{aligned} Q &= u?(x).\, \overbrace{u?(y).(\nu\, s)\,(x\,\overline{s} \mid s!\langle y\rangle)}^{Q'} \\ R &= \overline{u}!\langle V\rangle.\overline{u}!\langle \mathsf{true}\rangle.\mathbf{0} \\ V &= \lambda z.\, z?(b).\mathbf{0} \end{aligned}$$

The process P reduces as follows:

$$\begin{aligned} P &\longrightarrow (\nu\, u)\ \big(u?(y).(\nu\, s)\ (V\,\overline{s} \mid s!\langle y\rangle) \mid \overline{u}!\langle \mathsf{true}\rangle.\mathbf{0}\big) \longrightarrow (\nu\, s)\ (V\,\overline{s} \mid s!\langle \mathsf{true}\rangle) \\ &\longrightarrow (\nu\, s)\ (\overline{s}?(b).\mathbf{0} \mid s!\langle \mathsf{true}\rangle) = P' \end{aligned}$$

By Definition 4.3.9 we have that the decomposition of P is as follows:

$$\mathcal{D}(P) = (\nu\, c_1, \ldots, c_{10})\,(\overline{c_1}!\langle\rangle \mid \mathcal{B}^1_\epsilon(P\sigma))$$

where $\sigma = \{{}^{u_1\overline{u}_1}/_{u\overline{u}}\}$. We have:

$$\mathcal{B}^1_\epsilon(P\sigma) = (\nu\, u_1, u_2)\, c_1?().\overline{c_2}!\langle\rangle.\overline{c_3}!\langle\rangle \mid \mathcal{B}^2_\epsilon(Q\sigma) \mid \mathcal{B}^8_\epsilon(R\sigma)$$

The breakdowns of sub-processes Q and R are as follows:

$$\begin{aligned} \mathcal{B}^2_\epsilon(Q\sigma) &= c_2?().u_1?(x).\overline{c_3}!\langle x\rangle \mid \mathcal{B}^3_\epsilon(Q'\sigma') \\ \mathcal{B}^3_x(Q'\sigma') &= c_3?(x).u_2?(y).\overline{c_4}!\langle x, y\rangle \mid \mathcal{B}^4((\nu\, s)\ (x\,\overline{s} \mid s!\langle y\rangle)) \\ \mathcal{B}^4_{x,y}((\nu\, s)\ (x\,\overline{s} \mid s!\langle y\rangle)) &= (\nu\, s_1)\ (c_4?(x,y).\overline{c_5}!\langle x\rangle.\overline{c_6}!\langle y\rangle \mid c_5?(x).x\,\overline{s}_1 \mid c_6?(y).s_1!\langle y\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0}) \\ \mathcal{B}^8_\epsilon(R\sigma) &= c_8?().\overline{u}_1!\langle \mathcal{V}_\epsilon(V)\rangle.\overline{c_9}!\langle\rangle \mid \mathcal{B}^9_\epsilon(u_2!\langle \mathsf{true}\rangle.\mathbf{0}) \\ \mathcal{B}^9_\epsilon(u_2!\langle \mathsf{true}\rangle.\mathbf{0}) &= c_9?().\overline{u}_2!\langle \mathsf{true}\rangle.\overline{c_{10}}!\langle\rangle \mid c_{10}?().\mathbf{0} \\ \mathcal{V}_\epsilon(V) &= \lambda z_1.\,((\nu\, c^V_1, c^V_2)\ \overline{c^V_1}!\langle\rangle \mid c^V_1?().z_1?(b).\overline{c^V_2}!\langle\rangle \mid c^V_2?().\mathbf{0}) \end{aligned}$$

where $\sigma' = \{{}^{u_2\overline{u}_2}/_{u\overline{u}}\}$. By $\mathcal{G}(-)$ from Definition 4.3.3 we decompose S into M_1 and M_2 given as follows:

$$\begin{aligned} M_1 &= ?(\mathcal{G}(U));\mathtt{end} = ?(U);\mathtt{end} \\ M_2 &= ?(\mathsf{bool});\mathtt{end} \end{aligned}$$

Above we may notice that $\mathcal{G}(U) = U$. We remark that $\mathcal{D}(P)$ accordingly implements indexed names u_1, u_2 typed with M_1, M_2, respectively.

Let us inspect the reductions of $\mathcal{D}(P)$. First, there are three synchronizations on c_1, c_2, and c_8:

$$\begin{aligned} \mathcal{D}(P) &\longrightarrow (\nu\, c_2, \ldots, c_{10})\,(\nu\, u_1, u_2)\,\overline{c_2}!\langle\rangle.\overline{c_8}!\langle\rangle \mid \mathcal{B}^2_\epsilon(Q\sigma) \mid \mathcal{B}^8_\epsilon(R\sigma) \\ &\longrightarrow^2 (\nu\, c_3, \ldots, c_7, c_9, c_{10})\ u_1?(x).\,\overline{c_3}!\langle x\rangle \mid \mathcal{B}^3_\epsilon(Q'\sigma') \\ &\qquad \mid\ \overline{u}_1!\langle \mathcal{V}_\epsilon(V)\rangle.\,\overline{c_9}!\langle\rangle \quad \mid \mathcal{B}^9_\epsilon(u_2!\langle \mathsf{true}\rangle.\mathbf{0}) = D^1 \end{aligned}$$

After reductions on propagators, D^1 is able to mimic the original synchronization on channel u (highlighted above). It is followed by two administrative reductions on c_3 and c_9:

$$\begin{aligned}
D^1 \longrightarrow\ & (\nu\, c_3, \ldots, c_7, c_9, c_{10})\, \overline{c_3}!\langle \mathcal{V}_\epsilon(V) \rangle \mid c_3?(x).u_2?(y).\overline{c_4}!\langle x, y\rangle \mid \mathcal{B}^4((\nu\, s)\,(x\,\overline{s} \mid s!\langle y\rangle)) \mid \\
& \overline{c_9}!\langle\rangle \mid c_9?().\overline{u}_2!\langle \mathsf{true}\rangle.\overline{c_{10}}!\langle\rangle \mid c_{10}?().\mathbf{0} \\
\longrightarrow^2\ & (\nu\, c_4, \ldots, c_7, c_{10})\ u_2?(y).\ \overline{c_4}!\langle \mathcal{V}_\epsilon(V), y\rangle \mid \\
& (\nu\, s_1)\,(c_4?(x,y).\overline{c_5}!\langle x\rangle.\overline{c_6}!\langle y\rangle \mid c_5?(x).x\,\overline{s}_1 \mid c_6?(y).s_1!\langle y\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0}) \mid \\
& \overline{u}_2!\langle \mathsf{true}\rangle.\ \overline{c_{10}}!\langle\rangle \mid c_{10}?().\mathbf{0} = D^2
\end{aligned}$$

Similarly, D^2 can mimic the next synchronization of the original process on name u_2. Following up on that, syncronization on c_{10} takes place:

$$\begin{aligned}
D^2 \longrightarrow^2\ & (\nu\, c_4, \ldots, c_7)\, \overline{c_4}!\langle \mathcal{V}_\epsilon(V), \mathsf{true}\rangle \mid \\
& (\nu\, s_1)\,(c_4?(x,y).\overline{c_5}!\langle x\rangle.\overline{c_6}!\langle y\rangle \mid c_5?(x).x\,\overline{s}_1 \mid c_6?(y).s_1!\langle y\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0}) = D^3
\end{aligned}$$

Now, we can see that the next three reductions on c_4, c_5, and c_6 appropriately propagate values $\mathcal{V}_\epsilon(V)$ and true to the breakdown of sub-processes. Subsequently, value $\mathcal{V}_\epsilon(V)$ is applied to name $\overline{s}_1$:

$$\begin{aligned}
D^3 \longrightarrow\ & (\nu\, c_5, \ldots, c_7)\,(\nu\, s_1)\,(\overline{c_5}!\langle \mathcal{V}_\epsilon(V)\rangle.\overline{c_6}!\langle \mathsf{true}\rangle \mid c_5?(x).x\,\overline{s}_1 \mid c_6?(y).s_1!\langle y\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0}) \\
\longrightarrow^2\ & (\nu\, c_7)\,(\nu\, s_1)\,(\mathcal{V}_\epsilon(V)\,\overline{s}_1 \mid s_1!\langle \mathsf{true}\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0}) \\
\longrightarrow\ & (\nu\, c_7)\,(\nu\, s_1)\,((\nu\, c_1^V, c_2^V)\, \overline{c_1^V}!\langle\rangle \mid c_1^V?().s_1?(b).\overline{c_2^V}!\langle\rangle \mid c_2^V?().\mathbf{0}) \mid s_1!\langle \mathsf{true}\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0} = D^4
\end{aligned}$$

Finally, after syncronization on c_1^V we reach the process D^5 that is clearly able to simulate P', and its internal communication on the channel s:

$$D^4 \longrightarrow (\nu\, c_7)\,(\nu\, s_1)\,((\nu\, c_2^V)\, s_1?(b).\overline{c_2^V}!\langle\rangle \mid c_2^V?().\mathbf{0}) \mid s_1!\langle \mathsf{true}\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0} = D^5$$

$\triangleleft$

Example 4.3.8 (Breaking Down Name-Passing)**.** Consider the following process P, in which a channel m is passed, through which a boolean value is sent back:

$$P = (\nu\, u)\,(u!\langle \ulcorner m \urcorner\rangle.\overline{m}?(b) \mid \overline{u}?(\ulcorner x \urcorner).x!\langle \mathsf{true}\rangle)$$

After expanding the syntactic sugar of name-passing, we get a process $P = (\nu\, u)\,(Q \mid R)$, where

$$\begin{aligned}
Q &= u!\langle V\rangle.\overline{m}?(y).(\nu\, s)\,(y\, s \mid \overline{s}!\langle \lambda b.\, \mathbf{0}\rangle) & V &= \lambda z.\, z?(x).(x\, m) \\
R &= \overline{u}?(y).(\nu\, s)\,(y\, s \mid \overline{s}!\langle W\rangle) & W &= \lambda x.\, x!\langle W'\rangle \text{ with } W' = \lambda z.\, z?(x).(x\, \mathsf{true})
\end{aligned}$$

Note that to mimic the name-passing synchronization, we require exactly four reduction steps:

$$P \longrightarrow^4 [\![\overline{m}?(b) \mid m!\langle \mathsf{true}\rangle]\!] \longrightarrow^4 \mathbf{0} \tag{4.18}$$

We will now investigate the decomposition of P and its reduction chain. First, we use Definition 4.3.6 to compute $\wr Q \int = 6$, and similarly, $\wr R \int = 5$. Therefore, $\wr P \int = 12$. Following Definition 4.3.9, we see that $\sigma = \{m_1\overline{m_1}/m\overline{m}\}$, which we silently apply. Taking $k = 1$, the breakdown of P and its subprocesses is shown in Table 4.2.

In Table 4.2 we have omitted substitutions that have no effect and trailing $\mathbf{0}$s. The first interesting action appears after synchronizations on c_1, c_2, and c_8. At that point, the process

$$
\begin{aligned}
\mathcal{D}(P) &= (\nu\, c_1, \ldots, c_{12}) \Big(\overline{c_1}!\langle\rangle \mid (\nu\, u_1)\, (c_1?().\overline{c_2}!\langle\rangle.\overline{c_8}!\langle\rangle \mid \mathcal{B}^{2}_{\epsilon}(Q) \mid \mathcal{B}^{8}_{\epsilon}(R))\Big) \\
\mathcal{B}^{2}_{\epsilon}(Q) &= c_2?().u_1!\langle \mathcal{V}_{\epsilon}(V) \rangle.\overline{c_3}!\langle\rangle \mid c_3?().\overline{m_1}?(y).\overline{c_4}!\langle y\rangle \mid \\
&\qquad (\nu\, s_1)\, (c_4?(y).\overline{c_5}!\langle y\rangle.\overline{c_6}!\langle\rangle \mid c_5?(y).(y\, s_1) \mid c_6?().\overline{s_1}!\langle \mathcal{V}_{\epsilon}(\lambda b.\, \mathbf{0})\rangle.\overline{c_7}!\langle\rangle \mid c_7?()) \\
\mathcal{B}^{8}_{\epsilon}(R) &= c_8?().\overline{u_1}?(y).\overline{c_9}!\langle y\rangle \mid \\
&\qquad (\nu\, s_1)\, (c_9?(y).\overline{c_{10}}!\langle y\rangle.\overline{c_{11}}!\langle\rangle \mid c_{10}?(y).(y\, s_1) \mid c_{11}?().\overline{s_1}!\langle \mathcal{V}_{\epsilon}(W)\rangle.\overline{c_{12}}!\langle\rangle \mid c_{12}?()) \\
\mathcal{V}_{\epsilon}(V) &= \lambda z_1.\, (\nu\, c^V_1, c^V_2)\, (\overline{c^V_1}!\langle\rangle \mid c^V_1?().z_1?(x).\overline{c^V_2}!\langle x\rangle \mid c^V_2?(x).(x\, m_1)) \\
\mathcal{V}_{\epsilon}(\lambda b.\, \mathbf{0}) &= \lambda b_1.\, (\nu\, c^b_1)\, (\overline{c^b_1}!\langle\rangle \mid c^b_1?()) \\
\mathcal{V}_{\epsilon}(W) &= \lambda x_1.\, (\nu\, c^W_1, c^W_2)\, (\overline{c^W_1}!\langle\rangle \mid c^W_1?().x_1!\langle \mathcal{V}_{\epsilon}(W')\rangle.\overline{c^W_2}!\langle\rangle \mid c^W_2?()) \\
\mathcal{V}_{\epsilon}(W') &= \lambda z_1.\, (\nu\, c^{W'}_1, c^{W'}_2)\, (\overline{c^{W'}_1}!\langle\rangle \mid c^{W'}_1?().z_1?(x).\overline{c^{W'}_2}!\langle x\rangle \mid c^{W'}_2?(x).(x\, \mathsf{true}))
\end{aligned}
$$

Table 4.2: The decomposition on processes discussed in Example 4.3.8.

will be ready to mimic the first action that is performed by P, i.e., u_1 will send $\mathcal{V}_{\epsilon}(V)$, the breakdown of V, from the breakdown of Q to the breakdown of R. Next, c_9 and c_{10} will synchronize, and $\mathcal{V}_{\epsilon}(V)$ is passed further along, until s_1 is ready to be applied to it in the breakdown of R. At this point, we know that $P \longrightarrow^{7} (\nu\, \widetilde{c})\, P'$, where $\widetilde{c} = (c_3, \ldots, c_{12})$, and

$$
\begin{aligned}
P' = {}& \overline{c_3}!\langle\rangle \mid c_3?().\overline{m_1}?(y).\overline{c_4}!\langle y\rangle \\
&\mid (\nu\, s_1)\, (c_4?(y).\overline{c_5}!\langle y\rangle.\overline{c_6}!\langle\rangle \mid c_5?(y).y\, s_1 \mid c_6?().\overline{s_1}!\langle \mathcal{V}_{\epsilon}(\lambda b.\, \mathbf{0})\rangle.\overline{c_7}!\langle\rangle \mid c_7?()) \\
&\mid (\nu\, s_1)\, (\mathcal{V}_{\epsilon}(V)\, s_1 \mid \overline{s_1}!\langle \mathcal{V}_{\epsilon}(W)\rangle.\overline{c_{12}}!\langle\rangle \mid c_{12}?())
\end{aligned}
$$

After s_1 is applied, the trio guarded by c_3 will be activated, where z_1 has been substituted by s_1. Then $\overline{s_1}$ and s_1 will synchronize, and the breakdown of W is passed along. Then c_4 and c_{19} synchronize, and now m_1 is ready to be applied to $\mathcal{V}_{\epsilon}(W)$, which was the input for c_4 in the breakdown of V. After this application, c_3 and c^W_1 can synchronize with their duals, and we know that $(\nu\, \widetilde{c})\, P' \longrightarrow^{8} (\nu\, \widetilde{c}')\, P''$, where $\widetilde{c}' = (c_4, \ldots, c_7, c^{W'}_2)$, and

$$
\begin{aligned}
P'' = {}& \overline{m_1}?(y).\overline{c_4}!\langle y\rangle \mid m_1!\langle \mathcal{V}_{\epsilon}(W')\rangle.\overline{c^{W'}_2}!\langle\rangle \mid c^{W'}_2?() \\
&\mid (\nu\, s_1)\, (c_4?(y).\overline{c_5}!\langle y\rangle.\overline{c_6}!\langle\rangle \mid c_5?(y).y\, s_1 \mid c_6?().\overline{s_1}!\langle \mathcal{V}_{\epsilon}(\lambda b.\, \mathbf{0})\rangle.\overline{c_7}!\langle\rangle \mid c_7?())
\end{aligned}
$$

Remarkably, P'' is standing by to mimic the encoded exchange of value true. Indeed, the decomposition of the four-step reduced process in (4.18) will reduce in three steps to a process that is equal (up to $\equiv_\alpha$) to the process we obtained here. This strongly suggests a tight operational correspondence between a process and its decomposition, which we will explore in Section 4.4. $\lhd$

Decomposing Processes with Recursive Names

Next, we illustrate the decomposition of processes involving names with tail-recursive types. Recall process R_1, which we used in Section 4.3.1 to motivate the need for recursive propagators:

$$R_1 = r?(z).r!\langle -z\rangle.r?(z).r!\langle z\rangle.V\, r$$

Two following examples illustrate the low-level workings of the propagation mechanism of the decomposition in Figure 4.3. The first example illustrates how the propagation of recursive names works in the case of input and output actions on names with recursive types (the "first part" of R_1). The second example shows how an application where a value is applied to a tuple of names with recursive names is broken down (the "second part" of R_1).

Example 4.3.9 (Decomposing Processes with Recursive Names (I))**.** Let $P = r?(x).r!\langle x\rangle.P'$ be a process where r has type $S = \mu\mathsf{t}.?(\mathsf{int});!\langle\mathsf{int}\rangle;\mathsf{t}$ and $r \in \mathtt{fn}(P')$. By Definition 4.3.9 we have:

$$\mathcal{D}(P) = (\nu\,\tilde{c})\,(\nu\,c^r)\,(\overline{c_1}!\langle\rangle \mid \mathcal{B}^1_\epsilon(P) \mid c^r?(x).x\,(r_1, r_2))$$

where $\tilde{c} = (c_1, \dots, c_{|P|})$. The control trio in the parallel composition provides a decomposition of r on name c^r, which is *shared*. The decomposition $\mathcal{B}^1_\epsilon(P)$ is defined as follows:

$$\begin{aligned}\mathcal{B}^1_\epsilon(P) &= \overline{c_1}!\langle\rangle \mid c_1?().c^r!\langle N_1\rangle \mid c_2?(y).c^r!\langle N_2\rangle \mid \mathcal{B}^3_\epsilon(P'))\\ N_1 &= \lambda(z_1, z_2).\, z_1?(x).c^r?(x).x\,(z_1, z_2)\\ N_2 &= \lambda(z_1, z_2).\, z_2!\langle x\rangle.\overline{c_3}!\langle\rangle.c^r?(x).x\,(z_1, z_2)\end{aligned}$$

Each trio in $\mathcal{B}^1_\epsilon(P)$ that mimics some action on r requests the sequence $\tilde{r}$ from the server on c^r. We can see that this request is realized by a higher-order communication: trios send abstractions (N_1 and N_2) to the server; these abstractions contain further actions of trios and it will be applied to the sequence $\tilde{r}$. Hence, the formal arguments for these values are meant to correspond to $\tilde{r}$.

After two reductions (the trio activation on c_1 and the communication on c^r), we have:

$$\mathcal{D}(P) \longrightarrow^2 (\nu\,c_2, \dots, c_{\lfloor P\rfloor})\; r_1?(x).\overline{c_2}!\langle x\rangle.c^r?(x).x\,(r_1, r_2) \mid c_2?(y).c^r!\langle N_2\rangle \mid \mathcal{B}^3_\epsilon(P') = P_1$$

By synchronizing with the top-level server on c^r, the bound names in N_1 are instantiated with r_1, r_2. Now, the first trio in P_1 is able to mimic the action on r_1 that is followed by the activation of the next trio on c_2. Then, the server on c^r gets reinstantiated making names r_1, r_2 available for future trios. The break down of the output action follows the same pattern. $\triangleleft$

Example 4.3.10 (Decomposing Processes with Recursive Names (II))**.** Let $S = \mu\mathsf{t}.?(\mathsf{int});!\langle\mathsf{int}\rangle;\mathsf{t}$ and $T = \mu\mathsf{t}.?(\mathsf{bool});!\langle\mathsf{bool}\rangle;\mathsf{t}$, and define $Q = V\,(u, v)$ as a process where $u : S$ and $v : T$, where V is some value of type $(S, T) \rightarrow \diamond$. By Definition 4.3.9, the decomposition of Q is as in the previous example, except that now there are two servers, one for u and one for v:

$$\begin{aligned}\mathcal{D}(Q) &= (\nu\,c_1\tilde{c})\,(\nu\,c^u c^v)\,(c^u?(x).x\,(u_1, u_2) \mid c^v?(x).x\,(v_1, v_2) \mid \overline{c_1}!\langle\rangle \mid \mathcal{B}^1_\epsilon(Q))\\ \mathcal{B}^1_\epsilon(Q) &= c_1?().c^u!\langle\lambda(x_1, x_2).\, c^v!\langle\lambda(y_1, y_2).\, \mathcal{V}_\epsilon(V)\,(x_1, x_2, y_1, y_2)\rangle\rangle\end{aligned}$$

with $\tilde{c} = (c_2, \dots, c_{\lfloor Q\rfloor})$. Process Q is broken down in such a way that it communicates with both servers to collect $\tilde{u}$ and $\tilde{v}$. To this end, $\mathcal{B}^1_\epsilon(Q)$ is a process in which abstractions are nested using output prefixes and whose innermost process is an application. After successive communications with multiple servers this innermost application will have collected all names in $\tilde{u}$ and $\tilde{v}$.

Observe that we use two nested outputs, one for each name with recursive types in Q. We now look at the reductions of $\mathcal{D}(Q)$ to analyze how the communication of nested abstractions allows us to collect all name sequences needed. After the first reduction along c_1 we have:

$$\begin{aligned}\mathcal{D}(Q) \longrightarrow &(\nu\,\tilde{c})\,(\nu\,c^u c^v)\,(c^u?(x).x\,(u_1, u_2) \mid c^v?(x).x\,(v_1, v_2) \mid\\ &\qquad c^u!\langle\lambda(x_1, x_2).\, c^v!\langle\lambda(y_1, y_2).\, \mathcal{V}_\epsilon(V)\,(x_1, x_2, y_1, y_2)\rangle\rangle) = R^1\end{aligned}$$

From R^1 we have a synchronization along name c^u:

$$R^1 \longrightarrow (\nu\,\tilde{c})\,(\nu\,c^u c^v)\,((\lambda(x_1, x_2).\, c^v!\langle\lambda(y_1, y_2).\, \mathcal{V}_\epsilon(V)\,(x_1, x_2, y_1, y_2)\rangle)\,(u_1, u_2) \mid c^v?(x).x\,(v_1, v_2)) = R^2$$

Upon receiving the value, the server applies it to (u_1, u_2), thus obtaining the following process:

$$R^2 \longrightarrow (\nu\,\tilde{c})\,(\nu\,c^u c^v)\,(c^v!\langle \lambda(y_1,y_2).\,\mathcal{V}_{\epsilon}(V)\,(u_1,u_2,y_1,y_2)\rangle \mid c^v?(x).x\,(v_1,v_2)) = R^3$$

Up to here, we have partially instantiated name variables of a value with the sequence $\tilde{u}$. Next, the first trio in R^3 can communicate with the server on name c^v:

$$\begin{aligned} R^3 &\longrightarrow (\nu\,\tilde{c})\,(\nu\,c^u c^v)\,(\lambda(y_1,y_2).\,\mathcal{V}_{\epsilon}(V)\,(u_1,u_2,y_1,y_2)\,(v_1,v_2)) \\ &\longrightarrow (\nu\,\tilde{c})\,(\nu\,c^u c^v)\,(\mathcal{V}_{\epsilon}(V)\,(u_1,u_2,v_1,v_2)) \end{aligned}$$

This completes the instantiation of name variables with appropriate sequences of names with recursive types. At this point, $\mathcal{D}(Q)$ can proceed to mimic the application in Q.

$\lhd$

Example 4.3.11 (Breakdown of Recursion Encoding)**.** We recall process $[\![P]\!]$ from Example 4.3.4:

$$\begin{aligned} [\![P]\!] &= a?(m).a!\langle m\rangle.(\nu\,s)\,(V\,(a,s) \mid \overline{s}!\langle V\rangle) \\ V &= \lambda(x_a,y_1).\,y_1?(z_x).x_a?(m).x_a!\langle m\rangle.(\nu\,s)\,(z_x\,(x_a,s) \mid \overline{s}!\langle z_x\rangle.\mathbf{0}) \end{aligned}$$

Here, bound name s is typed with S, from Example 4.3.3, defined as:

$$S = \mu\mathsf{t}.?((?(\mathsf{str});!\langle\mathsf{str}\rangle;\mathsf{end},\mathsf{t})\rightarrow\diamond);\mathsf{end}$$

We now analyze $\mathcal{D}([\![P]\!])$ and its reduction chain. By Definition 4.3.6, we have $\wr[\![P]\!]\int = 7$. Then, we choose $k = 1$ and observe that $\sigma = \{{}^{a_1\overline{a_1}}/_{a\overline{a}}\}$. Following Definition 4.3.9, we get:

$$\begin{aligned} \mathcal{D}([\![P]\!]) &= (\nu\,c_1,\dots,c_7)\,(\nu\,c^a)\,(c^a?(x).x\,(a_1,a_2) \mid \overline{c_1}!\langle\rangle \mid \mathcal{B}^1_{\epsilon}([\![P]\!]\sigma)) \\ \mathcal{B}^1_{\epsilon}([\![P]\!]) &= c_1?().c^a!\langle\lambda(z_1,z_2).\,z_1?(m).\overline{c_2}!\langle m\rangle.c^a?(x).x\,(z_1,z_2)\rangle \\ &\quad\mid c_2?(m).c^a!\langle\lambda(z_1,z_2).\,z_2!\langle m\rangle.\overline{c_3}!\langle\rangle.c^a?(x).x\,(z_1,z_2)\rangle \\ &\quad\mid (\nu\,s_1)\,(c_3?().\overline{c_4}!\langle\rangle.\overline{c_5}!\langle\rangle \mid c_4?().\overline{c^a}!\langle\lambda(z_1,z_2).\,\mathcal{V}_{\epsilon}(V)\,(z_1,z_2,s_1)\rangle \\ &\qquad\mid c_5?().\overline{s}_1!\langle\mathcal{V}_{\epsilon}(V)\rangle.\overline{c_7}!\langle\rangle \mid c_7?()) \end{aligned}$$

In accordance with Example 4.3.3, the type of s_1 in the decomposed process is

$$M = \mu\mathsf{t}.?((?(\mathsf{str}),!\langle\mathsf{str}\rangle,\mathsf{t})\rightarrow\diamond).$$

The decomposition relies twice on $\mathcal{V}_{\epsilon}(V)$, the breakdown of value V, which we give below. For this, we observe that V is an abstraction of a process Q with $|Q| = 7$. We also α-convert the process abstracted in $\mathcal{V}_{\epsilon}(V)$ renaming bound propagators $c_1,\dots,c_7$ to $c^V_1,\dots,c^V_7$ to avoid name clashes.

$$\begin{aligned} \mathcal{V}_{\epsilon}(V) &= \lambda(x_{a_1},x_{a_2},y_1).\,(\nu\,c^V_1,\dots,c^V_7)\,(\overline{c^V_1}!\langle\rangle \mid \mathcal{B}^1_{\epsilon}(Q)\{c^V_1,\dots,c^V_7/c_1,\dots,c_7\} \mid c^{x_a}?(x).x\,(x_{a_1},x_{a_2})) \\ \mathcal{B}^1_{\epsilon}(Q) &= c_1?().y_1?(z_x).\overline{c_2}!\langle z_x\rangle \\ &\quad\mid c_2?(z_x).c^a!\langle\lambda(z_1,z_2).\,z_1?(m).\overline{c_3}!\langle z_x,m\rangle.c^a?(x).x\,(z_1,z_2)\rangle \\ &\quad\mid c_3?(z_x).c^a!\langle\lambda(z_1,z_2).\,z_2!\langle m\rangle.\overline{c_4}!\langle z_x\rangle.c^a?(x).x\,(z_1,z_2)\rangle \\ &\quad\mid (\nu\,s_1)\,(c_4?(x_z).\overline{c_5}!\langle z_x\rangle.\overline{c_6}!\langle z_x\rangle \mid \\ &\qquad\qquad c_5?(z_x).c^a!\langle\lambda(z_1,z_2).\,z_x\,(z_1,z_2,s_1)\rangle \mid c_6?(z_x).\overline{s_1}!\langle z_x\rangle.\overline{c_7}!\langle\rangle \mid c_7?()) \end{aligned}$$

We follow the reduction chain on $\mathcal{D}([\![P]\!])$ until it is ready to mimic the first action with channel a, which is an input. First, c_1 will synchronize, after which c^a sends the abstraction to which then (a_1, a_2) is applied. We obtain $\mathcal{D}(\llbracket P \rrbracket) \longrightarrow^3 (\nu\, c_2, \dots, c_7, c^a)\, P'$, where

$$\begin{aligned} P' = {} & a_1?(m).\overline{c_2}!\langle m\rangle.c^a?(x).x\,(a_1, a_2) \\ & \mid c_2?(m).c^a!\langle \lambda(z_1, z_2).\, z_2!\langle m\rangle.\overline{c_3}!\langle\rangle.c^a?(x).x\,(z_1, z_2)\rangle \\ & \mid (\nu\, s_1)\,(c_3?().\overline{c_4}!\langle\rangle.\overline{c_5}!\langle\rangle. \mid c_4?().\overline{c^a}!\langle \lambda(z_1, z_2).\, \mathcal{V}_\epsilon(V)\,(z_1, z_2, s_1)\rangle \mid \\ & \qquad c_5?().\overline{s}_1!\langle \mathcal{V}_\epsilon(V)\rangle.\overline{c_7}!\langle\rangle \mid c_7?()) \end{aligned}$$

Note that this process is awaiting an input on channel a_1, after which c_2 can synchronize with its dual. At that point, c^a is ready to receive another abstraction that mimics an input on a_1. This strongly suggests a tight operational correspondence between a process P and its decomposition in the case where P performs higher-order recursion. $\lhd$

4.3.4 Static Correctness

Having presented and illustrated our decomposition, we may now state its technical results. Given an environment $\Delta = \Delta_1, \Delta_2$, below we write $\Delta_1 \circ \Delta_2$ to indicate the split of Δ into a Δ_1 containing non-recursive names and a Δ_2 containing recursive names.

We extend the decomposition function $\mathcal{G}(-)$ to typing environments in the obvious way. We rely on the following notation. Given a tuple of names $\widetilde{s} = s_1, \dots, s_n$ and a tuple of (session) types $\widetilde{S} = S_1, \dots, S_n$ of the same length, we write $\widetilde{s} : \widetilde{S}$ to denote a list of typing assignments $s_1 : S_1, \dots, s_n : S_n$.

Definition 4.3.10 (Decomposition of Environments)**.** Let Γ, Λ, and Δ be typing environments. We define $\mathcal{G}(\Gamma)$, $\mathcal{G}(\Lambda)$, and $\mathcal{G}(\Delta)$ inductively as follows:

$$\begin{aligned} \mathcal{G}(\Delta, u_i : S) &= \mathcal{G}(\Delta), (u_i, \dots, u_{i+|\mathcal{G}(S)|-1}) : \mathcal{G}(S) \\ \mathcal{G}(\Gamma, u_i : \langle U\rangle) &= \mathcal{G}(\Gamma), u_i : \mathcal{G}(\langle U\rangle) \\ \mathcal{G}(\Gamma, x : U) &= \mathcal{G}(\Gamma), x : \mathcal{G}(U) \\ \mathcal{G}(\Lambda, x : U) &= \mathcal{G}(\Lambda), x : \mathcal{G}(U) \\ \mathcal{G}(\emptyset) &= \emptyset \end{aligned}$$

Lemma 4.3.1. *Let P be an indexed* HO *process and V be a value.*

1. *If $\Gamma; \Lambda; \Delta \circ \Delta_\mu \vdash P \triangleright \diamond$ then $\mathcal{G}(\Gamma_1), \Phi; \emptyset; \mathcal{G}(\Delta), \Theta \vdash \mathcal{B}^k_{\widetilde{x}}(P) \triangleright \diamond$, where:*
   - $k > 0$
   - $\widetilde{r} = dom(\Delta_\mu)$
   - $\Phi = \prod_{r \in \widetilde{r}} c^r : \langle \mathcal{R}^\star(\Delta_\mu(r)) {\multimap} \diamond\rangle$
   - $\widetilde{x} = \mathtt{fv}(P)$
   - $\Gamma_1 = \Gamma \setminus \widetilde{x}$
   - $dom(\Theta) = \{c_k, \dots, c_{k+\wr P \wr -1}\} \cup \{\overline{c_{k+1}}, \dots, \overline{c_{k+\wr P \wr -1}}\}$
   - $\Theta(c_k) = ?(U_1, \dots, U_n)$*, where* $(\mathcal{G}(\Gamma), \mathcal{G}(\Lambda))(\widetilde{x}) = (x_1 : U_1, \dots, x_n : U_n)$
   - $\mathsf{balanced}(\Theta)$
2. *If $\Gamma; \Lambda; \Delta \circ \Delta_\mu \vdash V \triangleright \widetilde{T} {\multimap} \diamond$ then $\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta) \vdash \mathcal{V}_{\widetilde{x}}(V) \triangleright \mathcal{G}(\widetilde{T}) {\multimap} \diamond$, where:*
   - $\widetilde{x} = \mathtt{fv}(V)$
   - $\Phi = \prod_{r \in \widetilde{r}} c^r : \langle \mathcal{R}^\star(\Delta_\mu(r)) {\multimap} \diamond\rangle$

Proof. By mutual induction on the structure of P and V. See Appendix A.1.1 for details. $\square$

Using the above lemma we can prove our static correctness result, which explains how our decomposition induces minimal session types.

Theorem 4.3.1 (Static Correctness)**.** *Let P be a closed* HO *process (i.e.* $\mathtt{fv}(P) = \emptyset$*) with* $\widetilde{u} = \mathtt{fn}(P)$*. If* $\Gamma; \emptyset; \Delta \circ \Delta_\mu \vdash P \triangleright \diamond$*, then* $\mathcal{G}(\Gamma\sigma); \emptyset; \mathcal{G}(\Delta\sigma), \mathcal{G}(\Delta_\mu\sigma) \vdash \mathcal{D}(P) \triangleright \diamond$*, where* $\sigma = \{\mathsf{init}(\widetilde{u})/\widetilde{u}\}$.

Proof. Directly from the definitions, using Lemma 4.3.1. See Appendix A.1.2 for details. □

4.4 Dynamic Correctness

In this section, we establish the dynamic correctness of our decomposition, stated in terms of a typed behavioral equivalence. More specifically, we would like to show that any typed process P is equivalent to its decomposition $\mathcal{D}(P)$. But how do we even state it formally? Both P and $\mathcal{D}(P)$ are typed HO processes (as any minimally typed process is also an HO process), so we can consider compare them as HO terms inside the HO type system. The conventional notion of typed equivalence for HO processes is contextual equivalence, which is given a local characterization in terms of *higher-order bisimulations* [40]. In our case, however, contextual equivalence is not the right choice: contextual equivalence applies to processes of the *same type*, whereas the process P and its decomposition $\mathcal{D}(P)$ have different types and typing contexts. Instead of using contextual equivalence, we generalize the notion of higher-order bisimilarity to a notion that we call *MST bisimilarity*, which relates processes of (potentially) different types.

This section is organized as follows. In Section 4.4.1 we recall the notion of higher-order bisimulation, used for characterizing behavioral equivalence in HO, and discuss its limitations for our purposes. We use higher-order bisimulation as a basis to give a formal definition of MST bisimulation in Section 4.4.2, which we will use as a notion of behavioral equivalence for comparing P and $\mathcal{D}(P)$. In order to show that our decomposition is correct, in Section 4.4.3 we exhibit a bisimulation relation $\mathcal{S}$ which relates a process and its decomposition, containing a number of intermediate pairs, working from a motivating example in Section 4.4.3. Finally, in Section 4.4.4 we show that $\mathcal{S}$ is indeed an MST bisimulation.

4.4.1 Behavioral Equivalence in HO and its Limitations

Let us begin by recalling the notion of HO bisimulation, defined in [40] to characterize contextual equivalence of HO processes.

Definition 4.4.1 (Definition 17 in [40])**.** A typed relation $\Re$ is an HO *bisimulation* if for all $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1 \;\Re\; \Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1$,

1) Whenever $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1 \xmapsto{(\nu\, \widetilde{m_1})\, n!\langle V_1\rangle} \Lambda_1'; \Delta_1' \vdash P_2$ then there exist Q_2, Δ_2', and Λ_2' such that $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \overset{(\nu\, \widetilde{m_2})\, n!\langle V_2\rangle}{\Longmapsto} \Lambda_2'; \Delta_2' \vdash Q_2$ where, for a fresh t,

$$\Gamma_1; \Lambda_1; \Delta_1'' \vdash (\nu\, \widetilde{m_1})(P_2 \mid t \hookleftarrow_{\mathtt{H}} V_1) \;\Re\; \Gamma_2; \Lambda_2; \Delta_2'' \vdash (\nu\, \widetilde{m_2})(Q_2 \mid t \hookleftarrow_{\mathtt{H}} V_2)$$

2) Whenever $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1 \overset{\ell}{\mapsto} \Lambda_1'; \Delta_1' \vdash P_2$, with ℓ not an output, then there exist Q_2, Λ_2', and Δ_2' such that $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \overset{\hat{\ell}}{\Longmapsto} \Lambda_2'; \Delta_2' \vdash Q_2$ and $\Gamma_1; \Lambda_1'; \Delta_1' \vdash P_2 \;\Re\; \Gamma_2; \Lambda_2'; \Delta_2' \vdash Q_2$.

3) The symmetric cases of 1, 2.

The largest such bisimulation is called HO *bisimilarity*, denoted by $\approx^{\mathtt{H}}$.

There are two points worth highlighting in this definition. Firstly, the labeled transition system $\overset{\ell}{\mapsto}$ used in the definition of $\approx^{\mathtt{H}}$ is what is called the *refined transition system*, different from the standard labeled transition system for the higher-order π-calculus. The idea behind the refined transition system is that we want to disallow arbitrary inputs $P \xmapsto{x(V)} P'$; having to consider such transitions in the definition of bisimilarity is undesirable, because it involves input of an arbitrary (higher-order) value V, making the definition very much non-local and ensuring that the bisimulations are very large. As it turns out, due to the typed nature of the system, it suffices to consider inputs of the processes of a very particular kind—*characteristic values*, defined based on the type.

Secondly, because the inputs are restricted in the refined LTS, there is some price to pay in the handling of the outputs. If an output action $P_1 \xmapsto{(\nu\,\widetilde{m_1})\,n!\langle V_1\rangle} P_2$ is matched by an output action $Q_1 \overset{(\nu\,\widetilde{m_2})\,n!\langle V_2\rangle}{\Longmapsto} Q_2$, then we need to ensure that that the output processes V_1 and V_2 are somehow related. We have to ensure this in the output clause, because on the receiving end transitions inputing values V_1 or V_2 might not even be considered. To that extent, we package the values V_1 or V_2 in *trigger processes* (denoted $t \hookleftarrow_{\mathtt{H}} V_1$ and $t \hookleftarrow_{\mathtt{H}} V_2$), which are defined based on the typing. We then make them part of the processes that are considered at the "next step" of the bisimulation.

This notion of HO bisimilarity works for processes of the same type. For our case, we need to compare processes of different, but related types. To that extent we make several changes to the definition above. Firstly, during the decomposition a single name x in a source process is decomposed into a sequence of names $x_1, \ldots, x_k$ in the target process. So in the definition of MST bisimilarity we match an action on a name x with an action on an *indexed* name x_i. Secondly, such discrepancy between names might arise in input and output values. This also needs to be considered as part of the definition. For this, we need to accommodate the difference between characteristic values and trigger processes for MST and HO. In the next subsection we work out the details sketched above.

4.4.2 MST Bisimilarity

In this section we define a generalized version of HO bisimilarity allowing for comparing MST and HO process terms. Our goal is to define *MST bisimilarity* (denoted $\approx^{\mathtt{M}}$), a typed behavioral equivalence, which we give in Definition 4.4.9. To define $\approx^{\mathtt{M}}$, we require some auxiliary definitions, in particular:

- A refined LTS on typed processes (Definition 4.4.5);
- A relation $\bowtie_{\mathsf{c}}$ on values (Definition 4.4.6) and on names (Definition 4.4.14);
- A revised notion of trigger processes (Definition 4.4.8).

Refined LTS and characteristic values. The idea behind defining the refined LTS is to restrict the input of arbitrary processes (values) and make the transition system image-finite (modulo names).

The *refined LTS* for HO is defined in [40] in three layers. First comes the *untyped LTS* $P \xrightarrow{\ell} P'$, which describes reductions of untyped processes in the usual style of the LTS semantics for π-calculus. Secondly, there is a notion of the *environmental LTS* $(\Gamma_1; \Lambda_1; \Delta_1) \xrightarrow{\ell} (\Gamma_2; \Lambda_2; \Delta_2)$, which describes reductions of typing environments. This LTS describes the way a typing context can evolve in accordance with its session types. On top of these layers there are notions of *refined environmental LTS* and *refined LTS for processes.* The former restricts the environmental LTS to inputs on characteristic values, as we discussed in Section 4.4.1.

Finally, the refined LTS for processes restricts the untyped LTS to those actions which are supported by the refined environmental LTS.

We follow this approach for defining the refined LTS for MST processes. Both the untyped LTS for processes and the environmental LTS for MST processes coincides with the same LTSs for HO (or, to be more precise, with its restriction to minimal session types). It remains, then, to define the refined environmental LTS for MST processes, with the idea that the refined LTS restricts inputs to the inputs on *minimal characteristic values* and *minimal trigger values.*

Definition 4.4.2 (Minimal trigger value)**.** Given a value type $C \rightsquigarrow \diamond$ and fresh (indexed) name t_1, the *minimal trigger value* on t_1 of type $\mathcal{G}(C) \rightsquigarrow \diamond$ is defined as the abstraction

$$\lambda \widetilde{x}.\, t_1?(y).y\,\widetilde{x}$$

where $\widetilde{x} = (x_1, \ldots, x_{|\mathcal{G}(C)|})$.

Definition 4.4.3 (Minimal characteristic values)**.** Let u be a name and $i > 0$. We define $\langle - \rangle_i^u$ and $\langle - \rangle_{\mathtt{c}}$ on types as follows.

$$\begin{aligned}
\langle ?(L);S \rangle_i^u &\stackrel{\mathsf{def}}{=} u_i?(x).(t_1!\langle \ulcorner u_{i+1}, \ldots, u_{i+|\mathcal{G}(S)|} \urcorner \rangle.\mathbf{0} \mid \langle L \rangle_i^x) \\
\langle !\langle L \rangle;S \rangle_i^u &\stackrel{\mathsf{def}}{=} u_i!\langle \langle L \rangle_{\mathtt{c}} \rangle.t_1!\langle \ulcorner u_{i+1}, \ldots, u_{i+|\mathcal{G}(S)|} \urcorner \rangle.\mathbf{0} \\
\langle \mathtt{end} \rangle_i^u &\stackrel{\mathsf{def}}{=} \mathbf{0} \\
\langle \mu \mathtt{t}.S \rangle_i^u &\stackrel{\mathsf{def}}{=} \langle S\{\mathtt{end}/\mathtt{t}\} \rangle_i^u \\
\langle \langle L \rangle \rangle_i^u &\stackrel{\mathsf{def}}{=} u_1!\langle \langle L \rangle_{\mathtt{c}} \rangle.t_1!\langle \ulcorner u_1 \urcorner \rangle.\mathbf{0} \\
\langle C \!\rightarrow\! \diamond \rangle_i^x &\stackrel{\mathsf{def}}{=} x\, \langle C \rangle_{\mathtt{c}} \\
\langle C \!\multimap\! \diamond \rangle_i^x &\stackrel{\mathsf{def}}{=} x\, \langle C \rangle_{\mathtt{c}} \\
\langle S \rangle_{\mathtt{c}} &\stackrel{\mathsf{def}}{=} \widetilde{s} \quad (|\widetilde{s}| = |\mathcal{G}(S)|, \widetilde{s} \text{ fresh}) \\
\langle \langle L \rangle \rangle_{\mathtt{c}} &\stackrel{\mathsf{def}}{=} a_1 \quad (a_1 \text{ fresh}) \\
\langle C \!\rightarrow\! \diamond \rangle_{\mathtt{c}} &\stackrel{\mathsf{def}}{=} \lambda(x_1, \ldots, x_{|\mathcal{G}(C)|}).\, \langle C \rangle_1^x \\
\langle C \!\multimap\! \diamond \rangle_{\mathtt{c}} &\stackrel{\mathsf{def}}{=} \lambda(x_1, \ldots, x_{|\mathcal{G}(C)|}).\, \langle C \rangle_1^x
\end{aligned}$$

where t_1 is a fresh (indexed) name. In this definition we use name-passing constructs, as outlined in Example 2.1.1.

Definition 4.4.4 (Refined environmental LTS)**.** The refined LTS, denoted $\stackrel{\ell}{\longmapsto}_{\mathtt{m}}$, is defined on top of the environmental LTS using the following rules:

[MTR]
$$\frac{(\Gamma_1; \Lambda_1; \Delta_1) \xrightarrow{\ell} (\Gamma_2; \Lambda_2; \Delta_2) \qquad \ell \neq n?\langle V \rangle}{(\Gamma_1; \Lambda_1; \Delta_1) \stackrel{\ell}{\longmapsto}_{\mathtt{m}} (\Gamma_2; \Lambda_2; \Delta_2)}$$

[MRCV]
$$\frac{(\Gamma_1; \Lambda_1; \Delta_1) \xrightarrow{n?\langle V \rangle} (\Gamma_2; \Lambda_2; \Delta_2) \qquad (V \equiv \langle L \rangle_{\mathtt{c}}) \vee (V \equiv \lambda \widetilde{x}.\, t_1?(y).(y\,\widetilde{x})) \qquad \text{with } t_1 \text{ fresh}}{(\Gamma_1; \Lambda_1; \Delta_1) \stackrel{n?\langle V \rangle}{\longmapsto}_{\mathtt{m}} (\Gamma_2; \Lambda_2; \Delta_2)}$$

where $\lambda \widetilde{x}.\, t_1?(y).(y\,\widetilde{x})$ is a minimal trigger value of type $\mathcal{G}(C)$ (Definition 4.4.2).

Finally, the refined LTS for MST processes is just a combination of the untyped LTS with the refined environmental LTS:

Definition 4.4.5 (Refined LTS)**.** The environmental refined LTS extends to the typed refined LTS on processes. We write $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1 \overset{\ell}{\longmapsto}_{\mathtt{m}} \Lambda_1'; \Delta_1' \vdash P_2$ when

- $P_1 \xrightarrow{\ell} P_2$, and
- $(\Gamma_1; \Lambda_1; \Delta_1) \overset{\ell}{\longmapsto}_{\mathtt{m}} (\Gamma_2; \Lambda_2; \Delta_2)$.

We write $\overset{\ell}{\Longmapsto}_{\mathtt{m}}$ for the weak version of the transition $\overset{\ell}{\longmapsto}_{\mathtt{m}}$. Notice that while the untyped LTS and the non-refined environmental LTS coincide with that of HO, the refinement that we impose on the environmental LTS is different from its HO counterpart. Specifically in Rule [MRcv] we take special care to use minimal characteristic processes $\langle - \rangle_{\mathtt{c}}$, instead of general HO characteristic process $\llparenthesis - \rrparenthesis_{\mathtt{c}}$ as defined in [40].

Relating trigger and characteristic values. As we mentioned earlier, the notion of bisimulation that we consider requires matching transitions of the source HO term with the transitions of the target MST term. However, the two transitions might differ on the inputs of characteristic values. We accommodate for that difference by establishing a relation between the trigger and characteristic values of HO and MST.

Definition 4.4.6. We define the relation $\bowtie_{\mathtt{c}}$ between HO processes and indexed processes inductively as:

$$\frac{|\tilde{x}| = |\mathcal{G}(C)|}{\lambda x : C.\, t?(y).y\, x \bowtie_{\mathtt{c}} \lambda \tilde{x} : \mathcal{G}(C).\, t_1?(y).y\, \tilde{x}} \qquad \frac{}{\llparenthesis C \rightsquigarrow \diamond \rrparenthesis_{\mathtt{c}} \bowtie_{\mathtt{c}} \langle C \rightsquigarrow \diamond \rangle_{\mathtt{c}}}$$

where $\lambda \tilde{x} : \mathcal{G}(C).\, t_1?(y).(y\, \tilde{x})$ is a minimal trigger value of type $\mathcal{G}(C) \rightsquigarrow \diamond$ (Definition 4.4.2) and $\llparenthesis - \rrparenthesis_{\mathtt{c}}$ denotes the characteristic values defined in [40]. We write $\lambda x : C.\, t_1?(y).y\, x$ to mean that value $\lambda x.\, t_1?(y).y\, x$ is of type $C \rightsquigarrow \diamond$.

Trigger processes and MST bisimilarity. Before we give the definition of MST bisimilarity, we establish the following notations:

Definition 4.4.7 (Indexed name)**.** Given a name n, we write $\breve{n}$ to either denote n or any indexed name n_i, with $i > 0$.

Definition 4.4.8 (Trigger process)**.** Given a value V, a trigger process for a fresh (indexed) name t_1 is defined as:

$$t_1 \hookleftarrow_{\mathtt{H}} V \overset{\mathsf{def}}{=} t_1?(\tilde{x}).(V\ \tilde{x})$$

where $|\tilde{x}| = |\widetilde{C}|$ for $V : \widetilde{C} \rightsquigarrow \diamond$.

Lemma 4.4.1. *If* $\Gamma; \Lambda; \Delta \vdash V \triangleright \widetilde{C} \rightsquigarrow \diamond$*, then* $\Gamma; \Lambda; \Delta, t_1 : ?(\widetilde{C}) \vdash t_1 \hookleftarrow_{\mathtt{H}} V \triangleright \diamond$*.*

Finally, we are ready to formally define MST bisimilarity.

Definition 4.4.9 (MST Bisimilarity)**.** A typed relation $\Re$ is an *MST bisimulation* if for all $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\ \Re\ \Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1$,

1) Whenever $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1 \overset{(\nu\, \widetilde{m_1})\, n!\langle V_1 \rangle}{\longmapsto} \Lambda_1'; \Delta_1' \vdash P_2$ then there exist Q_2, Δ_2', and Λ_2' such that $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \overset{(\nu\, \widetilde{m_2})\, \breve{n}!\langle V_2 \rangle}{\Longmapsto}_{\mathtt{m}} \Lambda_2'; \Delta_2' \vdash Q_2$ where, for a fresh t,

$$\Gamma_1; \Lambda_1; \Delta_1'' \vdash (\nu\, \widetilde{m_1})(P_2 \mid t \hookleftarrow_{\mathtt{H}} V_1)\ \Re\ \Gamma_2; \Lambda_2; \Delta_2'' \vdash (\nu\, \widetilde{m_2})(Q_2 \mid \breve{t} \hookleftarrow_{\mathtt{H}} V_2)$$

2) Whenever $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1 \xrightarrow{n?(V_1)} \Lambda_1'; \Delta_1' \vdash P_2$ then there exist Q_2, Λ_2', and Δ_2' such that $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \overset{\breve{n}?(V_2)}{\Longmapsto}_{\mathtt{m}} \Lambda_2', \Delta_2' \vdash Q_2$ where $V_1 \bowtie_{\mathtt{c}} V_2$ and $\Gamma_1; \Lambda_1'; \Delta_1' \vdash P_2 \ \Re\ \Gamma_2; \Lambda_2'; \Delta_2' \vdash Q_2$,

3) The symmetric cases of 1 and 2.

The largest such bisimulation is called *MST bisimilarity*, denoted by $\approx^{\mathtt{M}}$.

In all clauses, we use the refined LTS (Definition 4.4.5) and rely on notation $\breve{n}$ (Definition 4.4.7). In the output clause, we use the triggers (Definition 4.4.8). In the input clause, we use the relation $\bowtie_{\mathtt{c}}$ on values (Definition 4.4.6).

We discuss differences between MST bisimilarity and higher-order bisimilarity as defined in [40]. First, an action in P_1 must be matched by an action on an indexed name in Q_1, and refined LTS actions in P_1 are matched by minimal refined LTS actions in Q_1 (Definition 4.4.6). As a consequence of the latter, in the input case the observed values are not identical but related by $\bowtie_{\mathtt{c}}$ (Definition 4.4.6). In other words, whenever P_1 receives a trigger or a characteristic value, then Q_1 should receive their minimal counterparts (Definition 4.4.2 and Definition 4.4.3). Further, as names could be indexed on the right-hand side, the typing environments could differ for open processes, so the MST bisimilarity assumes different typing environments on both sides.

4.4.3 The Bisimulation Relation

Our goal is to complement our static correctness result (Theorem 4.3.1) by proving the following statement about the decomposition of processes (Definition 4.3.9):

Theorem 4.4.1. *Let P be an* HO *process such that $\Gamma; \Delta; \Lambda \vdash P \triangleright \diamond$. We have*

$$\Gamma; \Lambda; \Delta \vdash P \ \approx^{\mathtt{M}}\ \mathcal{G}(\Gamma); \mathcal{G}(\Lambda); \mathcal{G}(\Delta) \vdash \mathcal{D}(P)$$

To show that P and $\mathcal{D}(P)$ are MST-bisimilar, we provide a concrete bisimulation relation $\mathcal{S}$ that contains $(P, \mathcal{D}(P))$. Defining $\mathcal{S}$ to be just the set of such pairs is, however, not going to work; instead, the relation $\mathcal{S}$ should also contain pairs corresponding to "intermediate" states in which the process and its decomposition may get "desynchronized". Before we give the concrete definition of $\mathcal{S}$ we look at an example, illustrating the need for such intermediate pairs.

A Motivating Example

Consider the following process:

$$P_1 = u?(t).v?(x).(\nu\, s : S)\,(u!\langle x\rangle.\mathbf{0} \mid t\, s \mid \overline{s}!\langle x\rangle.\mathbf{0}) \mid \overline{v}!\langle V\rangle.\mathbf{0}$$

where $u : ?(\langle U_t\rangle);!\langle U_V\rangle;\mathtt{end}$ and $v : S$ with $S = ?(U_V)$, $U_t = S \multimap \diamond$, and U_V is some shared value type, i.e. $U_V = S_V \multimap \diamond$, for some session type S_V. Further, V is some value, such that $V = \lambda y : S_V.\, R$.

Thus, P_1 is typed using the typing of its constituents:

$$\dfrac{\emptyset; \emptyset; \overline{v} : \overline{S} \vdash \overline{v}!\langle V\rangle.\mathbf{0} \triangleright \diamond \qquad \emptyset; \emptyset; u : ?(\langle U_t\rangle);!\langle U_V\rangle;\mathtt{end}, v : S \vdash u?(t).v?(x).(\nu\, s : S)\,(u!\langle x\rangle.\mathbf{0} \mid t\, s \mid \overline{s}!\langle x\rangle.\mathbf{0}) \triangleright \diamond}{\emptyset; \emptyset; u : ?(\langle U_t\rangle);!\langle U_V\rangle;\mathtt{end}, v : S, \overline{v} : \overline{S} \vdash P_1 \triangleright \diamond}$$

The decomposition of P_1 is as follows:

$$\begin{aligned}\mathcal{D}(P_1) &= (\nu\, \widetilde{c})\ \big(\overline{c_1}!\langle\rangle \mid \mathcal{B}^1_\epsilon(P_1)\big)\\ &= (\nu\, \widetilde{c})\ (\overline{c_1}!\langle\rangle \mid c_1?().\overline{c_2}!\langle\rangle.\overline{c_{11}}!\langle\rangle\\ &\qquad \mid c_2?().u_1?(t).\overline{c_3}!\langle t\rangle \mid c_3?(t).v_1?(x).\overline{c_4}!\langle t,x\rangle\\ &\qquad\quad \mid (\nu\, s_1)\,(c_4?(t,x).\overline{c_5}!\langle x\rangle.\overline{c_6}!\langle t,x\rangle \mid c_5?(x).u_2!\langle x\rangle.\overline{c_6}!\langle\rangle \mid c_6?() \mid\\ &\qquad\qquad \mid c_7?(t,x).\overline{c_8}!\langle t\rangle.\overline{c_9}!\langle x\rangle \mid c_8?(t).t\, s_1 \mid c_9?(x).\overline{s_1}!\langle x\rangle.\overline{c_{10}}!\langle\rangle \mid c_{10}?())\\ &\qquad \mid c_{11}?().\overline{v_1}!\langle\mathcal{V}_\epsilon(V)\rangle.\overline{c_{12}}!\langle\rangle \mid c_{12}?()),\end{aligned}$$

where $\widetilde{c} = c_1, \ldots, c_{12}$. Let us write Q_1 for the decomposition $\mathcal{D}(P_1)$.

We wish to show $P_1 \approx^{\mathtt{M}} Q_1$. For this, we must exhibit a relation $\mathcal{S}$ included in $\approx^{\mathtt{M}}$ such that $(P_1, \mathcal{D}(P_1)) \in \mathcal{S}$. To illustrate the notions required to define the additional pairs, we consider possible transitions of P_1 and Q_1, denoted schematically in Figure 4.5. First, let us consider a possible (refined) transition of P_1, an input on u of a characteristic value:

$$P_1 \xrightarrow{u?\langle V_C\rangle} v?(x).(\nu\, s : S)\,(u!\langle x\rangle.\mathbf{0} \mid V_C\, s \mid \overline{s}!\langle x\rangle.\mathbf{0}) \mid \overline{v}!\langle V\rangle.\mathbf{0} = P_2$$

where $V_C = \lbrace\!\lbrack U_t\rbrack\!\rbrace_{\mathsf{c}} = \lambda y : S.\, y?(x').(!\langle\rangle.\mathbf{0} \mid x'\, s')$ is the *characteristic value* of U_t.[1] Process Q_1 can weakly match this input action on the indexed name u_1. This input does not involve V_C but the *minimal* characteristic value of type U_t (Definition 4.4.3). We have:

$$\begin{aligned}Q_1 \xrightarrow{\tau} Q_1' \xrightarrow{\tau} Q_1'' \xrightarrow{u_1?\langle V_C^m\rangle} &(\nu\, \widetilde{c}_\bullet)\, \overline{c_3}!\langle V_C^m\rangle \mid c_3?(t).v_1?(x).\overline{c_4}!\langle t,x\rangle \mid \overline{c_{11}}!\langle\rangle\\ &\mid (\nu\, s_1)\,(c_4?(t,x).\overline{c_5}!\langle x\rangle.\overline{c_7}!\langle t,x\rangle \mid c_5?(x).u_2!\langle x\rangle.\overline{c_6}!\langle\rangle \mid c_6?() \mid\\ &\quad \mid c_7?(t,x).\overline{c_8}!\langle t\rangle.\overline{c_9}!\langle x\rangle \mid c_8?(t).t\, s_1 \mid c_9?(x).\overline{s_1}!\langle x\rangle.\overline{c_{10}}!\langle\rangle \mid c_{10}?())\\ &\mid c_{11}?().\overline{v_1}!\langle\mathcal{V}_\epsilon(V)\rangle.\overline{c_{12}}!\langle\rangle \mid c_{12}?() = Q_2\end{aligned}$$

where $V_C^m = \langle U_t\rangle_{\mathsf{c}} = \lambda(y_1).\, y_1?(x').(t_1!\langle\rangle.\mathbf{0} \mid x'\, \widetilde{s}')$, with $y_1 : ?(S)$, $|\widetilde{s}'| = |\mathcal{G}(S_V)|$, and $\widetilde{c}_\bullet = c_3, \ldots, c_{12}$.

Hence, we should have $P_2\ \mathcal{S}\ Q_2$. Observe that Q_2 is not exactly the decomposition of P_2. First, V_C^m is not the breakdown of V_C. Second, V_C^m is not at the same position in Q_2 as V_C; the later being in the application position and the former being pushed through several propagators. Therefore, the relation $\mathcal{S}$ needs to (1) relate V_C and V_C^m and (2) account for the fact that a value related to V_C and thus it needs to be propagated (as in Q_2). To address the first point, we establish a relation $\bowtie$ between characteristic values and their minimal counterparts. For the second point, we record this fact by "decomposing" the process as $P_2 = P_2'\{{}^{V_C}/t\}$, and propagating the information about this substitution when computing the set of processes that are related to P_2.

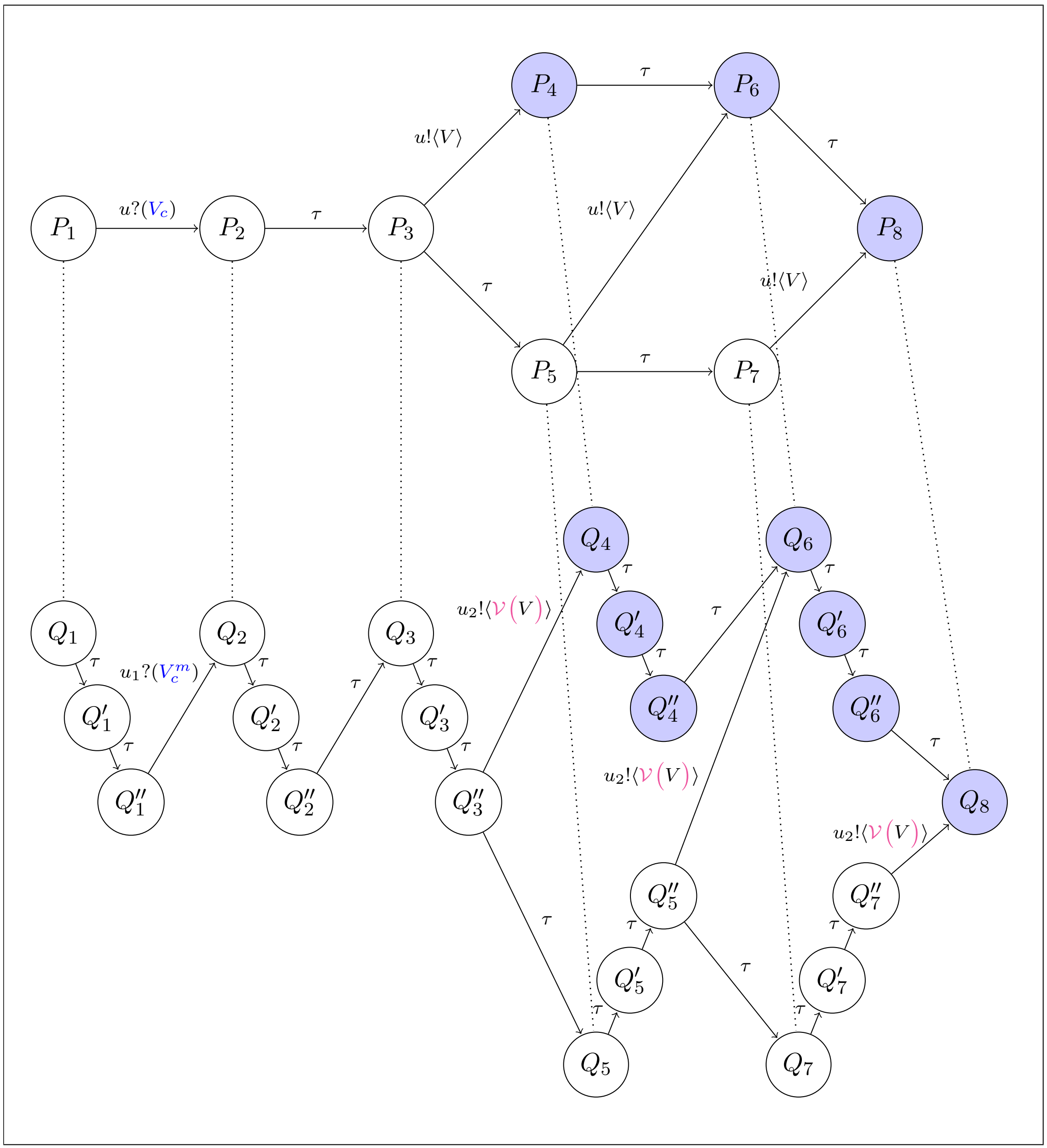
The same considerations we mentioned also apply to the value V, which is transmitted internally, via a synchronization:

$$P_2 \xrightarrow{\tau} (\nu\, s)\,(u!\langle V\rangle.\mathbf{0} \mid V_C\, s \mid \overline{s}!\langle V\rangle.\mathbf{0}) = P_3$$

Value V transmitted in P_2 should be related to its corresponding breakdown $\mathcal{V}_\epsilon(V)$, which should be propagated through the decomposition:

$$\begin{aligned}Q_2 \xrightarrow{\tau} Q_2' \xrightarrow{\tau} Q_2'' \xRightarrow{\tau} &(\nu\, \widetilde{c}_{\bullet\bullet})\, \overline{c_4}!\langle V_C^m, \mathcal{V}_\epsilon(V)\rangle\\ &\mid (\nu\, s_1)\,(c_4?(t,x).\overline{c_5}!\langle x\rangle.\overline{c_7}!\langle t,x\rangle \mid c_5?(x).u_2!\langle x\rangle.\overline{c_6}!\langle\rangle \mid c_6?() \mid\\ &\quad \mid c_7?(t,x).\overline{c_8}!\langle t\rangle.\overline{c_9}!\langle x\rangle \mid c_8?(t).t\, s_1 \mid c_9?(x).\overline{s_1}!\langle x\rangle.\overline{c_{10}}!\langle\rangle \mid c_{10}?()) \mid\\ &\quad \mid \overline{c_{12}}!\langle\rangle \mid c_{12}?() = Q_3\end{aligned}$$

[1]We use blue to denote characteristic values and trigger processes that do no occur in the original process, but which are induced by the bisimilarities defined in [40].



$u?(V_c)$
τ
$u!\langle V\rangle$
$u_1?(V_c^m)$
$u_2!\langle \mathcal{V}(V)\rangle$
P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8
Q_1 Q_1' Q_1'' Q_2 Q_2' Q_2'' Q_3 Q_3' Q_3'' Q_4 Q_4' Q_4'' Q_5 Q_5' Q_5'' Q_6 Q_6' Q_6'' Q_7 Q_7' Q_7'' Q_8

Figure 4.5: Transitions of P_1 and $Q_1 = \mathcal{D}(P_1)$ in Section 4.4.3. The blue nodes represent processes that contain characteristic values and trigger processes induced by the bisimilarites defined in [40].

where $\widetilde{c}_{\bullet\bullet} = c_4, \ldots, c_{10}, c_{12}$.

Now, in P_3 we can observe the output of V along u:

$$P_3 \xrightarrow{u!\langle V\rangle} (\nu\, s)\,(\mathbf{0} \mid V_C\, s \mid \overline{s}!\langle x\rangle.\mathbf{0}) = P_4$$

Process Q_3 mimics this action by sending the process $\mathcal{V}_\epsilon(V)$ along name u_2:

$$Q_3 \xRightarrow{u_2!\langle \mathcal{V}_\epsilon(V)\rangle} (\nu\, \widetilde{c}_*)\, \overline{c_7}!\langle V_C^m, \mathcal{V}_\epsilon(V)\rangle \mid \overline{c_6}!\langle\rangle \mid c_6?() \mid$$

$$\mid c_7?(t,x).\overline{c_8}!\langle t\rangle.\overline{c_9}!\langle x\rangle \mid c_8?(t).t\, s_1 \mid c_9?(x).\overline{s_1}!\langle x\rangle.\overline{c_{10}}!\langle\rangle \mid c_{10}?()) = Q_4$$

where $\widetilde{c}_* = c_6, \ldots, c_{10}$. Following the definition of higher-order bisimilarity, we should have:

$$P_4 \parallel t' \hookleftarrow_{\mathtt{H}} V \;\mathcal{S}\; Q_4 \parallel t'_1 \hookleftarrow_{\mathtt{H}} \mathcal{V}_\epsilon(V)$$

for a fresh t', where we have used '$\parallel$' (rather than '$|$') to denote process composition: we find it convenient to highlight those sub-processes in parallel that originate from trigger and characteristic processes.

We can see that the trigger process for V on the left-hand side should be matched with a trigger process for the *breakdown* of V on the right-hand side. Moreover, the definition of trigger processes should be generalized to polyadic values, as $\mathcal{V}_\epsilon(V)$ could be polyadic (see Definition 4.4.8).

Let us briefly consider how $P_4 \parallel t' \hookleftarrow_{\mathtt{H}} V$ evolves after due to the synchronization in sub-process $V_c\, s$ within P_4:

$$P_4 \parallel t' \hookleftarrow_{\mathtt{H}} V \xrightarrow{\tau} (\nu\, s)\, (s?(x').(t!\langle\rangle \mid x'\, s') \parallel \overline{s}!\langle V\rangle.\mathbf{0}) \parallel t' \hookleftarrow_{\mathtt{H}} V = P_6 \parallel t' \hookleftarrow_{\mathtt{H}} V$$

We can see that Q_4 can mimic this synchronization after a few administrative reductions on propagators:

$$\begin{aligned} Q_4 &\overset{\tau}{\Longrightarrow} (\nu\, c_9 c_{10})\, \overline{c_9}!\langle \mathcal{V}_\epsilon(V)\rangle \mid s_1?(x').(t_1!\langle\rangle \mid x'\, \widetilde{s}') \mid c_9?(x).\overline{s_1}!\langle x\rangle.\overline{c_{10}}!\langle\rangle \mid c_{10}?()) \parallel t'_1 \hookleftarrow_{\mathtt{H}} \mathcal{V}_\epsilon(V) \\ &= Q_6 \parallel t'_1 \hookleftarrow_{\mathtt{H}} \mathcal{V}_\epsilon(V) \end{aligned}$$

Therefore, we need to have:

$$P_6 \parallel t' \hookleftarrow_{\mathtt{H}} V \;\mathcal{S}\; Q_6 \parallel t'_1 \hookleftarrow_{\mathtt{H}} \mathcal{V}_\epsilon(V)$$

To ensure that this pair is in $\mathcal{S}$, we introduce an auxiliary relation, denoted $\diamond$ (Definition 4.4.15), which allows us to account for the sub-processes that originate from characteristic values or trigger processes (in blue). We need to account for them separately, because one of them is not the decomposition of the other. We thus decree:

$$\begin{gathered} s?(x').(t!\langle\rangle \mid x'\, s') \diamond s_1?(x').(t_1!\langle\rangle \mid x'\, \widetilde{s}') \\ t' \hookleftarrow_{\mathtt{H}} V \diamond t'_1 \hookleftarrow_{\mathtt{H}} \mathcal{V}_\epsilon(V) \end{gathered}$$

Next, the synchronization on s in P_6 is mimicked by Q_6 with a synchronization on s_1:

$$\begin{gathered} P_6 \parallel t \hookleftarrow_{\mathtt{H}} V \xrightarrow{\tau} (t!\langle\rangle.\mathbf{0} \mid V\, s') \parallel t' \hookleftarrow_{\mathtt{H}} V = P_8 \parallel t' \hookleftarrow_{\mathtt{H}} V \\ Q_6 \parallel t'_1 \hookleftarrow_{\mathtt{H}} \mathcal{V}_\epsilon(V) \overset{\tau}{\Longrightarrow} (\nu\, c_{10})\, (t_1!\langle\rangle \mid \mathcal{V}_\epsilon(V)\, \widetilde{s}') \mid \overline{c_{10}}!\langle\rangle \mid c_{10}?()) = Q_8 \parallel t'_1 \hookleftarrow_{\mathtt{H}} \mathcal{V}_\epsilon(V) \end{gathered}$$

Finally, we can see that after the output on the trigger name t there is an application that activates R, the body of V:

$$\begin{aligned} &P_8 \xrightarrow{t!\langle\rangle} V\, s' \xrightarrow{\tau} R\{s'/y\} \\ &Q_8 \xrightarrow{t!\langle\rangle} \mathcal{V}_\epsilon(V)\, \widetilde{s}' \xrightarrow{\tau} (\nu\, \widetilde{c}_{**})\, \overline{c_{12}}!\langle\rangle \mid \mathcal{B}^{12}_\epsilon(R)\{\widetilde{s}'/\widetilde{y}\} \equiv \mathcal{D}(R\{s'/y\}) \end{aligned}$$

We reached the point where we relate process $R\{s'/y\}$ with its decomposition $\mathcal{D}(R\{s'/y\})$. Hence, the remaining pairs in $\mathcal{S}$ are obtained in the same way.

Key insights. We summarize some key insights from the example:

- A received value can either be a pure value or a characteristic value. In the former case, the pure value has to be related to its decomposition, but in the later case the value should be related to an MST characteristic value of the same type. We define the relation $\bowtie$ on values to account for this (Definition 4.4.13).
- Trigger processes mentioned in the output case of MST bisimilarity should be matched with their minimal counterparts, and the same applies to processes originating from such trigger processes. The relation $\diamond$ accounts for this (see Definition 4.4.15).
- Any value in process P could have been previously received. The definition of $\mathcal{S}$ takes this into account by explicitly relating processes with substitutions (see Definition 4.4.17). That is, for P, it relates $P'\{\tilde{W}/\tilde{x}\}$ such that $P'\{\tilde{W}/\tilde{x}\} = P$. Here, the substitution $\{\tilde{W}/\tilde{x}\}$ records values that should be propagated.

The relation $\mathcal{S}$

In this section we give the definition of the relation $\mathcal{S}$ (Definition 4.4.17), following the insights gathered from the example. More specifically, we define

- a relation $\bowtie$ on values, which includes the relation $\bowtie_{\mathsf{c}}$ from Definition 4.4.6, (Definition 4.4.13);
- a relation $\diamond$ on processes, for relating characteristic and trigger processes with their MST counterparts, (Definition 4.4.15);
- a set $\mathcal{C}_{\tilde{x}}^{\tilde{W}}(P)$ of processes *correlated* to a process $P\{\tilde{W}/\tilde{x}\}$, (Table 4.3).

Because we will be working extensively with indexed processes, we will use the following function, which returns a set of all valid indexing substitutions for a list of names.

Definition 4.4.10 (Indexed names substitutions)**.** Let $\widetilde{u} = (a, b, r, \overline{r}, r', \overline{r}', s, \overline{s}, s', \overline{s}', \dots)$ be a finite tuple of names, where $a, b, \dots$ denote shared names, $r, \overline{r}, r', \overline{r}', \dots$ denote tail-recursive names , and $s, \overline{s}, s', \overline{s}', \dots$ denote linear (non tail-recursive names). We write $\mathsf{index}(\widetilde{u})$ to denote

$$\mathsf{index}(\widetilde{u}) = \{{}^{a_1, b_1, r_1, \overline{r}_1, r'_1, \overline{r}'_1, s_i, \overline{s}_i, s'_j, \overline{s}'_j, \cdots}/_{a, b, r, \overline{r}, r', \overline{r}', s, \overline{s}, s', \overline{s}', \dots} : i, j, \dots > 0\}$$

Any substitution $\sigma \in \mathsf{index}(\mathtt{fn}(P))$ turns an HO process P into an indexed process $P\sigma$.

Correlated values. The main ingredient in defining the relation $\mathcal{S}$ is the the set $\mathcal{C}_{\tilde{x}}^{\tilde{W}}(P)$, which contains processes *correlated* to process P with a substitution $\{\tilde{W}/\tilde{x}\}$. The substitution, as discussed above, denotes previously received values, and we assume that $\mathtt{fv}(P) = \widetilde{x}$. Essentially, $\mathcal{C}_{-}^{-}(-)$ computes a breakdown of $P\{\tilde{W}/\tilde{x}\}$ in parallel with an activating trio, that mimics the original actions of P up to transitions on propagators. The activating trio propagates not the original values $\widetilde{W}$, but the values related to $\widetilde{W}$. To do that we introduce the set $\mathcal{C}_{-}^{-}(V)$ of correlated values and the relation $\bowtie$ on values, which are defined mutually recursively in the three following definitions.

Definition 4.4.11 (Broken down values)**.** Given a value V, the set $\mathcal{C}(V)$ is defined as follows:

$$\mathcal{C}(V) = \bigcup \{\mathcal{C}_{\tilde{x}}^{\tilde{W}}(V') : V = V'\{\tilde{W}/\tilde{x}\} \text{ and } V' \text{ is not a variable}\}$$

We extend $\mathcal{C}(-)$ to work on a list of values $\widetilde{V}$ component-wise, that is:

$$\mathcal{C}(V_1, \dots, V_n) = \{B_1, \dots, B_n : B_i \in \mathcal{C}(V_i) \text{ for } i \in 1 \dots n\}.$$

This way, the elements in $\mathcal{C}(V)$ differ in the propagated values $\widetilde{W}$. Consider the following example:

Example 4.4.1. Let $V = \lambda y.\, y!\langle V_1\rangle.y!\langle V_2\rangle.\mathbf{0}$. There are four possibilities of V', $\widetilde{W}$, and $\widetilde{x}$ such that $V = V'\{\tilde{W}/\tilde{x}\}$. That is,

- $V = V^1\{V_1V_2/x_1x_2\}$ where $V_1 = \lambda y.\, y!\langle x_1\rangle.y!\langle x_2\rangle.\mathbf{0}$
- $V = V^2\{V_1/x_1\}$ where $V^2 = \lambda y.\, y!\langle x_1\rangle.y!\langle V_2\rangle.\mathbf{0}$
- $V = V^3\{V_2/x_2\}$ where $V^3 = \lambda y.\, y!\langle V_1\rangle.y!\langle x_2\rangle.\mathbf{0}$
- Finally, we can take the identity substitution $\widetilde{W} = \epsilon$ and $\widetilde{x} = \epsilon$.

Thus, we have $\mathcal{C}(V) = \{\mathcal{C}^{V_1V_2}_{x_1x_2}(V^1),\ \mathcal{C}^{V_1}_{x_1}(V^2),\ \mathcal{C}^{V_2}_{x_2}(V^3),\ \mathcal{C}^{\epsilon}_{\epsilon}(V)\}$.

Definition 4.4.12. Given a value V, the set $\mathcal{C}^{\tilde{W}}_{\tilde{x}}(V)$, where $\mathtt{fn}(V) = \widetilde{x}$ is defined as follows:

$$\mathcal{C}^{\tilde{W}}_{\tilde{x}}(V) = \{\mathcal{V}_{\tilde{x}}(V)\{\tilde{B}/\tilde{x}\} \mid \widetilde{W} \bowtie \widetilde{B}\}.$$

Definition 4.4.13 (Relating values)**.** The relation $\bowtie$ on values (with indexed names) is defined as follows:

$$V_1 \bowtie V_2 \iff \begin{cases} \exists V_1',\, \sigma \in \mathsf{index}(\mathtt{fn}(V_1')).\ V_1 = V_1'\sigma \wedge V_1' \bowtie_{\mathsf{c}} V_2 & \text{if } V_1 \text{ is a characteristic or a trigger value} \\ V_2 \in \mathcal{C}(V_1) & \text{otherwise.} \end{cases}$$

where $\bowtie_{\mathsf{c}}$ is the relation from Definition 4.4.6.

Thus, in the definition of $\mathcal{C}^{\tilde{W}}_{\tilde{x}}(V)$, the value V is related to the triggered break down values with $\widetilde{B}$ substituted for $\widetilde{x}$ such that $\widetilde{W} \bowtie \widetilde{B}$.

Additionally, to define $\mathcal{C}^{\tilde{W}}_{\tilde{x}}(-)$ for processes, we have to observe the behaviour of processes enclosed in the received trigger and characteristic values. Further, we have to observe the behaviour of trigger processes of shape $t \hookleftarrow_{\mathtt{H}} V$. For this we need to define a relation $\diamond$ on processes that contains pairs

$$(\llparenthesis C \rrparenthesis^{x},\ \langle C\rangle^{\tilde{x}}_{1}), (t?(y).y\,x,\ t_1?(y).y\,\widetilde{x}), (t \hookleftarrow_{\mathtt{H}} V,\ t_1 \hookleftarrow_{\mathtt{H}} W)$$

where $x : C$ and $|\widetilde{x}| = |\mathcal{G}(C)|$ and $V \bowtie W$.

Before we define $\diamond$ we need the following auxiliary definition:

Definition 4.4.14 (Relating names)**.** We define $\diamond$ as the relation on names defined as

$$\frac{}{\epsilon \diamond \epsilon} \qquad \frac{\Gamma; \Lambda; \Delta \vdash n_i \triangleright C}{n_i \diamond (n_i, \ldots, n_{i+|\mathcal{G}(C)|-1})} \qquad \frac{\tilde{n} \diamond \tilde{m}_1 \quad n_i \diamond \tilde{m}_2}{\tilde{n}, n_i \diamond \tilde{m}_1, \tilde{m}_2}$$

where ϵ denotes the empty list.

Now, we are ready to relate processes, modulo indexed names (cf. Definition 4.4.7), using the relation $\diamond$ defined as follows:

Definition 4.4.15 ($\diamond$ Indexed process relation)**.** We define the relation $\diamond$ as

$$\text{[IPApp]}\ \frac{V \bowtie W \qquad x_i \diamond \tilde{x}}{V\, x_i \diamond W\, \tilde{x}} \qquad \text{[IPPar]}\ \frac{P \diamond P' \qquad Q \diamond Q'}{P \mid Q \diamond P' \mid Q'}$$

$$\text{[IPInact]}\ \frac{}{\mathbf{0} \diamond \mathbf{0}} \qquad \text{[IPNews]}\ \frac{P \diamond P' \qquad \tilde{m}_1 \diamond \tilde{m}_2}{(\nu\, \tilde{m}_1)\, P \diamond (\nu\, \tilde{m}_2)\, P'}$$

$$\text{[IPSnd]}\ \frac{P\sigma \diamond P' \qquad V\sigma \bowtie W \qquad \sigma = \mathsf{next}(n_i)}{n_i!\langle V\rangle.P \diamond n_i!\langle W\rangle.P'} \qquad \text{[IPRcv]}\ \frac{P\sigma \diamond P' \qquad \sigma = \mathsf{next}(n_i)}{n_i?(y).P \diamond n_i?(y).P'}$$

We can now show the property that we wanted, namely that: the bodies of trigger values and minimal trigger values (Definition 4.4.2) are related; the bodies of characteristic values and minimal characteristic values (Definition 4.4.3) are related; and that the trigger processes and minimal trigger processes (Definition 4.4.8) are related, with appropriate name substitutions.

Lemma 4.4.2. *We have:*

$$\{(\llparenthesis C \rrparenthesis^x\{{}^{x_i,t_1}/_{x,t}\},\ \langle C\rangle_i^x),\ (t_1?(y).y\, x\{{}^{x_i}/_{x}\},\ t_1?(y).y\, \tilde{x}),\ (t \hookleftarrow_{\mathtt{H}} V\sigma,\ t_1 \hookleftarrow_{\mathtt{H}} W)\} \subset \diamond$$

where $i, j > 0$*,* $x : C$*,* $\tilde{x} = (x_i, \dots, x_{i+|\mathcal{G}(C)|-1})$*,* $\sigma \in \mathsf{index}(\tilde{u})$*,* $\tilde{u} = \mathtt{fn}(V)$*, and* $V\sigma \bowtie W$*.*

Proof (Sketch). We may notice that $\diamond$ relates process up to incremented indexed names and values related by $V\sigma \bowtie W$ for some σ. More precisely, free names as subject of actions are indexed and incremented accordingly in a related process, and names as objects of output actions are broken down in a related process, by $V\sigma \bowtie_{\mathsf{c}} W$ when $V\sigma = m_i$, that is $m_i \bowtie_{\mathsf{c}} \tilde{m}$ where $m_i : C$ and $\tilde{m} = (m_i, \dots, m_{i+|\mathcal{G}(C)|-1})$.

For the first pair $(\llparenthesis C \rrparenthesis^x\{{}^{x_i,t_1}/_{x,t}\},\ \langle C\rangle_i^x)$ by inspection of Definition 4.4.7 we can observe that $\langle C\rangle_i^x$ is essentially $\llparenthesis C \rrparenthesis^x$ with its subject names indexed and incremented (starting with index i) and objects names broken down. Thus, it is contained in $\diamond$. Similarly, $(t?(y).y\, x\{{}^{x_i}/_{x}\},\ t_1?(y).y\, \tilde{x})$ is contained by observing that $x_i \diamond \tilde{x}$. Finally, for $(t \hookleftarrow_{\mathtt{H}} V\sigma,\ t_1 \hookleftarrow_{\mathtt{H}} W)$, by Definition 4.4.8, $V\sigma \bowtie W$. $\square$

Correlated Processes. Finally, we can use the introduced notions to define the set $\mathcal{C}_{-}^{-}(\,-\,)$ of correlated processes. As mentioned, the set $\mathcal{C}_{\tilde{x}}^{\tilde{W}}(P)$ contains processes correlated to process P with a substitution $\{{}^{\tilde{W}}/_{\tilde{x}}\}$. The definition of $\mathcal{C}_{-}^{-}(\,-\,)$ is given in Table 4.3. Before looking into the details, we first describe how the $\mathcal{C}_{-}^{-}(\,-\,)$ is used.

We introduce auxiliary notions for treating free (tail-recursive) names in processes.

Definition 4.4.16 (Auxiliary Notions)**.** Let P be an HO process.

- We write $\mathsf{fpn}(P)$ to denote the set of free propagator names in P.
- We define $\mathtt{rfv}(P)$ to denote free tail-recursive names in values in P.
- We define $\mathsf{cr}(P)$ to denote free names of form c^r in P.
- We define $\mathsf{rfni}(P)$ such that $r \in \mathsf{rfni}(P)$ if and only if $(r_i, \dots, r_j) \subseteq \mathtt{rn}(P)$ for some $i, j > 0$.

- Given $r : S$ and $\widetilde{r} = (r_1, \ldots, r_{|\mathcal{G}(S)|})$, we write $\mathcal{R}_{\widetilde{v}}$ to denote the process

$$\mathcal{R}_{\widetilde{v}} = \prod_{r \in \widetilde{v}} c^r?(x).x\,\widetilde{r}$$

Definition 4.4.17 (Relation $\mathcal{S}$)**.** Let $P\{\widetilde{W}/\widetilde{x}\}$ be a well-typed process such that $\mathtt{fn}(P) \cap \mathtt{fn}(\widetilde{W}) = \emptyset$, and let the $\mathcal{C}$-set be as in Table 4.3. We define the relation $\mathcal{S}$ as follows:

$$\begin{aligned}\mathcal{S} \ = \{ & (P\{\widetilde{W}/\widetilde{x}\}, (\nu\,\widetilde{c}_r)\,(\nu\,\widetilde{c})\,R) : \ R \in \mathcal{C}^{\widetilde{W}\sigma}_{\widetilde{x}}(P\sigma) \\ & \text{with } \widetilde{u} = \mathtt{fn}(P\{\widetilde{W}/\widetilde{x}\}),\ \sigma \in \mathsf{index}(\widetilde{u}),\ \widetilde{c}_r = \mathsf{cr}(R),\ \widetilde{c} = \mathsf{fpn}(R)\}\end{aligned}$$

Now we describe the definition of $\mathcal{C}^{-}_{-}(\,-\,)$ in Table 4.3. Essentially, $\mathcal{C}^{-}_{-}(\,-\,)$ computes a breakdown of $P\{\widetilde{W}/\widetilde{x}\}$ in parallel with an activating trio, that mimics the original actions of P up to transitions on propagators. This is done with the help of $\mathcal{J}^{-}_{-}(\,-\,)$ (also given in Table 4.3), which computes a closure of a process with respect to τ-transitions on propagators.

To define the $\mathcal{C}$-set we distinguish processes that do not appear in the given process, but that are composed in parallel by the clauses of MST bisimilarity (Definition 4.4.9). For this we use the following notions:

Definition 4.4.18 (Trigger Collections)**.** We let H, H' to range over *trigger collections*: processes of the form $P_1 \mid \cdots \mid P_n$ (with $n \geq 1$), where each P_i is a trigger process or a process that originates from a trigger or from a characteristic value.

Example 4.4.2. Let $H_1 = t \hookleftarrow_{\mathtt{H}} V \mid (\!|C|\!)^u \mid t'!\langle n\rangle.\mathbf{0}$ where t, t', u, n are channel names, V is a value, and C a channel type. Then, we could see that $t'!\langle n\rangle.\mathbf{0}$ originates from a characteristic value. Thus, H_1 is a trigger collection.

Notice that we write P to denote a "pure" process that is not composed with a trigger collection. For processes with trigger collections, the following notation is relevant:

Definition 4.4.19 (Process in parallel with a trigger or a characteristic process)**.** We write $P \parallel Q$ to stand for $P \mid Q$ where either P or Q is a trigger collection.

Now we can describe all the cases in the definitions of the $\mathcal{J}$-set and the $\mathcal{C}$-set in Table 4.3 (Page 97). Observe that the second and third columns in Table 4.3 are closely related: the third column lists side conditions for the definitions in the second column. Note that in each case we assume the substitution $\rho = \{\widetilde{W}/\widetilde{x}\}$. We start with the cases for $\mathcal{C}^{\widetilde{W}}_{\widetilde{x}}(P)$:

Parallel with a trigger collection: The $\mathcal{C}$-set of $Q_1 \parallel Q_2$ is defined as:

$$\{R_1 \parallel R_2 : R_1 \in \mathcal{C}^{\widetilde{W}_1}_{\widetilde{y}}(Q_1),\ R_2 \in \mathcal{C}^{\widetilde{W}_2}_{\widetilde{w}}(Q_2)\}$$

By Definition 4.4.19, either Q_1 or Q_2 is a trigger collection. Notice that a composition $Q_1 \mid Q_2$ (where both Q_1 and Q_2 are "pure") is handled by $\mathcal{J}(\,-\,)$, see below. We treat $Q_1 \parallel Q_2$ compositionally: we split the substitution into parts concerning Q_1 and Q_2, i.e., $\{\widetilde{W}/\widetilde{x}\} = \{\widetilde{W_1}/\widetilde{y}\} \cdot \{\widetilde{W_2}/\widetilde{w}\}$ such that $\widetilde{y} = \mathtt{fv}(Q_1)$ and $\widetilde{w} = \mathtt{fv}(Q_2)$, and relate it to a parallel composition whose components come from a corresponding $\mathcal{C}$-set.

Restriction: The $\mathcal{C}$-set of $(\nu\, m : C)\, Q$ is inductively defined as:

$$\Big\{(\nu\,\widetilde{m} : \mathcal{G}(C))\ R : (\nu\,\widetilde{c}^m)\, R \in \mathcal{C}^{\widetilde{W}}_{\widetilde{x}}(Q\sigma)\Big\}$$

where $\sigma = \{m_1\overline{m_1}/m\overline{m}\}$ and $\widetilde{m} = (m_1, \ldots, m_{|\mathcal{G}(C)|})$ is the decomposition of m under C. The elements are processes from the $\mathcal{C}$-set of Q with names $\widetilde{m}$ restricted. In

P	$\mathcal{C}^{\tilde{W}}_{\tilde{x}}(P)$	
$Q_1 \parallel Q_2$	$\{R_1 \parallel R_2 : R_1 \in \mathcal{C}^{\tilde{W}_1}_{\tilde{y}}(Q_1), R_2 \in \mathcal{C}^{\tilde{W}_2}_{\tilde{w}}(Q_2)\}$	$\widetilde{y} = \mathtt{fv}(Q_1),\ \widetilde{w} = \mathtt{fv}(Q_2)$ $\{\tilde{W}/\tilde{x}\} = \{\tilde{W}_1/\tilde{y}\} \cdot \{\tilde{W}_2/\tilde{w}\}$
$(\nu\, m : C)\ Q$	$\Big\{(\nu\, \tilde{m} : \mathcal{G}(C))\,(\nu\, \tilde{c}^m)\ R : R \in \mathcal{C}^{\tilde{W}}_{\tilde{x}}(Q\sigma)\Big\}$	$\widetilde{m} = (m_1, \ldots, m_{\lvert\mathcal{G}(C)\rvert})$ $\sigma = \{m_1\overline{m_1}/m\overline{m}\}$ $\tilde{c}^m = (\mathsf{tr}(C))\,?\, c^m \cdot c^{\overline{m}} : \epsilon$
Q	$\{\mathcal{R}_{\tilde{v}} \mid \overline{c_k}!\langle \widetilde{B}\rangle \mid \mathcal{B}^k_{\tilde{x}}(P)\}$ $\cup\ \{\mathcal{R}_{\tilde{v}\setminus\tilde{r}} \mid R : R \in \mathcal{J}^{\tilde{W}}_{\tilde{x}}(P),\ \widetilde{r} = \mathsf{rfni}(R)\}$	$\widetilde{W} \bowtie \widetilde{B}$ $\widetilde{v} = \mathtt{rn}(P\{\tilde{W}/\tilde{x}\})$
H	$\{\mathcal{R}_{\tilde{v}} \parallel H' : H\{\tilde{W}/\tilde{x}\} \diamond H'\}$	$\tilde{v} = \mathtt{rn}(H\{\tilde{W}/\tilde{x}\})$

P	$\mathcal{J}^{\tilde{W}}_{\tilde{x}}(P)$	
$u_i!\langle V_1\rangle.Q$	• $\neg\mathsf{tr}(C)$: $\{u_i!\langle V_2\rangle.\overline{c_k}!\langle \widetilde{B}_2\rangle \mid \mathcal{B}^k_{\tilde{z}}(Q\sigma)\}$ • $\mathsf{tr}(C)$: $\{c^u!\langle M^{\tilde{B}_2}_{V_2}\rangle \mid\ \mathcal{B}^k_{\tilde{w}}(Q),\ M^{\tilde{B}_2}_{V_2}\,\widetilde{u} \mid \mathcal{B}^k_{\tilde{w}}(Q),$ $u_{[S\rangle}!\langle V_2\rangle.(\overline{c_k}!\langle \widetilde{B}_2\rangle \mid c^u?(x).x\,\widetilde{u}) \mid \mathcal{B}^k_{\tilde{w}}(Q)\}$ where: $M^{\tilde{B}}_{V} = \lambda\widetilde{z}.\, z_{[S\rangle}!\langle V\rangle.\ (\overline{c_k}!\langle \widetilde{B}\rangle \mid c^u?(x).x\,\widetilde{z})$	$\widetilde{y} = \mathtt{fv}(V_1),\ \widetilde{w} = \mathtt{fv}(Q)$ $\{\tilde{W}/\tilde{x}\} = \{\tilde{W}_1/\tilde{y}\} \cdot \{\tilde{W}_2/\tilde{w}\}$ $\sigma = \mathsf{next}(u_i)$ $V_1\sigma\{\tilde{W}_1/\tilde{y}\} \bowtie V_2,\ \widetilde{W}_2 \bowtie \widetilde{B}_2$ $\widetilde{z} = (z_1, \ldots, z_{\lvert\mathcal{R}^\star(S)\rvert})$ $\widetilde{u} = (u_1, \ldots, u_{\lvert\mathcal{R}^\star(S)\rvert})$
$u_i?(y).Q$	• $\neg\mathsf{tr}(C)$: $\{u_i?(y).\overline{c_k}!\langle \widetilde{B}y\rangle \mid \mathcal{B}^k_{\tilde{x}y}(Q\sigma)\}$ • $\mathsf{tr}(C)$: $\{c^u!\langle M^{\tilde{B}}_{y}\rangle \mid \mathcal{B}^k_{\tilde{x}y}(Q),\ M^{\tilde{B}}_{y}\,\widetilde{u} \mid \mathcal{B}^k_{\tilde{x}y}(Q),$ $u_{[S\rangle}?(y).(\overline{c_k}!\langle \widetilde{B}y\rangle \mid c^u?(x).x\,\widetilde{u}) \mid \mathcal{B}^k_{\tilde{x}y}(Q)\}$ where: $M^{\tilde{B}}_{y} = \lambda\widetilde{z}.\, z_{[S\rangle}?(y).(\overline{c_k}!\langle \widetilde{B}y\rangle \mid c^u?(x).x\,\widetilde{z})$	$\widetilde{W} \bowtie \widetilde{B}$ $\sigma = \mathsf{next}(u_i)$ $\widetilde{z} = (z_1, \ldots, z_{\lvert\mathcal{R}^\star(S)\rvert})$ $\widetilde{u} = (u_1, \ldots, u_{\lvert\mathcal{R}^\star(S)\rvert})$
$V_1\,(\widetilde{r}, u_i)$	$\{\ \overbrace{c^{r_l}!\langle\lambda\widetilde{z}_l.c^{r_{l+1}}!\langle\lambda\widetilde{z}_{l+1}.\cdots.c^{r_n}!\langle\lambda\widetilde{z}_n.}^{\lvert\tilde{r}\rvert-l+1}Q_l\rangle\,\rangle\rangle,$ $\lambda\widetilde{z}_l.\overbrace{c^{r_{l+1}}!\langle\lambda\widetilde{z}_{l+1}.\cdots.c^{r_n}!\langle\lambda\widetilde{z}_n.}^{\lvert\tilde{r}\rvert-l}Q_l\rangle\,\rangle\ \widetilde{r}_l,$ $: 1 \leq l \leq n,\ V_1\{\tilde{W}/\tilde{x}\} \bowtie V_2\}$ $\cup\,\{V_2\,\widetilde{r}_1, \ldots, \widetilde{r}_n, \widetilde{m} : V_1\{\tilde{W}/\tilde{x}\} \bowtie V_2\}$ where: $Q_l = V_2\,(\widetilde{r}_1, \ldots, \widetilde{r}_{l-1}, \widetilde{z}_l, \ldots, \widetilde{z}_n, \widetilde{m})$	$\forall r_i \in \widetilde{r}.(r_i : S_i \wedge \mathsf{tr}(S_i) \wedge$ $\widetilde{z}_i = (z^i_1, \ldots, z^i_{\lvert\mathcal{R}^\star(S_i)\rvert}),$ $\widetilde{r}_i = (r^i_1, \ldots, r^i_{\lvert\mathcal{R}^\star(S_i)\rvert}))$ $u_i : C$ $\widetilde{m} = (u_i, \ldots, u_{i+\lvert\mathcal{G}(C)\rvert-1})$
$Q_1 \mid Q_2$	$\{\overline{c_k}!\langle \widetilde{B}_1\rangle.\overline{c_{k+l}}!\langle \widetilde{B}_2\rangle \mid \mathcal{B}^k_{\tilde{y}}(Q_1) \mid \mathcal{B}^{k+l}_{\tilde{z}}(Q_2)\}$ $\cup$ $\{(R_1 \mid R_2) : R_1 \in \mathcal{C}^{\tilde{W}_1}_{\tilde{y}}(Q_1), R_2 \in \mathcal{C}^{\tilde{W}_2}_{\tilde{z}}(Q_2)\}$	$l = \lbag Q_1 \rbag,\ \widetilde{W}_1 \bowtie \widetilde{B}_1,\ \widetilde{W}_2 \bowtie \widetilde{B}_2$ $\widetilde{y} = \mathtt{fv}(Q_1),\ \widetilde{z} = \mathtt{fv}(Q_2)$ $\{\tilde{W}/\tilde{x}\} = \{\tilde{W}_1/\tilde{y}\} \cdot \{\tilde{W}_2/\tilde{z}\}$
$\mathbf{0}$	$\mathbf{0}$	

Table 4.3: The sets $\mathcal{C}^{\tilde{W}}_{\tilde{x}}(P)$ and $\mathcal{J}^{\tilde{W}}_{\tilde{x}}(P)$.

the case when restricted name m is a tail-recursive then we also restrict the special propagator names c^m and $c^{\overline{m}}$ which appear in R. Notice that the processes of the form $(\nu\, m)\,(Q_1 \parallel Q_2)$, which are induced by the output clause of MST bisimilarity, are treated in this case in the definition of $\mathcal{C}\,(\,-\,)$.

Pure process: The $\mathcal{C}$-set of a pure process Q is defined as follows:

$$\{\mathcal{R}_{\tilde{v}} \mid \overline{c_k}!\langle\widetilde{B}\rangle \mid \mathcal{B}^{k}_{\tilde{x}}(Q) : \widetilde{W} \bowtie \widetilde{B}\} \cup \{\mathcal{R}_{\tilde{v}\backslash\tilde{r}} \mid R : R \in \mathcal{J}^{\widetilde{W}}_{\tilde{x}}(Q),\ \tilde{r} = \mathsf{rfni}(R)\}$$

where $\widetilde{v} = \mathtt{rn}(Q\{\widetilde{W}/\tilde{x}\})$. The elements in the first set are essentially the decomposition of Q (without restrictions of recursive propagators, which are handled in $\mathcal{S}$) up to different possibilities of values $\widetilde{B}$ that are $\bowtie$-related to $\widetilde{W}$ (see Definition 4.4.13). Here, we remark that $\mathcal{R}_{\tilde{v}}$ is recursive name providers for all tail-recursive names of Q and $\widetilde{W}$ (by $\widetilde{v} = \mathtt{rn}(Q\{\widetilde{W}/\tilde{x}\})$). The second set contains elements of the $\mathcal{J}$-set of Q in parallel with $\mathcal{R}_{\tilde{v}\backslash\tilde{r}}$ where $\widetilde{r} = \mathsf{rfni}(R)$. By Definition 4.4.16 we can see that $\mathsf{rfni}(R)$ denotes tail-recursive names already gathered in R by communications that consumed $\mathcal{R}_{\tilde{r}}$: thus, we have $\mathcal{R}_{\tilde{v}\backslash\tilde{r}}$ as providers at top level.

In this sense, the processes from the second set can be seen as reducts of the processes from the first set. For example, if we examine the $\mathcal{C}$-set corresponding to the process P_2 from Figure 4.5, we note that the process Q_2 belongs to the first set, and the processes Q_2' and Q_3'' belong to the second set.

Trigger collection: The $\mathcal{C}$-set of a trigger collection H contains its minimal counterparts, defined using the relation $\diamond$ (Definition 4.4.15):

$$\{\mathcal{R}_{\tilde{v}} \parallel H' : H\{\widetilde{W}/\tilde{x}\} \diamond H'\}.$$

where $\widetilde{v} = \mathtt{rn}(H\{\widetilde{W}/\tilde{x}\})$. In this case we do not use the information on the substitution $\{\widetilde{W}/\tilde{x}\}$, because the substitution information is needed for values that are, or were, propagated. However, because H is a trigger collection, it will only contain propagators as part of values. The substitutions related the propagators in values are already handled by the relation $\bowtie$, invoked by $\diamond$. As in the case with pure processes, the process $\mathcal{R}_{\tilde{r}}$ is the recursive names provider for the tail-recursive names of H.

We now discuss the cases for $\mathcal{J}^{\widetilde{W}}_{\tilde{x}}(P)$:

Output: The $\mathcal{J}$-set of $u_i!\langle V_1\rangle.Q$ depends on whether (i) u_i is linear or shared name (i.e., $\neg\mathsf{tr}(u_i)$) or (ii) u_i is a tail-recursive name (i.e., $\mathsf{tr}(u_i)$). In sub-case (i) $\mathcal{J}$-set is defined as follows:

$$\{u_i!\langle V_2\rangle.\overline{c_k}!\langle\widetilde{B}_2\rangle \mid \mathcal{B}^{k}_{\tilde{z}}(Q\sigma) : V_1\sigma\{\widetilde{W}_1/\tilde{y}\} \bowtie V_2,\ \widetilde{W}_2 \bowtie \widetilde{B}_2\}$$

where $\sigma = \mathsf{next}(u_i)$. By the definition, the substitution σ depends on whether u_i is linear or shared: in the former case, we use a substitution that increments u_i; in the latter case we use an identity substitution. We split $\widetilde{W}$ into $\widetilde{W}_1$ and $\widetilde{W}_2$, associated to the emitted value V_1 and the continuation Q, respectively.

Instead of the emitted value V_1 we consider values V_2 that are $\bowtie$-related to $V_1\sigma\{\widetilde{W}_1/\tilde{y}\}$. This way, we uniformly handle cases when (i) V_1 is a pure value, (ii) variable, and (iii) a characteristic value. In particular, if V_1 is a pure value, the set $\mathcal{C}^{\widetilde{W}_1}_{\tilde{y}}(V_1\sigma)$ is included in all the values $\bowtie$-related to $V_1\sigma\{\widetilde{W}_1/\tilde{y}\}$.

Further, the propagator c_k actives the next trio with the values $\widetilde{B}_2$ such that $\widetilde{W}_2 \bowtie \widetilde{B}_2$: as $\widetilde{W}_2$ denotes previously received values, we take a context of $\bowtie$-related values. Again, received values could be either trigger and characteristic values (required to be observed by MST bisimilarity, cf. Definition 4.4.9) or pure values originated from internal actions. Again, by $\bowtie$ (Definition 4.4.13) we account for both cases.

In sub-case (ii), when u_i is a tail-recursive name, the elements are as follows:

$$\begin{aligned}&\{c^u!\langle M^{\tilde{B}_2}_{V_2}\rangle \mid \mathcal{B}^k_{\tilde{w}}(Q),\ M^{\tilde{B}_2}_{V_2}\,\widetilde{u} \mid \mathcal{B}^k_{\tilde{w}}(Q),\ u_{[S\rangle}!\langle V_2\rangle.(\overline{c_k}!\langle \widetilde{B}_2\rangle \mid c^u?(x).x\,\widetilde{u}) \mid \mathcal{B}^k_{\tilde{w}}(Q)\\ &\qquad : V_1\{\tilde{W}_1/\tilde{y}\} \bowtie V_2,\ \widetilde{W_2} \bowtie \widetilde{B}_2\}\\ &\text{where } M^{\tilde{B}}_V = \lambda\widetilde{z}.\, z_{[S\rangle}!\langle V\rangle.(\overline{c_k}!\langle\widetilde{B}\rangle \mid c^u?(x).x\,\widetilde{z})\end{aligned}$$

The first element is a process obtained by the activation from the preceding trio. The second element is a result of a communication of the first element with top-level provider $\mathcal{R}_{u_i}$ (Definition 4.4.16) on channel c^u. By this synchronization, the decomposition of recursive name u, that is $\widetilde{u}$, is gathered in application $M^{\tilde{B}_2}_{V_2}\,\widetilde{u}$. Finally, the third element represents the result of the application: it is a process ready to mimic the original output action on $u_{[S\rangle}$. Differently from sub-case (i), here we do not have to increment index of u_i in Q and V_1 as indices of recursive names are obtained based on the type S, that is $[S\rangle$.

Input: The $\mathcal{J}$-set of $u_i?(y).Q$ depends on whether (i) u_i is linear or shared name (i.e., $\neg\mathsf{tr}(u_i)$) or (ii) u_i is a tail-recursive name (i.e., $\mathsf{tr}(u_i)$). In both sub-cases $\mathcal{J}$-set is defined similarly to the output case, with only one caveat: we need to expand the context for the continuation with a newly received value y. The $\mathcal{J}$-set in sub-case (i) is defined as follows:

$$\{u_i?(y).\overline{c_k}!\langle\widetilde{B}y\rangle \mid \mathcal{B}^k_{\tilde{x}y}(Q\sigma) : \widetilde{W} \bowtie \widetilde{B}\}$$

where $\sigma = \mathsf{next}(u_i)$. The $\mathcal{J}$-set in sub-case (ii) is defined as follows:

$$\{c^u!\langle M^{\tilde{B}}_y\rangle \mid \mathcal{B}^k_{\tilde{x}y}(Q),\ M^{\tilde{B}}_y\,\widetilde{u} \mid \mathcal{B}^k_{\tilde{x}y}(Q),\ u_{[S\rangle}?(y).(\overline{c_k}!\langle\widetilde{B}y\rangle \mid c^u?(x).x\,\widetilde{z}) \mid \mathcal{B}^k_{\tilde{x}y}(Q) : \widetilde{W} \bowtie \widetilde{B}\}$$
where:
$$M^{\tilde{B}}_y = \lambda\widetilde{z}.\, z_{[S\rangle}?(y).(\overline{c_k}!\langle\widetilde{B}y\rangle \mid c^u?(x).x\,\widetilde{z}).$$

The elements of the set represent steps of obtaining name $u_{[S\rangle}$, along which the original action is mimicked, by synchronizing with the top-level provider $\mathcal{R}_{u_i}$, obtained in the corresponding $\mathcal{C}$-set.

Application: The $\mathcal{J}$-set of $V_1\,(\widetilde{r}, u_i)$ where $\widetilde{r}$ are tail-recursive names, is a union of two sets as follows:

$$\begin{aligned}&\{\,\overbrace{c^{r_l}!\langle\lambda\widetilde{z}_l.c^{r_{l+1}}!\langle\lambda\widetilde{z}_{l+1}.\cdots.c^{r_n}!\langle\lambda\widetilde{z}_n.}^{|\tilde{r}|-l+1}Q_l\rangle\,\rangle\rangle,\\ &\quad(\lambda\widetilde{z}_l.\,\overbrace{c^{r_{l+1}}!\langle\lambda\widetilde{z}_{l+1}.\cdots.c^{r_n}!\langle\lambda\widetilde{z}_n.}^{|\tilde{r}|-l}Q_l\rangle\rangle)\,\widetilde{r}_l : 1 \leq l \leq n,\quad V_1\{\tilde{W}/\tilde{x}\} \bowtie V_2\}\\ &\cup\{V_2\,\widetilde{r}_1, \ldots, \widetilde{r}_n, \widetilde{m} : V_1\{\tilde{W}/\tilde{x}\} \bowtie V_2\}\\ &\text{where:}\\ &\qquad Q_l = V_2\,(\widetilde{r}_1, \ldots, \widetilde{r}_{l-1}, \widetilde{z}_l, \ldots, \widetilde{z}_n, \widetilde{m})\end{aligned}$$

The first set contains intermediate processes emerging while collecting recursive names using synchronizations with recursive name providers. We can see that the body of the inner-most abstraction, Q_l, is an application of V_2 (such that $V_1\{\tilde{W}/\tilde{y}\} \bowtie V_2$) to partially instantiated recursive names: l denotes that decompositions of first $l-1$ recursive names are retrieved. The final tuple in arguments of Q_l, $\widetilde{m} = (u_i, \dots, u_{i+|\mathcal{G}(C)|-1})$, is a full decomposition of non-recursive (linear or shared) name u_i. Just like in the previous cases, by taking V_2 as a $\bowtie$-related value to $V_1\{\tilde{W}/\tilde{y}\}$, we uniformly handle all the three possibilities for V_1 (pure value, variable, and characteristic value).

In the first set, the first element is a process is ready to send an abstraction to an appropriate name provider, in order to retrieve the decomposition of l-th recursive name. The second element is a process that results from a communication of the first element with a provider: an application which will instantiate l-th recursive name in Q_l. Finally, the second set contains application processes in which the decompositions of all n recursive names are gathered, and it is ready to mimic the silent action (application reduction) of the original process.

Parallel composition: The $\mathcal{J}$-set of $Q_1 \mid Q_2$ is defined using two sets:

$$\begin{aligned}&\{\overline{c_k}!\langle \widetilde{B}_1\rangle.\overline{c_{k+l}}!\langle \widetilde{B}_2\rangle \mid \mathcal{B}^k_{\tilde{y}}(Q_1) \mid \mathcal{B}^{k+l}_{\tilde{z}}(Q_2) : \widetilde{W}_1 \bowtie \widetilde{B}_1, \widetilde{W}_2 \bowtie \widetilde{B}_2\}\\ &\cup\\ &\{(R_1 \mid R_2) : R_1 \in \mathcal{C}^{\widetilde{W}_1}_{\tilde{y}}(Q_1), R_2 \in \mathcal{C}^{\widetilde{W}_2}_{\tilde{z}}(Q_2)\}\end{aligned}$$

The first set contains a control trio that is ready to activate the decomposition of the two components in parallel. Just like in the other cases, the control trio propagates values that are $\bowtie$-related to $\widetilde{W}_1$ and $\widetilde{W}_2$. In order to close the set with respect to the τ-actions on propagators, the second set contains the composition of processes drawn from the $\mathcal{C}$-sets of Q_1 and Q_2, with appropriate substitutions.

4.4.4 Proving Operational Correspondence

Recall that we aim to establish Theorem 4.4.1. To that end, we prove that $\mathcal{S}$ (Definition 4.4.17) is an MST bisimulation, by establishing two results:

- Lemma 4.4.6 covers the case in which the given process performs an action, which is matched by an action of the decomposed process. In terms of operational correspondence (see, e.g., [29]), this establishes *completeness* of the decomposition.

- Lemma 4.4.7 covers the converse direction, in which the decomposed process performs an action, which is matched by the initial process. This established the *soundness* of the decomposition.

For proving both operational completeness and soundness, we will need the following result. Following Parrow [50], we refer to prefixes that do not correspond to prefixes of the original process, i.e. prefixes on propagators c_i, as *non-essential prefixes*. Then the relation $\mathcal{S}$ is closed under reductions that involve non-essential prefixes.

Lemma 4.4.3. *Given an indexed process $P_1\{\tilde{W}/\tilde{x}\}$, the set $\mathcal{C}^{\tilde{W}}_{\tilde{x}}(P_1)$ is closed under τ-transitions on non-essential prefixes. That is, if $R_1 \in \mathcal{C}^{\tilde{W}}_{\tilde{x}}(P_1)$ and $R_1 \xrightarrow{\tau} R_2$ is inferred from the actions on non-essential prefixes, then $R_2 \in \mathcal{C}^{\tilde{W}}_{\tilde{x}}(P_1)$.*

Proof. By the induction on the structure of P_1. See Appendix A.2.1 for more details. □

Operational Completeness. We first consider transitions using the unrestricted and untyped LTS; in Lemma 4.4.6 we will consider transitions with the refined LTS.

Lemma 4.4.4. *Assume $P_1\{\tilde{W}/\tilde{x}\}$ is a process such that $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \triangleright \diamond$ with $\mathsf{balanced}(\Delta_1)$ and $P_1\{\tilde{W}/\tilde{x}\} \, \mathcal{S} \, Q_1$.*

1. *Whenever $P_1\{\tilde{W}/\tilde{x}\} \xrightarrow{(\nu\, \widetilde{m}_1)\, n!\langle V_1\rangle} P_2$, such that $\overline{n} \notin \mathtt{fn}(P_1\{\tilde{W}/\tilde{x}\})$, then there exist Q_2 and V_2 such that $Q_1 \xRightarrow{(\nu\, \widetilde{m}_2)\, \breve{n}!\langle V_2\rangle} Q_2$ and, for a fresh t,*

$$(\nu\, \widetilde{m}_1)(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{\tilde{W}/\tilde{x}\} \, \mathcal{S} \, (\nu\, \widetilde{m}_2)(Q_2 \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2)$$

2. *Whenever $P_1\{\tilde{W}/\tilde{x}\} \xrightarrow{n?(V_1)} P_2$, such that $\overline{n} \notin \mathtt{fn}(P_1\{\tilde{W}/\tilde{x}\})$, then there exist Q_2, V_2, and σ such that $Q_1 \xRightarrow{\breve{n}?(V_2)} Q_2$ where $V_1\sigma \bowtie V_2$ and $P_2 \, \mathcal{S} \, Q_2$,*

3. *Whenever $P_1 \xrightarrow{\tau} P_2$ then there exists Q_2 such that $Q_1 \xRightarrow{\tau} Q_2$ and $P_2 \, \mathcal{S} \, Q_2$.*

Proof. By transition induction. See Appendix A.2.2 for more details. □

The following statement builds upon the previous one to address the case of the typed LTS (Definition 4.4.5):

Lemma 4.4.5. *Assume $P_1\{\tilde{W}/\tilde{x}\}$ is a process and $P_1\{\tilde{W}/\tilde{x}\} \, \mathcal{S} \, Q_1$.*

1. *Whenever $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \xrightarrow{(\nu\, \widetilde{m_1})\, n!\langle V_1\rangle} \Lambda_1'; \Delta_1' \vdash P_2$ then there exist Q_2, V_2, Δ_2', and Λ_2' such that $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \xRightarrow{(\nu\, \widetilde{m_2})\, \breve{n}!\langle V_2\rangle} \Lambda_2'; \Delta_2' \vdash Q_2$ and, for a fresh t,*

$$(\nu\, \widetilde{m_1})(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{\tilde{W}/\tilde{x}\} \, \mathcal{S} \, (\nu\, \widetilde{m_2})(Q_2 \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2)$$

2. *Whenever $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \xrightarrow{n?(V_1)} \Lambda_1'; \Delta_1' \vdash P_2$ then there exist Q_2, V_2, σ, Λ_2', and Δ_2' such that $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \xRightarrow{\breve{n}?(V_2)} \Lambda_2', \Delta_2' \vdash Q_2$ where $V_1\sigma \bowtie V_2$ and $P_2 \, \mathcal{S} \, Q_2$,*

3. *Whenever $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \xrightarrow{\tau} \Lambda_1'; \Delta_1' \vdash P_2$ then there exist Q_2, Λ_2', and Δ_2' such that $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \xRightarrow{\tau} \Lambda_2'; \Delta_2' \vdash Q_2$ and $P_2 \, \mathcal{S} \, Q_2$.*

Proof. The proof uses results of Lemma 4.4.4. We consider the first case, the other two being similar.

By the definition of the typed LTS we have:

$$\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \tag{4.19}$$

$$(\Gamma_1; \emptyset; \Delta_1) \xrightarrow{(\nu\, \widetilde{m})\, n!\langle V\rangle} (\Gamma_1; \emptyset; \Delta_2) \tag{4.20}$$

By (4.20) we further have

$$[\text{SSnd}] \; \frac{\begin{array}{ccc} \Gamma, \Gamma'; \Lambda'; \Delta' \vdash V \triangleright U & \Gamma'; \emptyset; \Delta_j \vdash m_j \triangleright U_j & \overline{n} \notin \mathtt{dom}(\Delta) \\ \Delta' \backslash (\cup_j \Delta_j) \subseteq (\Delta, n : S) & \Gamma'; \emptyset; \Delta_j' \vdash \overline{m}_j \triangleright U_j' & \Lambda' \subseteq \Lambda \end{array}}{(\Gamma; \Lambda; \Delta, s :!\langle U\rangle;S) \xrightarrow{(\nu\, \widetilde{m})\, n!\langle V\rangle} (\Gamma, \Gamma'; \Lambda \backslash \Lambda'; (\Delta, n : S, \cup_j \Delta_j') \backslash \Delta')}$$

By (4.19) and the condition $\overline{n} \notin \mathtt{dom}(\Delta)$ we have $\overline{n} \notin \mathtt{fn}(P_1\{\tilde{W}/\tilde{x}\})$. Therefore, we can apply Item 1 of Lemma 4.4.4. □

Finally, we are in a position to address the case of the refined typed LTS (Definition 4.4.5):

Lemma 4.4.6. *Assume $P_1\{\tilde{W}/\tilde{x}\}$ is a process and $P_1\{\tilde{W}/\tilde{x}\} \mathcal{S} Q_1$.*

1. *Whenever $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \xmapsto{(\nu\, \widetilde{m_1})\, n!\langle V_1\rangle} \Lambda_1'; \Delta_1' \vdash P_2$ then there exist Q_2, V_2, Δ_2', and Λ_2' such that $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \stackrel{(\nu\, \widetilde{m_2})\, \breve{n}!\langle V_2\rangle}{\Longmapsto}_{m} \Lambda_2'; \Delta_2' \vdash Q_2$ and, for a fresh t,*
$$(\nu\, \widetilde{m_1})(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{\tilde{W}/\tilde{x}\} \mathcal{S} (\nu\, \widetilde{m_2})(Q_2 \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2)$$
2. *Whenever $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \xmapsto{n?(V_1)} \Lambda_1'; \Delta_1' \vdash P_2$ then there exist Q_2, V_2, Λ_2', and Δ_2' such that $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \stackrel{\breve{n}?(V_2)}{\Longmapsto}_{m} \Lambda_2', \Delta_2' \vdash Q_2$ where $V_1 \bowtie_{\mathtt{c}} V_2$ and $P_2 \mathcal{S} Q_2$,*
3. *Whenever $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \xmapsto{\tau} \Lambda_1'; \Delta_1' \vdash P_2$ then there exist Q_2, Λ_2', and Δ_2' such that $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \stackrel{\tau}{\Longmapsto}_{m} \Lambda_2'; \Delta_2' \vdash Q_2$ and $P_2 \mathcal{S} Q_2$.*

Proof. By case analysis of the transition label ℓ. It uses results of Lemma 4.4.5. We consider two cases: (i) $\ell \equiv n?(V_1)$ and (ii) $\ell \not\equiv n?(V_1)$.

(i) Case $\ell \equiv n?(V_1)$. This case concerns Part (2) of the lemma. In this case we know $P_1 = n?(y).Q$. We have the following transition inference tree:

$$\langle\mathtt{Rv}\rangle \frac{}{(n?(y).P_2)\{\tilde{W}/\tilde{x}\} \xrightarrow{n?(V_1)} P_2} \tag{4.21}$$

$$\langle\mathtt{RRcv}\rangle \frac{(4.21) \qquad V_1 \equiv [\![U]\!]_{\mathtt{c}} \vee V_1 \equiv \lambda x.\, t?(y).(y\, x)\ t \text{ fresh}}{(n?(y).P_2)\{\tilde{W}/\tilde{x}\} \xmapsto{n?(V_1)} P_2} \tag{4.22}$$

From (4.21) and Lemma 4.4.5 we know that there exist Q_2, and V_2 such that $Q_1 \stackrel{\breve{n}?(V_2)}{\Longrightarrow} Q_2$ and $P_2 \mathcal{S} Q_2$ where $V_1\sigma \bowtie V_2$. Since V_1 is a characteristic or a trigger value, we have $V_1 \bowtie_{\mathtt{c}} V_2$ and that V_2 is a minimal characteristic or a trigger value. Hence, $Q_1 \stackrel{\breve{n}?(V_2)}{\Longmapsto}_{\mathtt{m}} Q_2$ using the Rule MTR (Definition 4.4.5).

- Case $\ell \not\equiv n?(V)$. This case concerns Parts (1) and (3) of the lemma. We only consider the first part, when $\ell \equiv (\nu\, \widetilde{m_1})\, n!\langle V_1\rangle$, since the other part is similar.

  We apply Lemma 4.4.4 to obtain Q_2 such that $Q_1 \stackrel{(\nu\, \widetilde{m_2})\, \breve{n}!\langle V_2\rangle}{\Longmapsto} Q_2$, and, for a fresh t,
$$(\nu\, \widetilde{m_1})(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{\tilde{W}/\tilde{x}\}\ \mathcal{S}\ (\nu\, \widetilde{m_2})(Q_2 \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2).$$
  Since we are dealing with an output action, we can immediately conclude that $Q_1 \stackrel{(\nu\, \widetilde{m_2})\, \breve{n}!\langle V_2\rangle}{\Longmapsto}_{\mathtt{m}} Q_2$.

□

Operational Soundness. For the proof of operational soundness we follow the same strategy of stratifying it into three lemmas.

Lemma 4.4.7. *Assume $P_1\{\tilde{W}/\tilde{x}\}$ is a process and $P_1\{\tilde{W}/\tilde{x}\} \mathcal{S} Q_1$.*

1. *Whenever $Q_1 \xrightarrow{(\nu\, \widetilde{m_2})\, n_i!\langle V_2\rangle} Q_2$, such that $\overline{n}_i \notin \mathtt{fn}(Q_1)$, then there exist P_2 and V_2 such that $P_1\{\tilde{W}/\tilde{x}\} \xrightarrow{(\nu\, \widetilde{m_2})\, n!\langle V_2\rangle} P_2$ and, for a fresh t,*
$$(\nu\, \widetilde{m_1})(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{\tilde{W}/\tilde{x}\} \mathcal{S} (\nu\, \widetilde{m_2})(Q_2 \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2).$$

2. *Whenever* $Q_1 \xrightarrow{n_i?\langle V_2\rangle} Q_2$ *, such that* $\overline{n_i} \notin \mathtt{fn}(Q_1)$*, there exist* P_2*,* V_2*, and* σ *such that* $P_1\{\tilde{W}/\tilde{x}\} \xrightarrow{n?\langle V_1\rangle} P_2$ *where* $V_1\sigma \bowtie V_2$ *and* $P_2 \,\mathcal{S}\, Q_2$*.*

3. *Whenever* $Q_1 \xrightarrow{\tau} Q_2$ *either (i)* $P_1\{\tilde{W}/\tilde{x}\} \,\mathcal{S}\, Q_2$ *or (ii) there exists* P_2 *such that* $P_1 \xrightarrow{\tau} P_2$ *and* $P_2 \,\mathcal{S}\, Q_2$*.*

Proof (Sketch). By transition induction. See Appendix A.2.3 for more details. □

Lemma 4.4.8. *Assume* $P_1\{\tilde{W}/\tilde{x}\}$ *is a process and* $P_1\{\tilde{W}/\tilde{x}\} \,\mathcal{S}\, Q_1$*.*

1. *Whenever* $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \xrightarrow{(\nu\,\widetilde{m_2})\,\breve{n}!\langle V_2\rangle} \Lambda_2'; \Delta_2' \vdash Q_2$ *then there exist* P_2*,* V_1*,* Δ_1'*, and* Λ_1' *such that* $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \xRightarrow{(\nu\,\widetilde{m_1})\,n!\langle V_1\rangle} \Lambda_1'; \Delta_1' \vdash P_2$ *and, for a fresh* t*,*

$$(\nu\,\widetilde{m_1})(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{\tilde{W}/\tilde{x}\} \,\mathcal{S}\, (\nu\,\widetilde{m_2})(Q_2 \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2)$$

2. *Whenever* $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \xrightarrow{\breve{n}?\langle V_2\rangle} \Lambda_2'; \Delta_2' \vdash Q_2$ *then there exist* P_2*,* V_1*,* σ*,* Λ_1'*, and* Δ_1' *such that* $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \xRightarrow{n?\langle V_1\rangle} \Lambda_1', \Delta_1' \vdash P_2$ *where* $V_1\sigma \bowtie V_2$ *and* $P_2 \,\mathcal{S}\, Q_2$*,*

3. *Whenever* $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \xrightarrow{\tau} \Lambda_2'; \Delta_2' \vdash Q_2$ *then either (i)* $P_1\{\tilde{W}/\tilde{x}\} \,\mathcal{S}\, Q_2$ *or (ii) there exist* P_2*,* Λ_1'*, and* Δ_1' *such that* $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \xrightarrow{\tau} \Lambda_1'; \Delta_1' \vdash P_2$ *and* $P_2 \,\mathcal{S}\, Q_2$*.*

Lemma 4.4.9. *Assume* $P_1\{\tilde{W}/\tilde{x}\}$ *is a process and* $P_1\{\tilde{W}/\tilde{x}\} \,\mathcal{S}\, Q_1$*.*

1. *Whenever* $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \xmapsto{(\nu\,\widetilde{m_2})\,\breve{n}!\langle V_2\rangle} \Lambda_2'; \Delta_2' \vdash Q_2$ *then there exist* P_2*,* V_1*,* Δ_1'*, and* Λ_1' *such that* $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \overset{(\nu\,\widetilde{m_1})\,n!\langle V_1\rangle}{\Longmapsto} \Lambda_1'; \Delta_1' \vdash P_2$ *and, for a fresh* t*,*

$$(\nu\,\widetilde{m_1})(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{\tilde{W}/\tilde{x}\} \,\mathcal{S}\, (\nu\,\widetilde{m_2})(Q_2 \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2)$$

2. *Whenever* $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \xmapsto{\breve{n}?\langle V_2\rangle} \Lambda_2'; \Delta_2' \vdash Q_2$ *then there exist* P_2*,* V_1*,*Λ_1'*, and* Δ_1' *such that* $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \overset{n?\langle V_1\rangle}{\Longmapsto} \Lambda_1', \Delta_1' \vdash P_2$ *where* $V_1 \bowtie_{\mathtt{c}} V_2$ *and* $P_2 \,\mathcal{S}\, Q_2$*,*

3. *Whenever* $\Gamma_2; \Lambda_2; \Delta_2 \vdash Q_1 \xmapsto{\tau} \Lambda_2'; \Delta_2' \vdash Q_2$ *then either (i)* $P_1\{\tilde{W}/\tilde{x}\} \,\mathcal{S}\, Q_2$ *or (ii) there exist* P_2*,* Λ_1'*, and* Δ_1' *such that* $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{W}/\tilde{x}\} \xmapsto{\tau} \Lambda_1'; \Delta_1' \vdash P_2$ *and* $P_2 \,\mathcal{S}\, Q_2$*.*

Summary. Together, Lemma 4.4.6 and Lemma 4.4.7 imply that $\mathcal{S}$ is an MST-bisimilarity. In summary, we have shown Theorem 4.4.1, i.e., that for any typed process P, we have that

$$\Gamma; \Lambda; \Delta \vdash P \;\approx^{\mathtt{M}}\; \mathcal{G}(\Gamma); \mathcal{G}(\Lambda); \mathcal{G}(\Delta) \vdash \mathcal{D}(P).$$

In this section we have defined a notion of MST bisimilarity, following the notion HO bisimilarity for non-minimal processes. Following the strategy of Parrow in the untyped setting, we defined a relation $\mathcal{S}$ containing all pairs $(P, \mathcal{D}(P))$, which we proved to be an MST bisimulation.

4.5 Optimizations of the Decomposition

In this section we discuss two optimizations that can be applied to the decomposition process. These optimizations simplify the structure of the trios and the nature of the underlying communication discipline.

The first optimization replaces trios in the decomposition with *duos* (i.e., processes with at most two sequential prefixes). The decomposition in Section 4.3 follows Parrow's approach in that it converts a process into a parallel composition of trios. The use of trios seems to be necessary in (plain) π-calculus; in our first optimization we show that, by exploiting the higher-order nature of communications in HO, the trios can be replaced by duos.

The second optimization replaces polyadic communications (sending and receiving several values at once) with monadic communications (sending and receiving only a single value per prefix). In the decomposition, we use polyadic communications in order to propagate dependencies through sub-processes. We show that the use of monadic communication prefixes is sufficient for that task.

From Trios to Duos. In the first optimization we replace trios with *duos*, i.e., processes with at most two sequential prefixes. This optimization is enabled by the higher-order nature of HO. In the translation we make of *thunk processes*, i.e., inactive processes that can be activated upon reception. We write $\{\{P\}\}$ to stand for the thunk process $\lambda x : \langle \mathtt{end} \!\rightarrow\! \diamond \rangle. P$, for a fresh $x \notin \mathtt{fn}(P)$. We write $\mathtt{run}\,\{\{P\}\}$ to denote the application of a thunk to a (dummy) name of type $\mathtt{end} \!\rightarrow\! \diamond$. This way, we have a reduction $\mathtt{run}\,\{\{P\}\} \longrightarrow P$.

The key idea behind replacing trios with duos is to transform a trio like

$$c_k?(\widetilde{x}).u!\langle V \rangle.c_{k+1}!\langle \widetilde{z} \rangle$$

into the composition of two duos, the second one being a "control" duo:

$$c_k?(\widetilde{x}).c_0!\langle \{\{u!\langle V \rangle.c_{k+1}!\langle \widetilde{z} \rangle\}\} \rangle \mid c_0?(y).(\mathtt{run}\,y) \tag{4.23}$$

The first action (on c_k) is as before; the two remaining prefixes (on u and c_{k+1}) are encapsulated into a thunk. This thunk is sent via an additional propagator (denoted c_0) to the control duo that activates it upon reception. Because of this additional propagator, this transformation involves minor modifications in the definition of the degree function $\wr - \int$ (cf. Definition 4.3.6).

In some cases, the breakdown function in Section 4.3.2 already produces duos. Breaking down input and output prefixes and parallel composition involves proper trios; following the scheme illustrated by (4.23), we can define a map $\{\!| - |\!\}$ to transform these trios into duos:

$$\begin{aligned}
\{\!| c_k?(\widetilde{x}).u_i!\langle V \rangle.\overline{c_{k+1}}!\langle \widetilde{z} \rangle |\!\} &= c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle \{\{u_i!\langle V \rangle.\overline{c_{k+2}}!\langle \widetilde{z} \rangle\}\} \rangle \mid c_{k+1}?(y).(\mathtt{run}\,y) \\
\{\!| c_k?(\widetilde{x}).u_i?(y).\overline{c_{k+1}}!\langle \widetilde{x}' \rangle |\!\} &= c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle \{\{u_i?(y).\overline{c_{k+2}}!\langle \widetilde{x}' \rangle\}\} \rangle \mid c_{k+1}?(y).(\mathtt{run}\,y) \\
\{\!| c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle \widetilde{y} \rangle.\overline{c_{k+l+1}}!\langle \widetilde{z} \rangle |\!\} &= c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle \{\{\overline{c_{k+2}}!\langle \widetilde{y} \rangle.\overline{c_{k+l+2}}!\langle \widetilde{z} \rangle\}\} \rangle \mid c_{k+1}?(y).(\mathtt{run}\,y)
\end{aligned}$$

In breaking down prefixes involving tail-recursive names (Table 4.1) we encounter trios of the following form:

$$\mathcal{B}(u_i?(y).Q) = c_k?(\widetilde{x}).c^u!\langle N_y \rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(Q) \quad \text{where} \quad N_y = \lambda \widetilde{z}.\, z_{\lceil S \rangle}?(y).(\overline{c_{k+1}}!\langle \widetilde{w} \rangle \mid c^u?(x).x\,\widetilde{z}))$$

Here we can see that the top-level process is a duo and that only N_y packs a proper trio process. By applying the same idea we can translate this trio into the following composition of duos:

$$\{\!| z_{\lceil S \rangle}?(y).(\overline{c_{k+1}}!\langle \widetilde{w} \rangle \mid c^u?(x).x\,\widetilde{z}) |\!\} = z_{\lceil S \rangle}?(y).\overline{c_{k+1}}!\langle \{\{\overline{c_{k+2}}!\langle \widetilde{w} \rangle \mid c^u?(x).x\,\widetilde{z}\}\} \rangle \mid c_{k+1}?(y).\mathtt{run}\,y$$

This is the idea behind the breakdown of a process starting with an input prefix; the breakdown of a process with an output prefix follows the same lines.

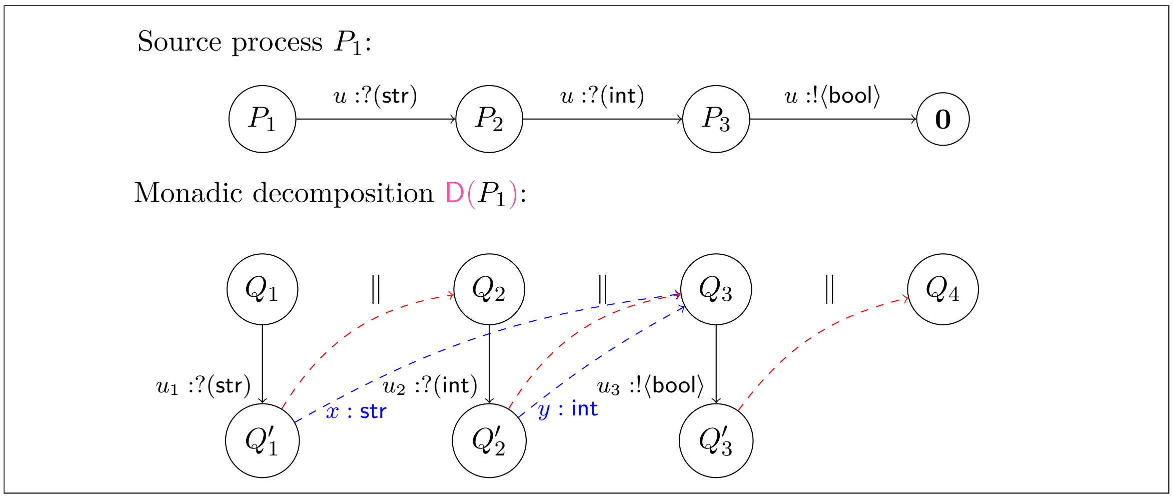


Figure 4.6: Our monadic decomposition function $\mathsf{D}(-)$, illustrated. As in Figure 4.2, nodes represent process states, '‖' represents parallel composition of processes, black arrows stand for actions, and red arrows indicate synchronizations that preserve the sequentiality of the source process; also, blue arrows indicates synchronizations that propagate (bound) values.

From Polyadic to Monadic Communication. Our second optimization replaces polyadic communications, used for the propagators, with monadic communications. Recall that propagators in $\mathcal{B}(-)$ serve two purposes: they (i) encode sequentiality by properly activating trios and (ii) propagate bound values. By separating propagators along those two roles, we can we can dispense with polyadic communication in the breakdown function.

We define *monadic breakdown*, $\mathsf{B}(-)$, and *monadic decomposition*, $\mathsf{D}(-)$, which use two *kinds* of propagators: (i) propagators for only activating trios of form c_k (where $k > 0$ is an index) and (ii) for propagating bound values of form c_x (where x is some variable). We depict the mechanism of the monadic breakdown in Figure 4.6. The main idea is to establish a direct link between trio that binds the variable x and trios that make use of x on propagator channel c_x. Thus, propagators on c_k only serve to activate next trios: they do so by *receiving* an abstraction that contains the next trio.

Formally, we define a *monadic decomposition*, $\mathsf{D}(P)$, that simplifies Definition 4.3.9 as follows:

$$\mathsf{D}(P) = (\nu\,\widetilde{c})\,(\overline{c_k}!\langle\rangle \mid \mathsf{B}^k(P\sigma))$$

where $k > 0$, $\widetilde{c} = (c_k, \dots, c_{k+\wr P \wr -1})$, and the initializing substitution $\sigma = \{\mathsf{index}(\widetilde{u})/\widetilde{u}\}$ is the same as in Definition 4.3.9.

The monadic break down function $\mathsf{B}^k(-)$, given in Figure 4.7, simplifies the one in Table 4.1 by using only one parameter, namely k. In Figure 4.7 we use σ to denote the subsequent substitution $\mathsf{next}(u_i)$, the same as in Table 4.1, and use $\widetilde{m}$ to denote the breakdown $(u_i, \dots, u_{i+|\mathcal{G}(C)|-1})$ of the name u_i.

The breakdown function $\mathsf{B}^k(-)$ uses propagators c_k $(k > 0)$ for encoding sequentiality and dedicated propagators c_x for each variable x. As propagators c_k now only serve to encode sequentiality, only dummy values are being communicated along these channels (see Remark 4).

Let us describe the breakdown of a process with an input prefix, as it illustrates the key points common to all the other cases. The breakdown $\mathsf{B}^k(u_i?(x).Q)$ consists of a trio in parallel with the breakdown of the continuation $\mathsf{B}^{k+1}(Q\sigma)$ with name c_x restricted. The trio is first activated on c_k. This is followed by the prefix that mimics original input action on

$$
\begin{aligned}
V_x &= \lambda y.\, y?(z).c_x!\langle x\rangle.(\nu\, s)\,(z\, s \mid \overline{s}!\langle z\rangle)\\
W_x^{\leadsto} &= \begin{cases} \overline{c_x}!\langle x\rangle & \text{if } \leadsto = \multimap \\ (\nu\, s)\,(V_x\, s \mid \overline{s}!\langle V_x\rangle) & \text{if } \leadsto = \rightarrow \end{cases}\\
\mathsf{B}^k(u_i?(x : C \leadsto \diamond).Q) &= (\nu\, c_x)\,(c_k?().u_i?(x).(\overline{c_{k+1}}!\langle\rangle \mid W_x^{\leadsto})) \mid \mathsf{B}^{k+1}(Q\sigma)\\
\mathsf{B}^k(u_i!\langle x\rangle.Q) &= c_k?().c_x?(x).u_i!\langle x\rangle.\overline{c_{k+1}}!\langle\rangle \mid \mathsf{B}^{k+1}(Q\sigma)\\
\mathsf{B}^k(u_i!\langle V\rangle.Q) &= c_k?().u_i!\langle \mathsf{V}(V\sigma)\rangle.\overline{c_{k+1}}!\langle\rangle \mid \mathsf{B}^{k+1}(Q\sigma)\\
\mathsf{B}^k(x\, u_i) &= c_k?().c_x?(x).x\, \widetilde{m}\\
\mathsf{B}^k(V\, u_i) &= c_k?().\mathsf{V}(V)\, \widetilde{m}\\
\mathsf{B}^k((\nu\, s)\, P') &= (\nu\, \widetilde{s})\, \mathsf{B}^k(P'\sigma)\\
\mathsf{B}^k(Q \mid R) &= c_k?().\overline{c_{k+1}}!\langle\rangle.\overline{c_{k+\wr Q \int+1}}!\langle\rangle \mid \mathsf{B}^{k+1}(Q) \mid \mathsf{B}^{k+\wr Q\int+1}(R)\\
\mathsf{V}(y) &= y\\
\mathsf{V}(\lambda y : C^{\leadsto}.\, P) &= \lambda(y_1,\dots,y_{|\mathcal{G}(C)|}) : \mathcal{G}(C)^{\leadsto}.\,(\nu\, c_1,\dots,c_{\wr P\int})\,(\overline{c_1}!\langle\rangle \mid \mathsf{B}^1(P\{y_1/y\}))
\end{aligned}
$$

Figure 4.7: Monadic breakdown of processes and values

indexed name u_i. Upon receiving value x, two things will happen in parallel. First, the next trio will be activated on name c_{k+1}. Second, the value x received on u_i is propagated further by the dedicated process $W_x^{\leadsto}$.

The specific mechanism of propagation depends on whether a received value is linear ($\leadsto = \multimap$) or shared ($\leadsto = \rightarrow$). In the former case, we simply propagate a value along the *linear* name c_x once. In the later case, we cannot propagate the value only once, because a shared variable can be used in multiple trios. Thus, $W_x^{\rightarrow}$ implements a recursive mechanism that repeatedly sends a value on the *shared* name c_x. The recursion is encoded in the same way as in Example 4.3.4: action $c_x!\langle x\rangle$ is enclosed in value V that gets appropriately duplicated upon a synchronization.

The breakdown function for values, $\mathsf{V}(-)$, is accordingly changed to invoke $\mathsf{B}^1(-)$ for breaking down a function body.

For simplicity, we defined the decomposition of the output process using a subprocess with four prefixes. Alternatively, we could have used a decomposition that relies on two trios, by introducing abstraction passing as in the previous section.

Let us illustrate the monadic breakdown by the means of an example:

Example 4.5.1 (Monadic Decomposition)**.** We again consider process $P = (\nu\, u)\,(Q \mid R)$ as in Example 4.3.7 where:

$$
\begin{aligned}
Q &= u?(x).\,\overbrace{u?(y).(\nu\, s)\,(x\,\overline{s} \mid s!\langle y\rangle)}^{Q'}\\
R &= \overline{u}!\langle V\rangle.\overline{u}!\langle \mathsf{true}\rangle.\mathbf{0}\\
V &= \lambda z.\, z?(w).\mathbf{0}
\end{aligned}
$$

Let us recall the reductions of P:

$$
\begin{aligned}
P &\longrightarrow u?(y).(\nu\, s)\,(V\,\overline{s} \mid s!\langle y\rangle) \mid \overline{u}!\langle \mathsf{true}\rangle.\mathbf{0} \longrightarrow (\nu\, s)\,(V\,\overline{s} \mid s!\langle \mathsf{true}\rangle)\\
&\longrightarrow (\nu\, s)\,(\overline{s}?(w).\mathbf{0} \mid s!\langle \mathsf{true}\rangle) = P'
\end{aligned}
$$

The monadic decomposition of P is as follows:

$$\mathsf{D}(P) = (\nu\, c_1,\dots,c_{10})\,(\nu\, u_1,u_2)\,(\overline{c_1}!\langle\rangle \mid \mathsf{B}^1(P\sigma))$$

where $\sigma = \{u_1\overline{u}_1/u\overline{u}\}$. We have:

$$\mathsf{B}^1(P\sigma) = c_1?().\overline{c_2}!\langle\rangle.\overline{c_8}!\langle\rangle \mid \mathsf{B}^2(Q\sigma) \mid \mathsf{B}^8(R\sigma)$$

where:

$$\begin{aligned}
\mathsf{B}^2(Q\sigma) &= (\nu\, c_x)\,(c_2?().u_1?(x).(\overline{c_3}!\langle\rangle \mid \overline{c_x}!\langle x\rangle) \mid \mathsf{B}^3(Q'\sigma'))\\
\mathsf{B}^3(Q'\sigma') &= (\nu\, c_y)\,(c_3?().u_2?(y).(\overline{c_4}!\langle\rangle \mid W_y) \mid \mathsf{B}^4((\nu\, s)\,(x\,\overline{s} \mid s!\langle y\rangle)))\\
\mathsf{B}^4((\nu\, s)\,(x\,\overline{s} \mid s!\langle y\rangle)) &= (\nu\, s_1)\, c_4?().\overline{c_5}!\langle\rangle.\overline{c_6}!\langle\rangle \mid c_5?().c_x?(x).x\,\overline{s}_1 \mid\\
&\qquad c_6?().c_y?(y).s_1!\langle y\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0}\\
\mathsf{B}^8(R\sigma) &= c_8?().\overline{u}_1!\langle \mathsf{V}(V)\rangle.\overline{c_9}!\langle\rangle \mid \mathsf{B}^9(\overline{u}_2!\langle \mathsf{true}\rangle.\mathbf{0})\\
\mathsf{B}^9(\overline{u}_2!\langle \mathsf{true}\rangle.\mathbf{0}) &= c_9?().\overline{u}_2!\langle \mathsf{true}\rangle.\overline{c_{10}}!\langle\rangle \mid c_{10}?().\mathbf{0}\\
\mathsf{V}(V) &= \lambda z_1.\,(\nu\, c_1^V, c_2^V)\,\overline{c_1^V}!\langle\rangle \mid\\
&\qquad (\nu\, c_w)\, c_1^V?(z).z_1?(w).(\overline{c_2^V}!\langle\rangle \mid W_w) \mid c_2^V?().\mathbf{0}
\end{aligned}$$

where $W_x = (\nu\, s)\,(V_x\, s \mid \overline{s}!\langle V_x\rangle)$ with $V_x = \lambda y.\, y?(z).c_x!\langle x\rangle.(\nu\, s)\,(z\,\overline{s} \mid s!\langle z\rangle)$. We may observe that $\mathsf{D}(P)$ correctly implements u_1 and u_2 typed with MSTs M_1 and M_2 (resp.) as given in Example 4.3.7.

Now, we inspect the reductions of $\mathsf{D}(P)$. First we have three reductions on propagators:

$$\begin{aligned}
\mathsf{D}(P) &\longrightarrow (\nu\, c_2,\dots,c_{10})\,(\nu\, u_1,u_2)\,\overline{c_2}!\langle\rangle.\overline{c_8}!\langle\rangle \mid \mathsf{B}^2(Q\sigma) \mid \mathsf{B}^8(R\sigma)\\
&\longrightarrow^2 (\nu\, c_3,\dots,c_7,c_9,c_{10})\,(\nu\, c_x)\,(\, u_1?(x).\,(\overline{c_3}!\langle\rangle \mid \overline{c_x}!\langle x\rangle) \mid \mathsf{B}^3(Q'\sigma'))\\
&\quad \mid \overline{u}_1!\langle \mathsf{V}(V)\rangle.\,\overline{c_9}!\langle\rangle \mid \mathsf{B}^9(u_2!\langle \mathsf{true}\rangle.\mathbf{0}) = D^1
\end{aligned}$$

Now, the synchronization on u_1 can take a place in D^1 (on the prefixes highlighted above). We can see that value $\mathsf{V}(V)$ received on u_1 can be propagated along c_x to a trio using it. Following up on that, propagators c_3 and c_9 are synchronized.

$$\begin{aligned}
D^1 &\longrightarrow (\nu\, c_3,\dots,c_7,c_9,c_{10})\,(\nu\, c_x)\,(\overline{c_3}!\langle\rangle \mid c_x!\langle \mathsf{V}(V)\rangle \mid\\
&\qquad \mid (\nu\, c_y)\,(c_3?().u_2?(y).(\overline{c_4}!\langle\rangle \mid W_y) \mid \mathsf{B}^4((\nu\, s)\,(x\,\overline{s} \mid s!\langle y\rangle)))) \mid \overline{c_9}!\langle\rangle \mid \mathsf{B}^9(\overline{u}_2!\langle \mathsf{true}\rangle.\mathbf{0})\\
&\longrightarrow^2 (\nu\, c_4,\dots,c_7,c_{10})\,(\nu\, c_x)\,(c_x!\langle \mathsf{V}(V)\rangle \mid\\
&\qquad \mid (\nu\, c_y)\,(\, u_2?(y).\,(\overline{c_4}!\langle\rangle \mid W_y) \mid \mathsf{B}^4((\nu\, s)\,(x\,\overline{s} \mid s!\langle y\rangle)))) \mid \overline{u}_2!\langle \mathsf{true}\rangle.\,\overline{c_{10}}!\langle\rangle \mid c_{10}?().\mathbf{0} = D^2
\end{aligned}$$

Similarly, D^2 can mimic the synchronization on name u_2. Again, this is followed by synchronizations on propagators.

$$\begin{aligned}
D^2 &\longrightarrow (\nu\, c_4,\dots,c_7,c_{10})\,(\nu\, c_x)\,(c_x!\langle \mathsf{V}(V)\rangle \mid (\nu\, c_y)\,(\overline{c_4}!\langle\rangle \mid W_y\{\mathsf{true}/y\} \mid \mathsf{B}^4((\nu\, s)\,(x\,\overline{s} \mid s!\langle y\rangle))))\\
&\qquad \mid \overline{c_{10}}!\langle\rangle \mid c_{10}?().\mathbf{0}\\
&\longrightarrow^4 (\nu\, c_7)\,(\nu\, c_x)\,(c_x!\langle \mathsf{V}(V)\rangle \mid (\nu\, c_y)\,(W_y\{\mathsf{true}/y\} \mid (\nu\, s_1)\, c_x?(x).x\,\overline{s}_1\\
&\qquad \mid c_y?(y).s_1!\langle y\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0})) = D^3
\end{aligned}$$

The subprocess $W_y\{\mathsf{true}/y\}$ is dedicated to providing the value true on a shared name c_y. Specifically, it reduces as follow Its reductions are as follows:

$$W_y\{\mathsf{true}/y\} \longrightarrow^2 c_y!\langle \mathsf{true}\rangle.W_y\{\mathsf{true}/y\}$$

In this example, the shared value received on y is used only once; in the general case, a process could use a shared value multiple times: thus there could be multiple trios requesting the shared value on c_y.

With this information, we have the following reductions of the decomposed process:

$$\begin{aligned} D^3 \longrightarrow^2\ & (\nu\, c_7)\,(\nu\, c_x)\,(c_x!\langle \mathsf{V}(V)\rangle \mid (\nu\, c_y)\,(c_y!\langle \mathsf{true}\rangle.W_y\{\mathsf{true}/y\} \mid (\nu\, s_1)\, c_x?(x).x\,\overline{s}_1 \\ & \mid c_y?(y).s_1!\langle y\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0})) = D^4 \end{aligned}$$

In D^4 a value for x is requested on name c_x before it is applied to name $\overline{s}_1$. Similarly, a value for y is gathered by the communication on c_y. These values are retrieved in two reductions steps as follows:

$$D^4 \longrightarrow^2 (\nu\, c_7)\,(\nu\, s_1)\,\mathsf{V}(V)\,\overline{s}_1 \mid s_1!\langle \mathsf{true}\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0} \mid (\nu\, c_y)\,W_y\{\mathsf{true}/y\} = D^5$$

We remark that $(\nu\, c_y)\,W_y\{\mathsf{true}/y\}$ reduces to $(\nu\, c_y)\,\overline{c_y}!\langle \mathsf{true}\rangle.W_y\{\mathsf{true}/y\}$ which is behaviorally equivalent to the inactive process.

Next, the application of the value is followed by the synchronization on propagator c_1^V:

$$\begin{aligned} D^5 \longrightarrow\ & (\nu\, c_7)\,(\nu\, s_1)\,(\nu\, c_1^V, c_2^V)\, c_1^V!\langle\rangle \mid (\nu\, c_w)\, c_1^V?().\overline{s}_1?(w).(\overline{c_2^V}!\langle\rangle \mid W_w) \mid c_2^V?()\mathbf{0} \\ & \mid s_1!\langle \mathsf{true}\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0} \mid (\nu\, c_y)\,W_y\{\mathsf{true}/y\} \\ \longrightarrow\ & (\nu\, c_7)\,(\nu\, s_1)\,(\nu\, c_2^V)\,(\nu\, c_w)\,\overline{s}_1?(w).(\overline{c_2^V}!\langle\rangle \mid W_w) \mid c_2^V?()\mathbf{0} \\ & \mid s_1!\langle \mathsf{true}\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\mathbf{0} \mid (\nu\, c_y)\,W_y\{\mathsf{true}/y\} = D^6 \end{aligned}$$

Here, we can see that D^6 can simulate P', and its internal communication on the channel s.

4.6 Extension with Labeled Choice

In this section we discuss how to extend our approach to include sessions with *selection* and *branching* – constructs which are used commonly in session types to express deterministic choices. Forgoing formal proofs, we illustrate by examples how to harness the expressive power of abstraction-passing to decompose these constructs at the process level. First, we demonstrate how to break down selection and branching constructs in absence of recursion in Section 4.6.1. Then, in Section 4.6.2 we explore the interplay of recursion and labeled choice, as it requires special attention. Finally, in Section 4.6.3 we sketch how the operational correspondence proof can be adapted to account for branching and selection.

Let us briefly recall the labeled choice constructs in HO, following [39]. On the level of processes, selection and branching are modeled using labeled choice:

$$P, Q \ ::= \ \ldots \ \mid \ u \triangleleft l.P \ \mid \ u \triangleright \{l_i : P_i\}_{i \in I}$$

The process $u \triangleleft l.P$ selects the label l on channel u and then proceeds as P. The process $u \triangleright \{l_i : P_i\}_{i\in I}$ receives a label on the channel u and proceeds with the continuation branch P_i based on the received label. Selection and branching constructs can synchronize with each other, as represented in the operational semantics by the following reduction rule:

$$u \triangleleft l_j.Q \mid \overline{u} \triangleright \{l_i : P_i\}_{i\in I} \longrightarrow Q \mid P_j \qquad (j \in I) \quad [\text{Sel}]$$

At the level of types, selection and branching are represented with the following types:

$$S \ ::= \ \ldots \ \mid \ \oplus\{l_i : S_i\}_{i\in I} \ \mid \ \&\{l_i : S_i\}_{i\in I}$$

The *selection type* $\oplus\{l_i : S_i\}_{i\in I}$ and the *branching type* $\&\{l_i : S_i\}_{i\in I}$ are used to type, respectively, the selection and branching process constructs. Note the implicit sequencing

(Sel)
$$\frac{\Gamma;\Lambda;\Delta, u : S_j \vdash P \triangleright \diamond \quad j \in I}{\Gamma;\Lambda;\Delta, u : \oplus\{l_i : S_i\}_{i\in I} \vdash u \triangleleft l_j.P \triangleright \diamond}$$

(Bra)
$$\frac{\forall i \in I \quad \Gamma;\Lambda;\Delta, u : S_i \vdash P_i \triangleright \diamond}{\Gamma;\Lambda;\Delta, u : \&\{l_i : S_i\}_{i\in I} \vdash u \triangleright \{l_i : P_i\}_{i\in I} \triangleright \diamond}$$

Figure 4.8: Typing rules for selection and branching.

in the sessions involving selection and branching: the exchange of a label l_i precedes the execution of one of the stipulated protocol S_i. The typing rules for type-checking branching and selection processes are given in Figure 4.8.

Given these process constructs and types, what are the minimal versions of the session types with labeled choice? We do not consider branching and selection as atomic actions as their purpose is to make a choice of a stipulated protocol. In other words, it is not meaningful to type a channel with branching type in which all protocols are end. Thus, we extend the minimal syntax types Definition 4.3.1 with branching and selection constructs as follows:

$$M \ ::= \ \dots \ \mid \ \oplus\{l_i : M_i\}_{i\in I} \ \mid \ \&\{l_i : M_i\}_{i\in I}$$

That is, MSTs also include branching and selection types with MSTs nested in branches.

Next we explain our strategy for extending the breakdown function to account for selection and branching.

4.6.1 Breaking Down Selection and Branching

Notice that in a branching process $u \triangleright \{l_i : P_i\}_{i\in I}$ each subprocess P_i can have a different session with a different degree. Abstraction-passing allows to uniformly handle these kinds of processes. We extend the breakdown function in Definition 4.3.3 to selection and branching as follows:

$$\mathcal{G}(\&\{l_i : S_i\}_{i\in I}) = \&\{l_i : !\langle \mathcal{G}(S_i) {\multimap} \diamond\rangle\}_{i\in I}$$
$$\mathcal{G}(\oplus\{l_i : S_i\}_{i\in I}) = \oplus\{l_i : ?(\mathcal{G}(\overline{S_i}) {\multimap} \diamond)\}_{i\in I}$$

This decomposition follows the intuition that branching and selection correspond to the input and output of labels, respectively. For example, in the case of branching, once a particular branch l_i has been selected, we would like to input names on which to provide sessions from the branch $\mathcal{G}(S_i)$. In our higher-order setting, we do not input or output names directly. Instead, we send out an abstraction of the continuation process, which binds those names. It is then the job of the (complementary) selecting process to activate that abstraction with the names we want to select.

To make this more concrete, let us consider decomposition of branching and selection at the level of processes through the following extended example.

Example 4.6.1. Consider a mathematical server Q that offers clients two operations: addition and negation of integers. The server uses name u to implement the following session type:

$$S = \&\{\mathsf{add} : \underbrace{?(\mathsf{int});?(\mathsf{int});!\langle\mathsf{int}\rangle;\mathtt{end}}_{S_{\mathsf{add}}}\ ,\ \mathsf{neg} : \underbrace{?(\mathsf{int});!\langle\mathsf{int}\rangle;\mathtt{end}}_{S_{\mathsf{neg}}}\}$$

The branches have session types with different lengths: one receives two integers and sends over their sum, the other has a single input of an integer followed by an output of its negation.

Let us consider a possible implementation for the server Q and for a client R that selects the first branch to add integers 16 and 26:

$$\begin{aligned} Q &\triangleq u \triangleright \{\mathsf{add} : Q_{\mathsf{add}},\ \mathsf{neg} : Q_{\mathsf{neg}}\} & R &\triangleq \overline{u} \triangleleft \mathsf{add}.\overline{u}!\langle \mathbf{16}\rangle.\overline{u}!\langle \mathbf{26}\rangle.\overline{u}?(r) \\ Q_{\mathsf{add}} &\triangleq u?(a).u?(b).u!\langle a+b\rangle \\ Q_{\mathsf{neg}} &\triangleq u?(a).u!\langle -a\rangle \end{aligned}$$

The composed process $P \triangleq (\nu\, u)\,(Q \mid R)$ can reduce as follows:

$$\begin{aligned} P &\longrightarrow (\nu\, u)\,(u?(a).u?(b).u!\langle a+b\rangle \mid \overline{u}!\langle \mathbf{16}\rangle.\overline{u}!\langle \mathbf{26}\rangle.\overline{u}?(r)) \\ &\longrightarrow^2 (\nu\, u)\,(u!\langle \mathbf{16}+\mathbf{26}\rangle \mid \overline{u}?(r)) = P' \end{aligned}$$

Let us discuss the decomposition of P. First, the decomposition of S is the minimal session type M, defined as follows:

$$\begin{aligned} M = \mathcal{G}(S) = \&\{&\mathsf{add} : !\langle (?(\mathsf{int}), ?(\mathsf{int}), !\langle \mathsf{int}\rangle) {\multimap} \diamond\rangle, \\ &\mathsf{neg} : !\langle (?(\mathsf{int}), !\langle \mathsf{int}\rangle) {\multimap} \diamond\rangle\} \end{aligned}$$

Following Definition 4.3.9, we decompose P as follows:

$$\mathcal{D}(P) = (\nu\, c_1 \dots c_7)\,(\overline{c_1}!\langle\rangle \mid (\nu\, u_1)\,(c_1?().\overline{c_2}!\langle\rangle.\overline{c_3}!\langle\rangle \mid \mathcal{B}^2_\epsilon(Q\sigma_2) \mid \mathcal{B}^3_\epsilon(R\sigma_2)))$$

where $\sigma_2 = \{u_1\overline{u_1}/u\overline{u}\}$. The breakdown of the server process Q, which implements the branching, is as follows:

$$\begin{aligned} \mathcal{B}^2_\epsilon(Q\sigma_2) = c_2?().u_1 \triangleright \{&\mathsf{add} : u_1!\langle \underbrace{\lambda(y_1,y_2,y_3).\,(\nu\, c^V_1 \dots c^V_4)\,\overline{c^V_1}!\langle\rangle \mid \mathcal{B}^1_\epsilon(Q_{\mathsf{add}}\{y_1/u\})\sigma_V}_{V} \rangle, \\ &\mathsf{neg} : u_1!\langle \underbrace{\lambda(y_1,y_2).\,(\nu\, c^W_1 \dots c^W_3)\,\overline{c^W_1}!\langle\rangle \mid \mathcal{B}^1_\epsilon(Q_{\mathsf{neg}}\{y_1/u\})\sigma_W}_{W} \rangle\} \end{aligned}$$

where:

$$\begin{aligned} \mathcal{B}^1_\epsilon(Q_{\mathsf{add}}\{y_1/u\}) &= c_1?().y_1?(a).\overline{c_2}!\langle a\rangle \mid c_2?(a).y_2?(b).\overline{c_3}!\langle a,b\rangle \mid c_3?(a,b).y_3!\langle a+b\rangle.\overline{c_4}!\langle\rangle \mid c_4?() \\ \mathcal{B}^1_\epsilon(Q_{\mathsf{neg}}\{y_1/u\}) &= c_1?().y_1?(a).\overline{c_2}!\langle a\rangle \mid c_2?(a).y_2!\langle -a\rangle.\overline{c_3}!\langle\rangle \mid c_3?() \end{aligned}$$

with $\sigma_V = \{c^V_1, \dots, c^V_4/c_1, \dots, c_4\}$ and $\sigma_W = \{c^W_1, c^W_2, c^W_3/c_1, c_2, c_3\}$. In process $\mathcal{B}^2_\epsilon(Q\sigma_2)$, name u_1 implements the minimal session type M. Following the common trio structure, the first prefix awaits activation on c_2. The next prefix mimics the branching action of Q on u_1. Then, each branch consists of the output of an abstraction along u_1. This output does not have a counterpart in Q; it is meant to synchronize with process $\mathcal{B}^3_\epsilon(R\sigma_2)$, the breakdown of the corresponding selection process (see below).

The abstractions sent along u_1 encapsulate the breakdown of subprocesses in the two branches (Q_{add} and Q_{neg}). An abstraction in the branch has the same structure as the breakdown of a value $\lambda y : C^{\rightarrow}.\, P$ in Table 4.1: it is a composition of a control trio and the breakdown of a subprocess; the generated propagators are restricted. In the first branch the server needs three actions to perform the session, and in the second branch the server needs to perform two actions. Because of that the first abstraction binds three names y_1, y_2, y_3, and the second abstraction binds two names y_1, y_2.

In the bodies of the abstractions we break down Q_{add} and Q_{neg}, but not before adjusting the names on which the broken down processes provide the sessions. For this, we substitute u with y_1 in both processes, ensuring that the broken down names are bound by the abstractions.

By binding decomposed names in abstractions we account for different session types of the original name in branches, while preserving typability: this way the decomposition of different branches can use (i) the same names but typed with different minimal types and (ii) a different number of names, as it is the case in this example.

The decomposition of the client process R, which implements the selection, is as follows:

$$\mathcal{B}^3_\epsilon(R\sigma_2) = (\nu\, u_2, u_3, u_4)\, c_3?().\overline{u_1} \triangleleft \mathsf{add}.\overline{u_1}?(z).\overline{c_4}!\langle\rangle.z\,(u_2,u_3,u_4) \mid \mathcal{B}^4_\epsilon(\overline{u}_2!\langle\mathbf{16}\rangle.\overline{u}_2!\langle\mathbf{26}\rangle.\overline{u}_2?(r))$$

where:

$$\mathcal{B}^4_\epsilon(\overline{u}_2!\langle\mathbf{16}\rangle.\overline{u}_2!\langle\mathbf{26}\rangle.\overline{u}_2?(r)) = c_4?().\overline{u}_2!\langle\mathbf{16}\rangle.\overline{c_5}!\langle\rangle \mid c_5?().\overline{u}_3!\langle\mathbf{26}\rangle.\overline{c_6}!\langle\rangle \mid c_6?().\overline{u}_4?(r).\overline{c_7}!\langle\rangle \mid c_7?()$$

After receiving the context on c_3 (empty in this case), the selection action on u_1 is mimicked; then, an abstraction (an encapsulation of the selected branch) is received and applied to (u_2, u_3, u_4), which are locally bound. The intention is to use these names to connect the received abstraction and the continuation of a selection process: the subprocess encapsulated within the abstraction will use (u_2, u_3, u_4), while the dual names $(\overline{u}_2, \overline{u}_3, \overline{u}_4)$ are present in the breakdown of the continuation.

For simplicity, we defined $\mathcal{B}^3_\epsilon(R\sigma_2)$ using a subprocess with four prefixes. Alternatively, we could have used a decomposition that relies on two trios, by introducing abstraction passing as in Section 4.5.

We will now examine the reductions of the decomposed process $\mathcal{D}(P)$. First, c_1, c_2, and c_3 will synchronize. We have $\mathcal{D}(P) \longrightarrow^4 D_1$, where

$$\begin{aligned} D_1 = (\nu\, c_4 \dots c_7)\,(\nu\, u_1)\,(u_1 \triangleright \{\mathsf{add}:\ u_1!\langle V\rangle,\ \mathsf{neg}:\ u_1!\langle W\rangle\} \\ \mid (\nu\, u_2,u_3,u_4)\,(\lambda(y_1,y_2,y_3).\,\overline{u}_1 \triangleleft \mathsf{add}.\overline{u}_1?(z).\overline{c_4}!\langle\rangle.z\,(y_1,y_2,y_3))\,(u_2,u_3,u_4) \mid \\ \mathcal{B}^4_\epsilon(\overline{u}!\langle\mathbf{26}\rangle.\overline{u}?(r))) \end{aligned}$$

In D_1, (u_2, u_3, u_4) will be applied to the abstraction; after that, the process chooses the label add on u_1. Process D_1 will reduce further as $D_1 \longrightarrow^2 D_2 \longrightarrow^2 D_3$, where:

$$\begin{aligned} D_2 &= (\nu\, c_4 \dots c_7)\,(\nu\, u_1)\,(\ u_1!\langle V\rangle \mid (\nu\, u_2,u_3,u_4)\,(\overline{u}_1?(z).\overline{c_4}!\langle\rangle.z\,(u_2,u_3,u_4) \mid \mathcal{B}^4_\epsilon(\overline{u}!\langle\mathbf{26}\rangle.\overline{u}?(r)))) \\ D_3 &= (\nu\, c_4 \dots c_7)\,(\nu\, u_1,u_2,u_3,u_4)\,(\overline{c_4}!\langle\rangle.V\,(u_2,u_3,u_4) \mid \\ &\qquad c_4?().\overline{u}_2!\langle\mathbf{16}\rangle.\overline{c_5}!\langle\rangle \mid c_5?().\overline{u}_3!\langle\mathbf{26}\rangle.\overline{c_6}!\langle\rangle \mid c_6?().\overline{u}_4?(r).\overline{c_7}!\langle\rangle \mid c_7?()) \end{aligned}$$

Then D_3 reduces as $D_3 \longrightarrow D_4 \longrightarrow D_5$, where:

$$\begin{aligned} D_4 &= (\nu\, c_5 \dots c_7)\,(\nu\, u_2,u_3,u_4)\,((\nu\, c^V_1 \dots c^V_4)\,(\overline{c^V_1}!\langle\rangle \mid c^V_1?().u_2?(a).\overline{c^V_2}!\langle a\rangle \mid c^V_2?(a).u_3?(b).\overline{c^V_3}!\langle a,b\rangle \mid \\ &\qquad c^V_3?(a,b).u_4!\langle a+b\rangle.\overline{c^V_4}!\langle\rangle \mid c^V_4?()) \mid \\ &\qquad \overline{u}_2!\langle\mathbf{16}\rangle.\overline{c_5}!\langle\rangle \mid c_5?().\overline{u}_3!\langle\mathbf{26}\rangle.\overline{c_6}!\langle\rangle \mid c_6?().\overline{u}_4?(r).\overline{c_7}!\langle\rangle \mid c_7?()) \\ D_5 &= (\nu\, c_5 \dots c_7)\,(\nu\, u_2,u_3,u_4)\,((\nu\, c^V_2 \dots c^V_4)\,(u_2?(a).\overline{c^V_2}!\langle a\rangle \mid c^V_2?(a).u_3?(b).\overline{c^V_3}!\langle a,b\rangle \mid \\ &\qquad c^V_3?(a,b).u_4!\langle a+b\rangle.\overline{c^V_4}!\langle\rangle \mid c^V_4?()) \mid \\ &\qquad \overline{u}_2!\langle\mathbf{16}\rangle.\overline{c_5}!\langle\rangle \mid c_5?().\overline{u}_3!\langle\mathbf{26}\rangle.\overline{c_6}!\langle\rangle \mid c_6?().\overline{u}_4?(r).\overline{c_7}!\langle\rangle \mid c_7?()) \end{aligned}$$

Now, process D_5 can mimic the original transmission of the integer 16 on channel u_2 as follows:

$$\begin{aligned} D_5 \longrightarrow (\nu\, c_5 \dots c_7)\,(\nu\, u_2,u_3,u_4)\,((\nu\, c^V_2 \dots c^V_4)\,(\overline{c^V_2}!\langle\mathbf{16}\rangle \mid c^V_2?(a).u_3?(b).\overline{c^V_3}!\langle a,b\rangle \mid \\ c^V_3?(a,b).u_4!\langle a+b\rangle.\overline{c^V_4}!\langle\rangle \mid c^V_4?()) \mid \\ \overline{c_5}!\langle\rangle \mid c_5?().\overline{u}_3!\langle\mathbf{26}\rangle.\overline{c_6}!\langle\rangle \mid c_6?().\overline{u}_4?(r).\overline{c_7}!\langle\rangle \mid c_7?()) = D_6 \end{aligned}$$

Finally, process D_6 reduces to D_7 in three steps, as follows:

$$D_6 \longrightarrow^3 (\nu\, c_5 \dots c_7)\,(\nu\, u_4)\,((\nu\, c^V_4)\,(u_4!\langle\mathbf{16}+\mathbf{26}\rangle.\overline{c^V_4}!\langle\rangle \mid c^V_4?()) \mid \overline{u}_4?(r).\overline{c_7}!\langle\rangle \mid c_7?()) = D_7$$

Clearly, process D_7 correctly simulates the synchronizations of the process P'. ◁

4.6.2 The Interplay of Selection/Branching and Recursion

Now, we discuss by example how recursive session types involving branching/selection are broken down. For simplicity, we consider recursive types without nested recursion and in which the recursive step is followed immediately by branching or selection, without any intermediate actions, i.e. types of the following form:

$$\mu \mathsf{t}.\&\{l_i : S_i\}_{i\in I} \qquad \mu \mathsf{t}.\oplus\{l_i : S_i\}_{i\in I}$$

where none of S_i contain branching/selection or recursion.

In this case, the decomposition of branching recursive types should be defined differently than for tail-recursive types: a type such as $\mu \mathsf{t}.\&\{l_i : S_i\}_{i\in I}$ does not necessarily describe a channel with an infinite behavior, because some of the branches S_i can result in termination. In such case, decomposing all actions in the type $\&\{l_i : S_i\}_{i\in I}$ as their own recursive types using the $\mathcal{R}(-)$ function would be incorrect.

Instead, we decompose the body of the recursive type with $\mathcal{G}(-)$ itself:

$$\mathcal{G}(\mu \mathsf{t}.\&\{l_i : S_i\}_{i\in I}) = \mu \mathsf{t}.\&\{l_i : !\langle \mathcal{G}(S_i) \multimap \diamond\rangle\}_{i\in I}$$
$$\mathcal{G}(\mu \mathsf{t}.\oplus\{l_i : S_i\}_{i\in I}) = \mu \mathsf{t}.\oplus\{l_i : ?(\mathcal{G}(\overline{S_i}) \multimap \diamond)\}_{i\in I}$$

If some branch S_i contains the recursion variable t, then it will appear in $\mathcal{G}(S_i)$, because $\mathcal{G}(\mathsf{t}) = \mathsf{t}$. That is, recursion variables will appear as part of the abstraction $\mathcal{G}(\overline{S_i}) \multimap \diamond$. That means that the decomposition of a tail-recursive type form can produce a minimal *non*-tail-recursive types.

Now, we illustrate this decomposition on the level of processes.

Example 4.6.2. We consider a process P with a name r that is typed as follows:

$$S = \mu \mathsf{t}.\&\{l_1 : ?(\mathsf{str});!\langle \mathsf{int}\rangle;\mathsf{t},\ l_2 : \mathsf{end}\}.$$

For simplicity, we give P in HOπ (which includes HO with recursion as sub-calculus):

$$\begin{aligned}
P &= R \mid Q\\
R &= \mu X.r \triangleright \{l_1 : r?(t).r!\langle \mathtt{len}(t)\rangle.X,\ l_2 : \mathbf{0}\}\\
Q &= \overline{r} \triangleleft l_1.\overline{r}!\langle \text{“Hello”}\rangle.\overline{r}?(a_1).\overline{r} \triangleleft l_1.\overline{r}!\langle \text{“World”}\rangle.\overline{r}?(a_2).\overline{r} \triangleleft l_2.\mathbf{0}
\end{aligned}$$

That is, P contains a server R which either accepts a new request to calculate a length of a string, or to terminate. Dually, P contains a client Q, which uses the server twice before terminating.

We can give an equivalent process in HO by encoding the recursion (as done in [39]):

$$\begin{aligned}
[\![P]\!] &= [\![R]\!] \mid Q\\
[\![R]\!] &= (\nu\, s)\,(V\, r, s \mid \overline{s}!\langle V\rangle)\\
V &= \lambda(x_r, x_s).\, x_s?(y).x_r \triangleright \{l_1 : x_r?(t).x_r!\langle \mathtt{len}(t)\rangle.(\nu\, s)\,(y\,(x_r, s) \mid \overline{s}!\langle y\rangle),\ l_2 : \mathbf{0}\}
\end{aligned}$$

The decomposition of S, denoted M^*, is the following minimal session type:

$$M^* = \mathcal{G}(S) = \mu \mathsf{t}.\&\{l_1 : !\langle(?(\mathsf{str}),\ !\langle \mathsf{int}\rangle,\ \mathsf{t}) \multimap \diamond\rangle,\ l_2 : \mathsf{end}\}$$

As in the previous example (Example 4.6.1), the continuation of a selected branch will be packed in an abstraction and sent over. This abstraction binds names on which the session actions should be performed. In addition, if a branch contains a recursive call, then the last argument of the abstraction will be a name on which the next instance of the recursion will be mimicked. We illustrate this mechanism by giving the decomposition of $[\![P]\!]$ and inspecting its reductions.

$$\begin{aligned}
\mathcal{D}([\![P]\!]) &= (\nu\, c_1,\ldots,c_{12})\, \overline{c_1}!\langle\rangle \mid c_1?().\overline{c_2}!\langle\rangle.\overline{c_5}!\langle\rangle \mid \mathcal{B}^2_\epsilon([\![R]\!]) \mid \mathcal{B}^5_\epsilon(Q)\\
\mathcal{B}^2_\epsilon([\![R]\!]) &= (\nu\, s_1)\,(c_2?().\overline{c_3}!\langle\rangle.\overline{c_4}!\langle\rangle \mid c_3?().\mathcal{V}_\epsilon(V)\,(r_1,s_1) \mid c_4?().\overline{s}_1!\langle \mathcal{V}_\epsilon(V)\rangle)\\
\mathcal{V}_\epsilon(V) &= \lambda(x_{r_1},x_{s_1}).\,(\nu\, c^V_1\, c^V_2)\, \overline{c^V_1}!\langle\rangle \mid c^V_1?().x_{s_1}?(y).\overline{c^V_2}!\langle y\rangle \mid c^V_2?(y).x_{r_1} \triangleright \{l_1 : x_{r_1}!\langle W\rangle,\ l_2 : \mathbf{0}\}\\
W &= \lambda(z_1,z_2,z_3).\,(\nu\, c^W_1 \ldots c^W_5)\, \overline{c^W_1}!\langle\rangle \mid c^W_1?().z_1?(t).\overline{c^W_2}!\langle t\rangle \mid c^W_2?(t).z_2!\langle \mathtt{len}(t)\rangle.\overline{c^W_3}!\langle\rangle\\
&\quad \mid (\nu\, s_1)\,(c^W_3?().\overline{c^W_4}!\langle\rangle.\overline{c^W_5}!\langle\rangle \mid c^W_4?().y\,(z_3,s_1) \mid c^W_5?().\overline{s}_1!\langle y\rangle)\\
\mathcal{B}^5(Q) &= (\nu\, r_2 : ?(\mathsf{str}),\ r_3 : !\langle \mathsf{int}\rangle,\ r_4 : M^*)\\
&\qquad c_5?().\overline{r}_1 \triangleleft l_1.\overline{c_6}!\langle\rangle.\overline{r}_1?(y).y\,(r_2,r_3,r_4) \mid\\
&\qquad c_6?().\overline{r}_2!\langle\text{“Hello”}\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\overline{r}_3?(t).\overline{c_{11}}!\langle\rangle \mid\\
&\qquad\quad (\nu\, r_5 : ?(\mathsf{str}),\ r_6 : !\langle \mathsf{int}\rangle,\ r_7 : M^*)\\
&\qquad\qquad c_8?().\overline{r}_4 \triangleleft l_1.\overline{c_{12}}!\langle\rangle.\overline{r}_4?(y).y\,(r_5,r_6,r_7) \mid c_9?().\overline{r}_5!\langle\text{“World”}\rangle.\overline{c_{10}}!\langle\rangle \mid\\
&\qquad\qquad c_{10}?().\overline{r}_6?(t).\overline{c_{11}}!\langle\rangle \mid c_{11}?().\overline{r}_7 \triangleleft l_2.\overline{c_{12}}!\langle\rangle \mid c_{12}?()
\end{aligned}$$

In the process $\mathcal{B}^5(Q)$, the restricted names (r_2,r_3,r_4) are the decomposition of the name r for the branch l_1. To calculate their types, we unfold S:

$$\begin{aligned}
S &= \mu \mathsf{t}.\&\{l_1 : ?(\mathsf{str});!\langle \mathsf{int}\rangle;\mathsf{t},\ l_2 : \mathsf{end}\} \equiv \&\{l_1 : ?(\mathsf{str});!\langle \mathsf{int}\rangle;\mathsf{t},\ l_2 : \mathsf{end}\}\{^S/_\mathsf{t}\}\\
&= \&\{l_1 : ?(\mathsf{str});!\langle \mathsf{int}\rangle;S,\ l_2 : \mathsf{end}\},
\end{aligned}$$

and we look at the decomposition of the type corresponding to the branch l_1:

$$\mathcal{G}(?(\mathsf{str});!\langle \mathsf{int}\rangle;S) = (?(\mathsf{str}),\ !\langle \mathsf{int}\rangle,\ M^*)$$

Now we inspect a few reductions of $\mathcal{D}(P)$. First, we have synchronizations on $c_1,\ldots,c_4$. This is followed by the application of the exchanged value $\mathcal{V}_\epsilon(V)$ to names r_1, s_1:

$$\begin{aligned}
\mathcal{D}([\![P]\!]) \longrightarrow^* &(\nu\, c^V_1\, c^V_2)\, \overline{c^V_1}!\langle\rangle \mid c^V_1?().s_1?(y).\overline{c^V_2}!\langle y\rangle \mid c^V_2?(y).r_1 \triangleright \{l_1 : x_{r_1}!\langle W\rangle,\ l_2 : \mathbf{0}\}\\
&\mid \overline{s}_1!\langle \mathcal{V}_\epsilon(V)\rangle \mid \overline{c_5}!\langle\rangle \mid \mathcal{B}^5_\epsilon(Q) = D_1
\end{aligned}$$

Then, after synchronizations on c^V_1, s_1, and c^V_2 in D_1 we have the following:

$$\begin{aligned}
D_1 \longrightarrow^* &(\nu\, c_6,\ldots,c_{12})\, r_1 \triangleright \{l_1 : r_1!\langle W\{^{\mathcal{V}_\epsilon(V)}/_y\}\rangle,\ l_2 : \mathbf{0}\} \mid\\
&\quad (\nu\, r_2 : ?(\mathsf{str}),\ r_3 : !\langle \mathsf{int}\rangle,\ r_4 : M^*)\\
&\qquad \overline{r}_1 \triangleleft l_1.\overline{c_6}!\langle\rangle.\overline{r}_1?(y).y\,(r_2,r_3,r_4) \mid\\
&\qquad c_6?().\overline{r}_2!\langle\text{“Hello”}\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\overline{r}_3?(t).\overline{c_{11}}!\langle\rangle \mid\\
&\qquad\quad (\nu\, r_5 : ?(\mathsf{str}),\ r_6 : !\langle \mathsf{int}\rangle,\ r_7 : M^*)\\
&\qquad\qquad c_8?().\overline{r}_4 \triangleleft l_1.\overline{c_{12}}!\langle\rangle.\overline{r}_4?(y).y\,(r_5,r_6,r_7) \mid c_9?().\overline{r}_5!\langle\text{“World”}\rangle.\overline{c_{10}}!\langle\rangle \mid\\
&\qquad\qquad c_{10}?().\overline{r}_6?(t).\overline{c_{11}}!\langle\rangle \mid c_{11}?().\overline{r}_7 \triangleleft l_2.\overline{c_{12}}!\langle\rangle \mid c_{12}?() = D_2
\end{aligned}$$

D_2 can mimic a silent select action on r_1; this is followed by a reception of value $W\{^{\mathcal{V}_\epsilon(V)}/_y\}$

on name $\overline{r}_1$, which is then applied to names (r_2, r_3, r_4). The resulting process is as follows:

$$\begin{aligned} D_2 \longrightarrow^* &(\nu\, c_8, \ldots, c_{12})\, (\nu\, r_2 :?(\mathsf{str}),\ r_3 :!\langle\mathsf{int}\rangle,\ r_4 : M^*) \\ &\quad (\nu\, c_1^W \ldots c_5^W)\, \overline{c_1^W}!\langle\rangle \mid c_1^W?().r_2?(t).\overline{c_2^W}!\langle t\rangle \mid c_2^W?(t).r_3!\langle \mathtt{len}(t)\rangle.\overline{c_3^W}!\langle\rangle \\ &\quad \mid (\nu\, s_1)\, (c_3^W?().\overline{c_4^W}!\langle\rangle.\overline{c_5^W}!\langle\rangle \mid c_4^W?().\mathcal{V}_\epsilon(V)\, (r_4, \overline{s}_1) \mid c_5^W?().s_1!\langle \mathcal{V}_\epsilon(V)\rangle) \mid \\ &\quad \overline{r}_2!\langle\text{“Hello”}\rangle.\overline{c_7}!\langle\rangle \mid c_7?().\overline{r}_3?(t).\overline{c_{11}}!\langle\rangle \mid \\ &\qquad (\nu\, r_5 :?(\mathsf{str}),\ r_6 :!\langle\mathsf{int}\rangle,\ r_7 : M^*) \\ &\qquad\quad c_8?().\overline{r}_4 \triangleleft l_1.\overline{c_{12}}!\langle\rangle.\overline{r}_4?(y).y\, (r_5, r_6, r_7) \mid c_9?().\overline{r}_5!\langle\text{“World”}\rangle.\overline{c_{10}}!\langle\rangle \mid \\ &\qquad\quad c_{10}?().\overline{r}_6?(t).\overline{c_{11}}!\langle\rangle \mid c_{11}?().\overline{r}_7 \triangleleft l_2.\overline{c_{12}}!\langle\rangle \mid c_{12}?() = D_3 \end{aligned}$$

The next interesting process emerges once silent actions on r are mimicked by r_2 and r_3:

$$\begin{aligned} D_3 \longrightarrow^* &(\nu\, c_8, \ldots, c_{12})\, (\nu\, r_4 : M^*)\, (\nu\, s_1)\, \mathcal{V}_\epsilon(V)\, (r_4, s_1) \mid \overline{s}_1!\langle \mathcal{V}_\epsilon(V)\rangle) \mid \\ &\quad (\nu\, r_5 :?(\mathsf{str}),\ r_6 :!\langle\mathsf{int}\rangle,\ r_7 : M^*) \\ &\qquad c_8?().\overline{r}_4 \triangleleft l_1.\overline{c_{12}}!\langle\rangle.\overline{r}_4?(y).y\, (r_5, r_6, r_7) \mid c_9?().\overline{r}_5!\langle\text{“World”}\rangle.\overline{c_{10}}!\langle\rangle \mid \\ &\qquad c_{10}?().\overline{r}_6?(t).\overline{c_{11}}!\langle\rangle \mid c_{11}?().\overline{r}_7 \triangleleft l_2.\overline{c_{12}}!\langle\rangle \mid c_{12}?() = D_4 \end{aligned}$$

In D_4, name r_4 with type M^*, is applied to the abstraction $\mathcal{V}_\epsilon(V)$, which encapsulates a “new instance” of the recursive branch process. After application, we obtain the following process:

$$\begin{aligned} D_4 \longrightarrow^* &(\nu\, c_8, \ldots, c_{12})\, (\nu\, r_4 : M^*)\, (\nu\, s_1)\, (\nu\, c_1^V c_2^V)\, \overline{c_1^V}!\langle\rangle \mid c_1^V?().s_1?(y).\overline{c_2^V}!\langle y\rangle \mid \\ &\qquad\quad c_2^V?(y).r_4 \triangleright \{l_1 : r_4!\langle V_1\rangle,\ l_2 : \mathbf{0}\} \mid s_1!\langle \mathcal{V}_\epsilon(V)\rangle) \mid \\ &\quad (\nu\, r_5 :?(\mathsf{str}),\ r_6 :!\langle\mathsf{int}\rangle,\ r_7 : M^*) \\ &\qquad c_8?().\overline{r}_4 \triangleleft l_1.\overline{c_{12}}!\langle\rangle.\overline{r}_4?(y).y\, (r_5, r_6, r_7) \mid c_9?().\overline{r}_5!\langle\text{“World”}\rangle.\overline{c_{10}}!\langle\rangle \mid \\ &\qquad c_{10}?().\overline{r}_6?(t).\overline{c_{11}}!\langle\rangle \mid c_{11}?().\overline{r}_7 \triangleleft l_2.\overline{c_{12}}!\langle\rangle \mid c_{12}?() = D_5 \end{aligned}$$

Thus, we can see that after few administrative reductions (on c_1^V, s_1, and c_2^V) the process is able to mimic the a next selection on r on name r_4. As the process again selects l_1, we can see that the next selection will occur on name r_7, again typed with M^*. $\triangleleft$

We would like to finish this subsection with the following remark. So far we have only considered recursive types which did not contain any actions between recursion and branching/selection. However, types with prefixed branching

$$\mu \mathsf{t}.\alpha_1.\ldots.\alpha_n.\&\{l_i : S_i\}_{i \in I},$$

where $\alpha_1, \ldots, \alpha_n$ are some session prefixes, can also be accommodated in the same framework, as these types can be written equivalently without prefixed branching:

$$\alpha_1.\ldots.\alpha_n.\mu \mathsf{t}.\&\{l_i : S_i\{{}^{\alpha_1.\ldots.\alpha_n.\mathsf{t}}/_{\mathsf{t}}\}\}_{i \in I}.$$

4.6.3 Adapting Operational Correspondence

We briefly remark on how to adapt the operational correspondence result from Section 4.4. For the operational correspondence result, and the related lemmas, we must enforce additional constraints on the processes that we break down. These concerns arise from the following fact. When a type $\&\{l_i : S_i\}_{i \in I}$ is broken down as

$$\mathcal{G}(\&\{l_i : S_i\}_{i \in I}) = \&\{l_i :!\langle \mathcal{G}(S_i) {\multimap} \diamond\rangle\}_{i \in I},$$

an additional action gets introduced on the level of MST processes. After performing the branching, an abstraction needs to be sent out. This additional action will be matched by a corresponding abstraction-input action on the side of selection, if present. However, this abstraction-sending action does not correspond to any action of the source process.

Therefore, to show the operational correspondence between the source term and its decomposition, we need to restrict our attention to processes in which branching and selection types are both present in (matching) pairs. Specifically, we assume the following conditions on the source process P:

- P is a well-typed, that is $\Gamma; \Delta; \Lambda \vdash P \triangleright \diamond$ with $\mathsf{balanced}(\Delta)$;
- for any name u, $u \in \mathtt{fn}(P)$ with $u : S$ such that S involves selection or branching constructs if and only if $\overline{u} \in \mathtt{fn}(P)$.

Intuitively, these two conditions ensure that every branching action in P has its complement (and vice-versa). Note that for closed typeable processes both the balancedness condition and the second condition on names are vacuously true.

With this condition in place, we need to enlarge the relation $\mathcal{S}$ in order to account for silent actions that are introduced by the breakdown of selection and branching constructs. That is, when matching the original silent action involving selection/branching, the corresponding broken down process need to perform several silent actions, in order to be able to mimic the process continuation.

4.7 Related Work

We draw inspiration from insights developed by Parrow [50], who showed that every process in the untyped, summation-free π-calculus with replication is weakly bisimilar to its decomposition into trios (i.e., $P \approx \mathcal{D}(P)$). As already mentioned, we are concerned with a different technical setting: our decomposition treats processes from a calculus without name-passing but with higher-order concurrency (abstraction-passing), supports recursive types, and can accommodate labeled choices. Our goals are different than those of Parrow [50]: for us, trios processes are a relevant instrument for defining and justifying minimal session types, but they are not an end in themselves. Still, we retain the definitional style and terminology for trios from [50], which are elegant and clear.

Our main results connect the typability and the behaviour of a process with its decomposition, as witnessed by the static and dynamic correctness theorems. Static correctness was not considered by Parrow, as he worked in an untyped setting. As for dynamic correctness, a similar result was established in [50], linking the process and its decomposition through weak bisimilarity. In our setting we had to use a different, typed notion of bisimilarity. An obstacle here is that known notions of typed bisimilarity for session-typed processes, such as those given by Kouzapas et al. [40], only relate processes typed under the *same* typing environments. To that extent, our notion of equivalence (MST bisimulations) is more flexible than prior related notions as it (i) relates processes typable under different environments (e.g., Δ and $\mathcal{G}(\Delta)$) and (ii) admits that actions along a name s from P can be matched by $\mathcal{D}(P)$ using actions along indexed names s_k, for some k (and viceversa).

As mentioned in the introduction, our approach is broadly related to works that relate session types with other type systems for the π-calculus (cf. [37, 15, 16, 20, 27]). Hence, these works target the *relative expressiveness* of session-typed process languages, by encoding processes between two different systems. By contrast, we relate a session types system with its subsystem of minimal types. Thus, by explaining session types in terms of themselves, our work emerges as the first study of *absolute expressiveness* in the context of session types.

In this context, works by Kobayashi [37] and Dardha et al. [15, 16] are worth discussing. Kobayashi [37] encoded a finite session π-calculus into a π-calculus with linear types with usages (without sequencing); this encoding uses a continuation-passing style to codify a session name using multiple linear channels. Dardha et al. [15, 16] formalize and extend Kobayashi's approach. They use two separate encodings, one for processes and one for types. The encoding of processes uses a freshly generated linear name to mimic each session action; this fresh name becomes an additional argument in communications. The encoding of types codifies sequencing in session types by nesting payload types. In contrast, we "slice" the n actions occurring in a session $s : S$ along indexed names $s_1, \ldots, s_n$ with minimal session types—n slices of S. Hence, Dardha et al.'s could be described as codifying sequencing in a "dynamic style", via the freshly generated names, whereas we follow a "static style" using names that are indexed according to the corresponding session type.

Recently, Jacobs [33] developed a small programming calculus with a single fork-like construct and a linear type system, which can be used to encode session-typed communications. His system can be seen as a distillation of Wadler's GV [59] which is, in essence, a λ-calculus with session-based concurrency; in contrast, HO can be seen as a π-calculus in which abstractions can be exchanged. While similar in spirit, our work and the developments by Jacobs are technically distant; we observe that the operational correspondences developed in [33] are strictly simpler than our dynamic correspondence result (Theorem 4.4.1) although they are mechanized in the Coq proof assistant.

Finally, we elaborate further on our choice of HO as source language for minimal session types. HO is one of the sub-calculi of HOπ, a higher-order process calculus with recursion and both name- and abstraction-passing. The basic theory of HOπ was studied by Kouzapas et al. [39, 40] as a hierarchy of session-typed calculi based on relative expressiveness. Our results enable us to place HO with minimal session types firmly within this hierarchy. Still, the definition of minimal session types does not rely on having HO as source language, as they can be defined on top of other process languages. In fact, in separate work we have defined minimal session types on top of the first-order sub-calculus of HOπ [4]. This development attests that minimal session types admit meaningful formulations independently from the kind of communicated objects (abstractions or names).

4.8 Concluding Remarks

We have presented a minimal formulation of session types, one of the most studied classes of behavioral types for message-passing programs. This minimal formulation forgoes sequencing on the level of types. We formally connect standard and minimal session types (MSTs), through a *decomposition* of session-typed processes, adopting the higher-order process calculus HO as target language. Following Parrow [50], we defined the decomposition of a process P, denoted $\mathcal{D}(P)$, as a collection of *trio processes* (processes with at most three actions) that trigger each other mimicking the sequencing in the original process. We proved that typability of P using standard session types implies the typability of $\mathcal{D}(P)$ with minimal session types; we also established that P and $\mathcal{D}(P)$ are behaviourally equivalent through an *MST bisimulation*. Our results hold for all session types constructs, including labeled choices and recursive types.

From a foundational standpoint, our study of minimal session types is a conceptual contribution to the theory of behavioral types, in that we clarify the status of sequencing in theories of session types. As remarked in Section 4.1, there are many session types variants, and their expressivity often comes at the price of an involved underlying theory. Our work contributes in the opposite direction, as we identified a simple yet expressive fragment of an established session-typed framework [39, 40], which allows us to justify session types in terms of themselves. Understanding further the underlying theory of minimal session types (e.g., notions such as type-based compatibility) is an exciting direction for future work.

As mentioned above, one insight derived from our results is that sequentiality in session types is convenient but not indispensable. Convenience is an important factor in the design of type systems for message-passing programs, because types are abstract specifications of communication structures. By identifying sequencing as a source of redundancy, our minimal formulation of session types does not contradict or invalidate the prior work on standard session types and their extensions; rather, it contributes to our understanding of the sources of convenience of those advanced type systems.

In formulating minimal session types we have committed to a specific notion of minimality, tied to sequencing constructs in types—arguably the most distinctive feature in session types. There could be other notions of minimality, unrelated to sequencing but worth exploring nevertheless. Consider, for instance, the framework of *context-free* session types [58], which extend standard session types by allowing sequencing of the form $S; T$. This form of sequential composition is quite powerful, and yet it could be seen as achieving a form of minimality different from the one we studied here: as illustrated in [58, Section 5], context-free session types allow to describe the communication of tree-structured data while minimizing the need for channel creation and avoiding channel passing.

Our work can be seen as a new twist on Parrow's decomposition results in the *untyped* setting [50]. While Parrow's work indeed does not consider types, in fairness we must observe that when Parrow's work appeared (1996) the study of types (and typed behavioral equivalences) for the π-calculus was rather incipient (for instance, the widely known formulation of binary session types, given in [30], appeared in 1998). That said, we would like to stress that our results are not merely an extension of Parrow's work with session types, for types in our setting drastically narrow down the range of conceivable decompositions. Additionally, in this work we exploit features not supported in [50], most notably higher-order concurrency (cf. Section 4.5).

Finally, from a practical standpoint, we believe that our approach paves a new avenue to the integration of session types in programming languages whose type systems lack sequencing, such as Go. It is natural to envision program analysis tools which, given a message-passing program that should conform to protocols specified as session types, exploit our decomposition as an intermediate step in the verification of communication correctness. Remarkably, our decomposition lends itself naturally to an implementation—in fact, we generated our examples automatically using MISTY, an associated artifact written in Haskell [6].

5 Minimal Session Types for the π-calculus

5.1 Introduction

In this chapter, we investigate a *minimal formulation* of session types for the π-calculus, the paradigmatic calculus of concurrency.

The minimality result in Chapter 4 holds for a *higher-order* process calculus called HO, in which values are only abstractions (functions from names to processes). HO does not include name passing nor process recursion, but it can encode them precisely [39, 41]. An important question left open is whether the minimality result holds for the session π-calculus (dubbed π), the language in which values are names and for which session types have been more widely studied from foundational and practical perspectives.

In this chapter, we positively answer this question. Our approach is simple, perhaps even deceptively so. In order to establish the minimality result for π, we compose the decomposition in [7] with the mutual encodings between HO and π given in [39, 41].

$$\begin{tikzcd}
\pi \arrow[r, "\mathcal{F}(\,\cdot\,)"'] \arrow[d, dotted, "\llbracket \cdot \rrbracket^1_g"] & \mu\pi \\
\mathsf{HO} \arrow[r, dotted, "\mathcal{D}(\cdot)"] & \mu\mathsf{HO} \arrow[u, dotted, "\llbracket \cdot \rrbracket^2"']
\end{tikzcd}$$

Figure 5.1: Decomposition by composition.

Let $\mu\pi$ and μHO denote the process languages π and HO with MSTs (rather than with standard session types). Also, let $\mathcal{D}(\cdot)$ denote the decomposition function from HO to μHO given in Definition 4.3.9. Kouzapas et al. [39, 41] gave typed encodings from π to HO (denoted $\llbracket \cdot \rrbracket^1_g$) and from HO to π (denoted $\llbracket \cdot \rrbracket^2$). Therefore, given $\mathcal{D}(\cdot)$, $\llbracket \cdot \rrbracket^1_g$, and $\llbracket \cdot \rrbracket^2$, to define a decomposition function $\mathcal{F}(\,\cdot\,) : \pi \to \mu\pi$, it suffices to follow Figure 5.1. This is sound for our purposes, because $\llbracket \cdot \rrbracket^1_g$ and $\llbracket \cdot \rrbracket^2$ preserve sequentiality in processes and their types.

The *first contribution* of this chapter is the decomposition function $\mathcal{F}(\,\cdot\,)$, whose correctness follows from that of its constituent functions. $\mathcal{F}(\,\cdot\,)$ is significant, as it provides an elegant, positive answer to the question of whether the minimality result in [7] holds for π. Indeed, it proves that the values exchanged do not influence sequentiality in session types: the minimality result of [7] is not specific to the abstraction-passing language HO.

However, $\mathcal{F}(\,\cdot\,)$ is not entirely satisfactory. A side effect of composing $\mathcal{D}(\cdot)$, $\llbracket \cdot \rrbracket^1_g$, and $\llbracket \cdot \rrbracket^2$ is that the resulting decomposition of π into $\mu\pi$ is inefficient, as it includes redundant synchronizations. These shortcomings are particularly noticeable in the treatment of recursive processes. The *second contribution* of this chapter is an optimized variant of $\mathcal{F}(\,\cdot\,)$, dubbed $\mathcal{F}^*(\,\cdot\,)$, in which we remove redundant synchronizations and target recursive processes and variables directly, exploiting the fact that π supports recursion natively.

Contributions The main contributions of this chapter are:

1. A minimality result for π, based on the function $\mathcal{F}(\,\cdot\,)$.
2. $\mathcal{F}^*(\,\cdot\,)$, an optimized variant of $\mathcal{F}(\,\cdot\,)$, without redundant communications and with native support for recursion.
3. Examples for $\mathcal{F}(\,\cdot\,)$ and $\mathcal{F}^*(\,\cdot\,)$.

Omitted material (definitions, correctness proofs for $\mathcal{F}(\,\cdot\,)$ and $\mathcal{F}^*(\,\cdot\,)$, additional examples) can be found in the appendices. Throughout the chapter, we use red and blue colors to distinguish elements of the first and second decompositions, respectively.

5.2 The Decomposition

Our goal is to define $\mu\pi$, i.e., the language π equipped with MSTs. In this section, we define a decomposition function $\mathcal{F}(\,\cdot\,) : \pi \to \mu\pi$, given in terms of a breakdown function denoted $\mathcal{A}^k_{\tilde{x}}(\,\cdot\,)_g$ (cf. Table 5.1). Following Figure 5.1, this breakdown function will result from the composition of $[\![\,\cdot\,]\!]^1_g$, $\mathcal{B}^k_{\tilde{x}}(\cdot)$, and $[\![\,\cdot\,]\!]^2$, i.e., $\mathcal{A}^k_{\tilde{x}}(\,\cdot\,)_g = [\![\mathcal{B}^k_{\tilde{x}}([\![\,\cdot\,]\!]^1_g)]\!]^2$. Using $\mathcal{F}(\,\cdot\,)$, we shall obtain a minimality result for π (Theorem 5.2.1).

5.2.1 Key Ideas

Conceptually, the breakdown function $\mathcal{A}^k_{\tilde{x}}(\,\cdot\,)_g$ can be obtained in two steps:

1. First, we use the composition $\mathcal{B}^k_{\tilde{x}}([\![\,\cdot\,]\!]^1_g)$, which given a π process returns a μHO process.
2. Second, we use $[\![\,\cdot\,]\!]^2$ to transform the μHO process obtained in (1) into a $\mu\pi$ process.

We illustrate these two steps for output, input, and recursive processes.

Output and Input Processes

Output Given $P = u_i!\langle w_j\rangle.Q$, we first obtain:

$$\mathcal{B}^k_{\tilde{x}}([\![u_i!\langle w_j\rangle.Q]\!]^1_g) = c_k?(\tilde{x}).u_i!\langle W\rangle.\overline{c_{k+3}}!\langle\tilde{x}\rangle \mid \mathcal{B}^{k+3}_{\tilde{x}}([\![Q\sigma]\!]^1_g)$$

where

$$\begin{aligned}\sigma &= (u_i : S)\,?\,\{^{u_{i+1}}/_{u_i}\}\!:\{\}\\ W &= \lambda z_1.\,(\overline{c_{k+1}}!\langle\rangle \mid c_{k+1}?().z_1?(x).\overline{c_{k+2}}!\langle x\rangle \mid c_{k+2}?(x).(x\,\widetilde{w}))\end{aligned}$$

Then, we transform $\mathcal{B}^k_{\tilde{x}}([\![u_i!\langle w_j\rangle.Q]\!]^1_g)$ into the following $\mu\pi$ process, using $[\![\,\cdot\,]\!]^2$:

$$\begin{aligned}\mathcal{A}^k_{\tilde{x}}(u_i!\langle w_j\rangle.Q)_g &= [\![\mathcal{B}^k_{\tilde{x}}([\![u_i!\langle w_j\rangle.Q]\!]^1_g)]\!]^2\\ &= c_k?(\widetilde{x}).(\nu\,a)\,(u_i!\langle a\rangle.(\overline{c_{k+3}}!\langle\widetilde{x}\rangle \mid \mathcal{A}^{k+3}_{\tilde{x}}(Q\sigma)_g \mid\\ &\qquad a?(y).y?(z_1).\overline{c_{k+1}}!\langle z_1\rangle \mid c_{k+1}?(z_1).z_1?(x).\overline{c_{k+2}}!\langle x\rangle \mid\\ &\qquad\quad c_{k+2}?(x).(\nu\,s)\,(x!\langle s\rangle.\overline{s}!\langle\widetilde{w}\rangle)))\end{aligned}$$

We briefly describe the resulting process. The subprocess mimicking the output action on u_i is guarded by an input on c_k. Then, the output of w_j on u_i is mimicked via several steps: first, a private name a is sent along u_i; then, after several redirections involving local trios, the breakdown of w_j is sent along name s. We can see that the output action on u_i enables the forwarding of the context $\tilde{x}$ to the breakdown of Q, the continuation of the output.

Another form of output is when both u_i and/or w_j have recursive types. We shall refer to names with tail-recursive types as *recursive names*. This output case with recursive names is similar to the one just discussed, and given in Table 5.1.

Input The breakdown of $u_i?(w).Q$ is as follows:

$$\begin{aligned}
\mathcal{A}^k_{\tilde{x}}(u_i?(w).Q)_g &= [\![\mathcal{B}^k_{\tilde{x}}([\![u_i?(w).Q]\!]^1_g)]\!]^2 \\
&= c_k?(\tilde{x}).u_i?(y).\overline{c_{k+1}}!\langle \tilde{x}, y\rangle \mid \\
&\quad (\nu\, s_1)\, (c_{k+1}?(\tilde{x}, y).\overline{c_{k+2}}!\langle y\rangle.\overline{c_{k+3}}!\langle \tilde{x}\rangle \mid \\
&\qquad c_{k+2}?(y).(\nu\, s)\, (y!\langle s\rangle.\overline{s}!\langle s_1\rangle) \mid \\
&\qquad\quad c_{k+3}?(\tilde{x}).(\nu\, a)\, (\overline{s_1}!\langle a\rangle.(\overline{c_{k+l+4}}!\langle\rangle \mid c_{k+l+4}?().\mathbf{0} \mid \\
&\qquad\qquad a?(y').y'?(\tilde{w}).(\overline{c_{k+4}}!\langle \tilde{x}\rangle \mid \mathcal{A}^{k+4}_{\tilde{x}}(Q\{^{w_1}/_{w}\}\sigma)_g))))
\end{aligned}$$

In this case, the activation on c_k enables the input on u_i. After several redirections, the actual input of variables $\tilde{w}$ is on a name received for y', which binds them in the breakdown of Q. Hence, the context $\tilde{x}$ does not get extended for an inductive call: it only gets extended locally (it is propagated by c_{k+1}). Indeed, the context is always empty and propagators only enable subsequent actions. The context does play a role in breaking down input actions with recursive names: in that case, the variables z_X (generated in the encoding of recursion, cf. Figure 2.8) get propagated as context.

Recursion

Now, we illustrate the resulting decomposition for processes involving tail-recursive names.

Output The breakdown of $r!\langle w_j\rangle.Q$ when $\mathsf{tr}(r)$ is as follows:

$$\begin{aligned}
\mathcal{A}^k_{\tilde{x}}(r!\langle w_j\rangle.Q) = &\, c_k?(\tilde{x}).(\nu\, a_1)\, c^r!\langle a_1\rangle.(\mathcal{A}^{k+3}_{\tilde{x}}(P)_g \mid a_1?(y_1).y_1?(\tilde{z}).W) \\
&\text{where:} \\
&\quad W = (\nu\, a_2)\, (z_{\iota(S)}!\langle a_2\rangle.(\overline{c_{k+3}}!\langle \tilde{x}\rangle.c^r?(b).(\nu\, s)\, (b!\langle s\rangle.\overline{s}!\langle \tilde{z}\rangle) \mid \\
&\qquad\quad a_2?(y_2).y_2?(z_1').(\overline{c_{k+1}}!\langle\rangle \mid c_{k+1}?().z_1'?(x).\overline{c_{k+2}}!\langle x\rangle \mid \\
&\qquad\qquad c_{k+2}?(x).(\nu\, s')\, (x!\langle s'\rangle.\overline{s'}!\langle \tilde{w}\rangle))))
\end{aligned}$$

This breakdown follows the essential ideas of the case $\neg\mathsf{tr}(r)$, discussed above. The difference is the following: before mimicking the action on r, the process has to obtain the decomposition of name r by a communication on c^r. Again, as a consequence of composing encodings, this communication is carried out via several channel redirections: first, the fresh name a_1 is sent along c^r; then, a name is received on a_1, along which the breakdown of r, denoted by $\tilde{z}$, is finally received. Notice that here we obtain the entire decomposition of r. However, to properly mimic the original output action in W, we need one specific channel from $\tilde{z}$, which is appropriately selected based on its type S by the index function $\iota(\cdot)$ given in Definition 5.2.6—this is the first action of W, i.e., $z_{\iota(S)}$. Finally, the entire decomposition of r is again made available for future trios by communication on c^r. This way, we are able to send back recursive names so they can be used in the next instance of a recursive body.

Recursive process The process $\mu X.P$ is broken down as follows:

$$\begin{aligned}
&(\nu\, s_1)\,(c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle \widetilde{x}\rangle.\overline{c_{k+3}}!\langle \widetilde{x}\rangle \mid\\
&\quad c_{k+1}?(\widetilde{x}).(\nu\, a_1)\,(\overline{s}_1!\langle a_1\rangle.(\overline{c_{k+2}}!\langle\rangle \mid c_{k+2}?() \mid\\
&\qquad c_{k+3}?(\widetilde{x}).s_1?(z_x).\overline{c_{k+4}}!\langle \widetilde{x}, z_x\rangle \mid\\
&\qquad \mathcal{A}^{k+4}_{\widetilde{x},z_x}(P)_{g,\{X\to\tilde{n}\}} \mid\\
&\qquad *\; a_1?(y_1').y_1'?(\|\widetilde{n}^1\|,\ldots,\|\widetilde{n}^m\|, y_1).\widehat{P}\,)))
\end{aligned}$$

where:

$$\begin{aligned}
\widehat{P} = (\nu\,\widetilde{c})\,(&\prod_{0<i\leq m} c^{n_i}?(b).(\nu\, s')\,(b!\langle s'\rangle.\overline{s}'!\langle\|\widetilde{n}^i\|\rangle) \mid\\
&\quad \overline{c_{k+2}}!\langle\widetilde{x}\rangle \mid c_{k+2}?(\widetilde{x}).y_1?(z_x).\overline{c_{k+3}}!\langle\widetilde{x}, z_x\rangle \mid\\
&\quad \llbracket \mathcal{A}^{k+3}_{\widetilde{x},z_x}(P)_{g,\{X\to\tilde{n}\}}\rrbracket_{\tilde{c},\tilde{c}_r})
\end{aligned}$$

Moreover, above we assume that $\widetilde{n} = \mathtt{fn}(P)$, $m = |\widetilde{n}|$, $\|\widetilde{n}\| = (\|n_1\|,\ldots,\|n_m\|)$, $\|n_i\| : S_i$ and $\|\widetilde{n}^i\| = (\|n^i_1\|,\ldots,\|n^i_{|\mathcal{H}(S_i)|}\|)$ for $i \in \{1,\ldots,m\}$.

The breakdown of this process works in coordination with the breakdown of X (described below). The main mechanism here is concerned with controlling (i) the activation of new instances of recursive body (i.e., P) and (ii) the propagation of recursive names to a subsequent instance (i.e., $\widetilde{n} = \mathtt{fn}(P)$). The first instance is given by $\mathcal{A}^{k+4}_{\widetilde{x},z_x}(P)_{g,\{X\to\tilde{n}\}}$. Notice that by the definition of other cases, this instance will first collect all the recursive names by the communication with top-level providers on names c^r for $r \in \widetilde{n}$. The mechanism for generating subsequent instances is given by the replicated process $*\, a_1?(y_1').y_1'?(\|\widetilde{n}^1\|,\ldots,\|\widetilde{n}^m\|, y_1).\widehat{P}$. Recall that replication is supported via the encoding $*P = \mu X.(P \mid X)$. Here, the intention is that this process again receives decompositions of recursive names in $\widetilde{n}$ on shared name a_1 from the breakdown of X: to see this, notice that link to a_1 is propagated to the breakdown of P via name z_x.

Recursive Variable Following the previous description, the variable X is broken down as follows:

$$\begin{aligned}
&(\nu\, s_1)\,(c_k?(z_x).\overline{c_{k+1}}!\langle z_x\rangle.\overline{c_{k+2}}!\langle z_x\rangle \mid\\
&\quad c_{k+2}?(z_x).\overline{s}_1!\langle z_x\rangle.\overline{c_{k+3}}!\langle\rangle \mid c_{k+3}?() \mid\\
&\qquad c_{k+1}?(z_x).(\nu\, a_1)\,(c^{n_1}!\langle a_1\rangle.(a_1?(y_1).y_1?(\widetilde{z}_1).\ldots(\nu\, a_j)\, Q)))
\end{aligned}$$

where:

$$Q = (c^{n_j}!\langle a_j\rangle.(a_j?(y_j).y_j?(\widetilde{z}_j).(\nu\, s')\,(z_x!\langle s'\rangle.\overline{s}'!\langle \widetilde{z}_1,\ldots,\widetilde{z}_j, s_1\rangle)))$$

and $\widetilde{n} = g(X)$, $|\widetilde{n}| = j$, $n_i : S$, and $\widetilde{z}_i = (z^i_1,\ldots,z^i_{|\mathcal{R}^{\star}(S_i)|})$ for $i \in \{1,\ldots,j\}$.

As described above, the main role of this breakdown is to send back decomposition of all recursive names used in a recursive body (given by $\widetilde{n} = g(X)$) to a next instance of a recursive body. This breakdown accomplishes this by (i) first collecting recursive names by communication on c^r for $r \in \widetilde{n}$ with trios of a breakdown of the recursive body, and (ii) the propagating those names to a subsequent instance by communication on z_x. As explained in the previous case, this link is established at the entry of recursive process, and propagated throughout trios of its decomposition up to this process.

5.2.2 Formal Definitions

We start by defining MSTs for π:

$$
\begin{array}{rcl}
C & ::= & M \mid \langle M \rangle \\
\gamma & ::= & \mathsf{end} \mid \mathsf{t} \\
M & ::= & \gamma \mid\ !\langle \widetilde{C} \rangle;\gamma \mid\ ?(\widetilde{C});\gamma \mid \mu \mathsf{t}.M
\end{array}
$$

Figure 5.2: Minimal Session Types for π (cf. Definition 5.2.1)

Definition 5.2.1 (Minimal Session Types, MSTs)**.** *Minimal session types* for π are defined in Figure 5.2.

P	$\mathcal{A}^{k}_{\tilde{x}}(P)_g$	
$u_i!\langle w_j\rangle.Q$	$c_k?(\widetilde{x}).(\nu\, a)\ (u_i!\langle a\rangle.(\overline{c_{k+3}}!\langle\widetilde{x}\rangle \mid \mathcal{A}^{k+3}_{\tilde{x}}(Q\sigma)_g \mid$ $a?(y).y?(z_1).\overline{c_{k+1}}!\langle z_1\rangle \mid$ $c_{k+1}?(z_1).z_1?(x).\overline{c_{k+2}}!\langle x\rangle \mid$ $c_{k+2}?(x).(\nu\, s)\ (x!\langle s\rangle.\overline{s}!\langle\widetilde{w}\rangle)))$	$w_j : C$ $k = j + \lvert\mathcal{H}(C)\rvert - 1$ $\widetilde{w} = (w_j, \ldots, w_k)$ $\sigma = \mathsf{next}(u_i)$
$u_i?(w).Q$	$c_k?(\widetilde{x}).u_i?(y).\overline{c_{k+1}}!\langle\widetilde{x}, y\rangle \mid$ $(\nu\, s_1)\ (c_{k+1}?(\widetilde{x}, y).\overline{c_{k+2}}!\langle y\rangle.\overline{c_{k+3}}!\langle\widetilde{x}\rangle \mid$ $c_{k+2}?(y).(\nu\, s)\ (y!\langle s\rangle.\overline{s}!\langle s_1\rangle) \mid$ $c_{k+3}?(\widetilde{x}).(\nu\, a)\ (\overline{s_1}!\langle a\rangle.$ $(\overline{c_{k+l+4}}!\langle\rangle \mid c_{k+l+4}?().\mathbf{0} \mid \widehat{Q}_{\tilde{x}}\ )))$ where: $\widehat{Q}_{\tilde{x}} = a?(y').y'?(\widetilde{w}).(\overline{c_{k+4}}!\langle\widetilde{x}\rangle \mid \mathcal{A}^{k+4}_{\widetilde{x}}(Q\{w_1/w\}\sigma)_g)$	$w : C$ $\widetilde{w} = (w_1, \ldots, w_{\lvert\mathcal{H}(C)\rvert})$ $l = \lfloor Q\rceil$ $\sigma = \mathsf{next}(u_i)$
$r_i!\langle w_j\rangle.P$	$c_k?(\widetilde{x}).(\nu\, a_1)\ c^r!\langle a_1\rangle.$ $(\mathcal{A}^{k+3}_{\tilde{x}}(P)_g \mid a_1?(y_1).y_1?(\widetilde{z}).W)$ where: $W = (\nu\, a_2)\ (z_{\iota(S)}!\langle a_2\rangle.$ $(\overline{c_{k+3}}!\langle\widetilde{x}\rangle.c^r?(b).(\nu\, s)\ (b!\langle s\rangle.\overline{s}!\langle\widetilde{z}\rangle) \mid$ $a_2?(y_2).y_2?(z'_1).$ $(\overline{c_{k+1}}!\langle\rangle \mid\ c_{k+1}?().z'_1?(x).\overline{c_{k+2}}!\langle x\rangle \mid$ $c_{k+2}?(x).(\nu\, s')\ (x!\langle s'\rangle.\overline{s}'!\langle\widetilde{w}\rangle))))$	$r : S \wedge \mathsf{tr}(S)$ $\widetilde{z} = (z_1, \ldots, z_{\lvert\mathcal{R}'^{\star}(S)\rvert})$ $\widetilde{c} = (c_{k+1}, c_{k+2})$ $w : C$ $k = j + \lvert\mathcal{H}(C)\rvert - 1$ $\widetilde{w} = (w_j, \ldots, w_k)$
$r_i?(w).P$	$c_k?(\widetilde{x}).(\nu\, a_1)$ $(c^r!\langle a_1\rangle.((\nu\, s_1)\ (c_{k+1}?(y).\overline{c_{k+2}}!\langle y\rangle.\overline{c_{k+3}}!\langle\rangle \mid$ $c_{k+2}?(y).(\nu\, s)\ (y!\langle s\rangle.\overline{s}!\langle s_1\rangle) \mid$ $c_{k+3}?().(\nu\, a_2)\ (s_1!\langle a_2\rangle.(\overline{c_{k+l+4}}!\langle\rangle \mid c_{k+l+4}?().\mathbf{0} \mid$ $a_2?(y_2).y_2?(\widetilde{w}).(\overline{c_{k+4}}!\langle\widetilde{x}\rangle \mid \mathcal{A}^{k+4}_{\tilde{x}}(P\{w_1/w\})_g))) \mid$ $a_1?(y_1).y_1?(\widetilde{z}).z_{\iota(S)}?(y).$ $\overline{c_{k+1}}!\langle y\rangle.c^r?(b).(\nu\, s')\ (b!\langle s'\rangle.\overline{s}'!\langle\widetilde{z}\rangle))))$	$r : S \wedge \mathsf{tr}(S)$ $\widetilde{z} = (z_1, \ldots, z_{\lvert\mathcal{R}'^{\star}(S)\rvert})$ $l = \lfloor P\rceil$ $w : C$ $\widetilde{w} = (w_1, \ldots, w_{\lvert\mathcal{H}(C)\rvert})$

Table 5.1: Decompose by composition (Part 1/2): Breakdown function $\mathcal{A}^{k}_{\tilde{x}}(\,\cdot\,)_g$ (cf. Definition 5.2.7).

The breakdown function $\mathcal{A}^{k}_{\tilde{x}}(\,\cdot\,)_g$ for all constructs of π is given in Tables 5.1 and 5.2; it relies on several auxiliary definitions, most notably:

- The *degree* of a process P, denoted $\lfloor P\rceil$;
- The predicate $\mathsf{tr}(C)$, which indicates that C is a tail-recursive session type;

P	$\mathcal{A}^k_{\tilde{x}}(P)_g$	
$(\nu\, s)\, P'$	$(\nu\, \widetilde{s} : \mathcal{H}(C))\ \mathcal{A}^k_{\tilde{x}}(P'\sigma)_g$	$s : C$ $\widetilde{s} = (s_1, \ldots, s_{\lvert\mathcal{H}(C)\rvert})$ $\sigma = \{s_1\overline{s_1}/s\overline{s}\}$
$(\nu\, r)\, P'$	$(\nu\, \widetilde{r} : \mathcal{R}'(S))\ c^r?(b).(\nu\, s')\ (b!\langle s'\rangle.\overline{s}'!\langle\widetilde{r}\rangle) \mid$ $c^{\overline{r}}?(b).(\nu\, s')\ (b!\langle s'\rangle.\overline{s}'!\langle\widetilde{\overline{r}}\rangle) \mid \mathcal{A}^k_{\tilde{x}}(P'\sigma)_g$	$r : \mu \mathsf{t}.S$ $\mathsf{tr}(\mu\mathsf{t}.S)$ $\sigma = \{r_1\overline{r_1}/r\overline{r}\}$ $\widetilde{r} = (r_1, \ldots, r_{\lvert\mathcal{R}'(S)\rvert})$ $\widetilde{\overline{r}} = (\overline{r}_1, \ldots, \overline{r}_{\lvert\mathcal{R}'(S)\rvert})$
$Q_1 \mid Q_2$	$c_k?(\tilde{x}).\overline{c_{k+1}}!\langle\tilde{y}\rangle.\overline{c_{k+l+1}}!\langle\tilde{z}\rangle \mid$ $\mathcal{A}^{k+1}_{\tilde{y}}(Q_1)_g \mid \mathcal{A}^{k+l+1}_{\tilde{z}}(Q_2)_g$	$\widetilde{y} = \mathtt{fv}(Q_1)$ $\widetilde{z} = \mathtt{fv}(Q_2)$ $l = \lfloor Q \rceil$
$\mathbf{0}$	$c_k?().\mathbf{0}$	
$\mu X.P$	$(\nu\, s_1)\ (c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle\widetilde{x}\rangle.\overline{c_{k+3}}!\langle\widetilde{x}\rangle \mid$ $c_{k+1}?(\widetilde{x}).(\nu\, a_1)\ (s_1!\langle\overline{a}_1\rangle.(\overline{c_{k+2}}!\langle\rangle \mid c_{k+2}?() \mid$ $c_{k+3}?(\widetilde{x}).s_1?(z_x).\overline{c_{k+4}}!\langle\widetilde{x}, z_x\rangle \mid$ $\mathcal{A}^{k+4}_{\tilde{x},z_x}(P)_{g,\{X\to\tilde{n}\}} \mid$ $* \, a_1?(y_1').y_1'?(\lVert\widetilde{n}^1\rVert, \ldots, \lVert\widetilde{n}^m\rVert, y_1).\widehat{P}\,)))$ where: $\widehat{P} = (\nu\, \widetilde{c})\ (\prod_{0<i\leq m} c^{n_i}?(b).(\nu\, s')\ (b!\langle s'\rangle.\overline{s}'!\langle\lVert\widetilde{n}^i\rVert\rangle) \mid$ $\overline{c_{k+2}}!\langle\widetilde{x}\rangle \mid\ c_{k+2}?(\widetilde{x}).y_1?(z_x).\overline{c_{k+3}}!\langle\widetilde{x}, z_x\rangle \mid$ $\llbracket \mathcal{A}^{k+3}_{\tilde{x},z_x}(P)_{g,\{X\to\tilde{n}\}} \rrbracket_{\tilde{c},\tilde{c}_r})$	$\widetilde{n} = \mathtt{fn}(P)$ $m = \lvert\widetilde{n}\rvert$ $\lVert\widetilde{n}\rVert = (\lVert n_1\rVert, \ldots, \lVert n_m\rVert)$ $i \in \{1, \ldots, m\}.$ $\lVert n_i\rVert : S_i$ $\lVert\widetilde{n}^i\rVert = (\lVert n^i_1\rVert, \ldots, \lVert n^i_{\lvert\mathcal{H}(S_i)\rvert}\rVert)$ $\widetilde{c} = (c_{k+2}, \ldots,$ $c_{k+\lfloor \lVert P\rVert^1_{g,\{X\to\tilde{n}\}} \rceil+1})$ $\widetilde{c}_r = \bigcup_{v\in\widetilde{n}} c^v$
X	$(\nu\, s_1)\ (c_k?(z_x).\overline{c_{k+1}}!\langle z_x\rangle.\overline{c_{k+2}}!\langle z_x\rangle \mid$ $c_{k+2}?(z_x).\overline{s}_1!\langle z_x\rangle.\overline{c_{k+3}}!\langle\rangle \mid c_{k+3}?() \mid$ $c_{k+1}?(z_x).(\nu\, a_1)\ (c^{n_1}!\langle a_1\rangle.$ $(a_1?(y_1).y_1?(\widetilde{z}_1).\ldots.(\nu\, a_j)\, Q)))$ where: $Q = (c^{n_j}!\langle a_j\rangle.(a_j?(y_j).y_j?(\widetilde{z}_j).$ $(\nu\, s')\ (z_x!\langle s'\rangle.\overline{s}'!\langle\widetilde{z}_1, \ldots, \widetilde{z}_j, s_1\rangle)))$	$\widetilde{n} = g(X)$ $\lvert\widetilde{n}\rvert = j$ $i \in \{1, \ldots, j\}$ $n_i : S \wedge \mathsf{tr}(S_i)$ $\widetilde{z}_i = (z^i_1, \ldots, z^i_{\lvert\mathcal{R}'^\star(S_i)\rvert})$

Table 5.2: Decompose by composition (Part 2/2): Breakdown function $\mathcal{A}^k_{\tilde{x}}(\,\cdot\,)_g$ (cf. Definition 5.2.7).

- The functions $\mathcal{H}(\,\cdot\,)$ and $\mathcal{R}'(\,\cdot\,)$, which decompose session types into MSTs.

We now formally define these notions.

Definition 5.2.2 (Degree of a Process)**.** The *degree* of a process P, denoted $\lfloor P \rceil$, is defined as:

$$
\begin{array}{rclrclrcl}
\lfloor u_i!\langle w_j\rangle.Q \rceil & = & \lfloor Q \rceil + 3 & \lfloor (\nu\, s : S)\, Q \rceil & = & \lfloor Q \rceil & \lfloor \mathbf{0} \rceil & = & 1 \\
\lfloor u_i?(x : C).Q \rceil & = & \lfloor Q \rceil + 5 & \lfloor Q \mid R \rceil & = & \lfloor Q \rceil + \lfloor R \rceil + 1 & & & \\
\lfloor X \rceil & = & 4 & \lfloor \mu X.Q \rceil & = & \lfloor Q \rceil + 4 & & &
\end{array}
$$

Definition 5.2.3 (Predicates on Types and Names)**.** Given a session type C, we write $\mathsf{tr}(C)$ to indicate that C is a tail-recursive session type. Also, given $u : C$, we write $\mathtt{lin}(u)$ if $C = S$ and $\neg\mathsf{tr}(S)$.

Definition 5.2.4 (Subsequent index substitution)**.** Let n_i be an indexed name. We define $\mathsf{next}(n_i) = (\mathtt{lin}(n_i))\,?\,\{^{n_{i+1}}/_{n_i}\}\colon \{\}$.

We define how to obtain MSTs for π from standard session types:

Definition 5.2.5 (Decomposing First-Order Types)**.** The decomposition function $\mathcal{H}(\,\cdot\,)$ on finite types, obtained by combining the mappings $\langle\!\langle\,\cdot\,\rangle\!\rangle^1, \mathcal{G}(\cdot)$, and $\langle\!\langle\,\cdot\,\rangle\!\rangle^2$, is defined in Figure 5.3 (top, where omitted cases are defined homomorphically). It is extended to account for recursive session types in Figure 5.3 (center).

The auxiliary function $\mathcal{R}'^{\star}(\,\cdot\,)$, given in Figure 5.3 (bottom), is used in Table 5.1 to decompose *guarded* tail-recursive types: it skips session prefixes until a type of form $\mu \mathsf{t}.S$ is encountered; when that occurs, the recursive type is decomposed using $\mathcal{R}'(\,\cdot\,)$.

We give the definition of the index function for π recursive types by composing the index function, denoted $[\cdot\rangle$, for HO recursive types (given in Definition 4.3.5) and the encoding of types from Figure 2.8 and Figure 2.9. This composition is straightforward: notice that $[\cdot\rangle$ counts prefixes of session types, and that the encodings of types from Figures 2.8 and 2.9 do not alter the number of prefixes.

Definition 5.2.6 (Index function)**.** Let S be an (unfolded) recursive session type. The function $\iota(S)$ is defined as follows:

$$\iota(S) = \begin{cases} \widehat{\iota}_0(S'\{^{S}/_{\mathsf{t}}\}) & \text{if } S = \mu \mathsf{t}.S' \\ \widehat{\iota}_0(S) & \text{otherwise} \end{cases} \qquad \widehat{\iota}_l(T) = \begin{cases} \widehat{\iota}_{l+1}(S) & \text{if } T = !\langle U\rangle;S \text{ or } T = ?(U);S \\ |\mathcal{R}'(S)| - l + 1 & \text{if } T = \mu \mathsf{t}.S \end{cases}$$

We are finally ready to define the decomposition function $\mathcal{F}(\cdot)$, the analog of Definition 4.3.9 but for processes in π:

Definition 5.2.7 (Process Decomposition)**.** Let P be a closed π process with $\widetilde{u} = \mathtt{fn}(P)$ and $\widetilde{v} = \mathtt{rn}(P)$. Given the breakdown function $\mathcal{A}^k_{\tilde{x}}(\,\cdot\,)_g$ in Tables 5.1 and 5.2, the decomposition $\mathcal{F}(P)$ is defined as:

$$\mathcal{F}(P) = (\nu\,\widetilde{c})\,(\nu\,\widetilde{c}_r)\Big(\overline{c_k}!\langle\rangle.\mathbf{0} \mid \mathcal{A}^k_\epsilon(P\sigma)_g \mid \prod_{r\in\tilde{v}} c^r?(b).(\nu\,s)\,(b!\langle s\rangle.\overline{s}!\langle\widetilde{r}\rangle)\Big)$$

where: $k > 0$, $\widetilde{c} = (c_k, \ldots, c_{k+\lfloor P\rceil-1})$; $\widetilde{c_r} = \bigcup_{r\in\tilde{v}} c^r$; $\sigma = \{^{\mathsf{init}(\tilde{u})}/_{\tilde{u}}\}$; for each $r \in \tilde{v}$, we have $r : S$ and $\widetilde{r} = r_1, \ldots, r_{|\mathcal{G}(S)|}$.

5.2.3 Examples

Example 5.2.1 (A Process with Delegation)**.** Let P be a π process which incorporates name-passing and implements channels u and $\overline{w}$ with types $S = !\langle \overline{T}\rangle;\mathtt{end}$ and $T = ?(\mathsf{Int});!\langle\mathsf{Bool}\rangle;\mathtt{end}$, respectively:

$$P = (\nu\,u : S)\,\big(\underbrace{u!\langle w\rangle.\overline{w}?(t).\overline{w}!\langle \mathsf{odd}(t)\rangle.\mathbf{0}}_{R} \mid \underbrace{\overline{u}?(x).x!\langle 5\rangle.x?(b).\mathbf{0}}_{Q}\big)$$

The degree of P is $\lfloor P\rceil = 25$. Then, the decomposition of P into a collection of first-order processes typed with minimal session types is:

$$\mathcal{F}(P) = (\nu\,c_1,\ldots,c_{25})\,(\overline{c_1}!\langle\rangle.\mathbf{0} \mid (\nu\,u_1)\,\mathcal{A}^1_\epsilon((R \mid Q)\sigma'))$$

where $\sigma = \mathsf{init}(\mathtt{fn}(P))$ and $\sigma' = \sigma\cdot\{^{u_1\overline{u_1}}/_{u\overline{u}}\}$. We omit parameter g, as it is empty. We have:

$$\mathcal{A}^1_\epsilon((R \mid Q)\sigma')) = c_1?().\overline{c_2}!\langle\rangle.\overline{c_{13}}!\langle\rangle \mid \mathcal{A}^2_\epsilon(R\sigma') \mid \mathcal{A}^{13}_\epsilon(Q\sigma')$$

$$\mathcal{H}(!\langle C\rangle;S) = \begin{cases} M & \text{if } S = \mathsf{end} \\ M, \mathcal{H}(S) & \text{otherwise} \end{cases}$$
$$\text{where } M = !\langle\langle ?(?(\langle ?(\mathcal{H}(C));\mathsf{end}\rangle);\mathsf{end});\mathsf{end}\rangle\rangle;\mathsf{end}$$
$$\mathcal{H}(?(C);S) = \begin{cases} M & \text{if } S = \mathsf{end} \\ M, \mathcal{H}(S) & \text{otherwise} \end{cases}$$
$$\text{where } M = ?(\langle ?(?(\langle ?(\mathcal{H}(C));\mathsf{end}\rangle);\mathsf{end});\mathsf{end}\rangle);\mathsf{end}$$
$$\mathcal{H}(\mathsf{end}) = \mathsf{end}$$
$$\mathcal{H}(S_1, \dots, S_n) = \mathcal{H}(S_1), \dots, \mathcal{H}(S_n)$$
$$\mathcal{H}(\langle S\rangle) = \langle \mathcal{H}(S)\rangle$$

$$\mathcal{H}(\mu \mathsf{t}.S) = \begin{cases} \mathcal{R}'(S) & \text{if } \mu\mathsf{t}.S \text{ is tail-recursive} \\ \mu\mathsf{t}.\mathcal{H}(S) & \text{otherwise} \end{cases}$$
$$\mathcal{R}'(!\langle S\rangle;S') = \mu\mathsf{t}.!\langle\langle ?(?(\langle ?(\mathcal{H}(S));\mathsf{end}\rangle);\mathsf{end});\mathsf{end}\rangle\rangle;\mathsf{t}, \mathcal{R}'(S')$$
$$\mathcal{R}'(?(S);S') = \mu\mathsf{t}.?(\langle ?(?(\langle ?(\mathcal{H}(S));\mathsf{end}\rangle);\mathsf{end});\mathsf{end}\rangle);\mathsf{t}, \mathcal{R}'(S')$$
$$\mathcal{H}(\mathsf{t}) = \mathsf{t}$$
$$\mathcal{R}'(\mathsf{t}) = \epsilon$$

$$\mathcal{R}'^{\star}(?(S);S') = \mathcal{R}'^{\star}(S') \qquad \mathcal{R}'^{\star}(!\langle S\rangle;S') = \mathcal{R}'^{\star}(S') \qquad \mathcal{R}'^{\star}(\mu\mathsf{t}.S) = \mathcal{R}'^{\star}(S)$$

Figure 5.3: Decomposition of session types $\mathcal{H}(\,\cdot\,)$ (cf. Definition 5.2.5)

We use some abbreviations for subprocesses of R and Q :

$$R' = \overline{w_1}?(t).R'' \qquad R'' = \overline{w_1}!\langle \mathsf{odd}(t)\rangle.\mathbf{0}$$
$$Q' = x_1!\langle 5\rangle.Q'' \qquad Q'' = x_1?(b).\mathbf{0}$$

The breakdown of R is:

$$\begin{aligned}
\mathcal{A}^2_\epsilon(R) = {} & c_2?().(\nu\, a_1)\, (u_1!\langle a_1\rangle.(\overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(R') \mid \\
& \quad a_1?(y_1).y_1?(z_1).\overline{c_3}!\langle z_1\rangle \mid c_3?(z_1).z_1?(x).\overline{c_4}!\langle x\rangle \mid \\
& \quad c_4?(x).(\nu\, s)\, (x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle)))) \\
\mathcal{A}^5_\epsilon(R') = {} & c_5?().\overline{w_1}?(y_2).\overline{c_6}!\langle y_2\rangle \mid \\
& (\nu\, s_1)\, (c_6?(y_2).\overline{c_7}!\langle y_2\rangle.\overline{c_8}!\langle\rangle \mid \\
& \quad c_7?(y_2).(\nu\, s')\, (y_2!\langle s'\rangle.\overline{s'}!\langle s_1\rangle.\mathbf{0})) \mid \\
& \quad c_8?().(\nu\, a_2)\, (\overline{s_1}!\langle a_2\rangle.(\overline{c_{10}}!\langle\rangle \mid c_{10}?().\mathbf{0} \mid \\
& \quad\quad a_2?(y_3).y_3?(t_1).(\overline{c_9}!\langle\rangle \mid \mathcal{A}^9_\epsilon(R'')))) \\
\mathcal{A}^9_\epsilon(R'') = {} & c_9?().(\nu\, a)\, (\overline{w_2}!\langle a\rangle.(\overline{c_{12}}!\langle\rangle \mid c_{12}?().\mathbf{0} \mid \\
& \quad a?(y).y?(z_1).\overline{c_{11}}!\langle z_1\rangle \mid c_{10}?(z_1).z_1?(x).\overline{c_{11}}!\langle x\rangle \mid \\
& \quad c_{11}?(x).(\nu\, s)\, (x!\langle s\rangle.\overline{s}!\langle \mathsf{odd}(t)\rangle)))
\end{aligned}$$

The breakdown of Q is:

$$\begin{aligned}
\mathcal{A}^{13}_{\epsilon}(Q) &= c_{13}?().\overline{u_1}?(y_4).\overline{c_{14}}!\langle y_4\rangle \mid (\nu\, s_1)\,(c_{14}?(y).\overline{c_{15}}!\langle y\rangle.\overline{c_{16}}!\langle\rangle \mid \\
&\qquad c_{15}?(y_4).(\nu\, s'')\,(y_4!\langle s''\rangle.\overline{s''}!\langle s_1\rangle.\mathbf{0}) \mid c_{16}?().(\nu\, a_3)\,(s_1!\langle a_3\rangle.(\overline{c_{21}}!\langle\rangle \mid \\
&\qquad\quad c_{21}?().\mathbf{0} \mid a_3?(y_5).y_5?(x_1,x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_{\epsilon}(Q'))))) \\
\mathcal{A}^{17}_{\epsilon}(Q') &= c_{17}?().(\nu\, a_4)\,(x_1!\langle a_4\rangle.(\overline{c_{20}}!\langle\rangle \mid \mathcal{A}^{20}_{\epsilon}(Q'') \mid a_4?(y_6).y_6?(z_1).\overline{c_{18}}!\langle z_1\rangle \mid \\
&\qquad c_{18}?(z_1).z_1?(x).\overline{c_{19}}!\langle x\rangle \mid c_{19}?(x).(\nu\, s''')\,(x!\langle s'''\rangle.\overline{s'''}!\langle 5\rangle))) \\
\mathcal{A}^{20}_{\epsilon}(Q'') &= c_{20}?().x_2?(y).\overline{c_{21}}!\langle y\rangle \mid \\
&\qquad (\nu\, s_1)\,(c_{21}?(y).\overline{c_{22}}!\langle y\rangle.\overline{c_{23}}!\langle\rangle \mid c_{22}?(y).(\nu\, s)\,(y!\langle s\rangle.\overline{s}!\langle s_1\rangle) \mid \\
&\qquad\quad c_{23}?(\widetilde{x}).(\nu\, a)\,(\overline{s_1}!\langle a\rangle.(\overline{c_{25}}!\langle\rangle \mid c_{25}?().\mathbf{0} \mid \\
&\qquad\quad\quad a?(y').y'?(b_1).(\overline{c_{24}}!\langle\rangle \mid c_{24}?().\mathbf{0}))))
\end{aligned}$$

Type S is broken down into MSTs M_1 and M_2, as follows:

$$\begin{aligned}
M_1 &= ?(\langle ?(?(\langle ?(\mathsf{Int});\mathtt{end}\rangle);\mathtt{end});\mathtt{end}\rangle);\mathtt{end} \\
M_2 &= !\langle\langle ?(?(\langle ?(\mathsf{Bool});\mathtt{end}\rangle);\mathtt{end});\mathtt{end}\rangle\rangle;\mathtt{end}
\end{aligned}$$

Names $\overline{w}_1$ and $\overline{w}_2$ are typed with M_1 and M_2, respectively. Then, name u_1 is typed with M, given by:

$$M = !\langle\langle ?(?(\langle ?(\overline{M}_1, \overline{M}_2);\mathtt{end}\rangle);\mathtt{end});\mathtt{end}\rangle\rangle;\mathtt{end}$$

Consider the reductions of $\mathcal{F}(P)$ that mimic the exchange of w along u in P. We first have three synchronizations on c_1, c_2, c_{13}:

$$\begin{aligned}
\mathcal{F}(P) \longrightarrow^3\ & (\nu\, \widetilde{c})\,(\,(\nu\, a_1)\,(\,u_1!\langle a_1\rangle.\,(\overline{c_5}!\langle\rangle \mid \mathcal{A}^{5}_{\epsilon}(R') \mid \\
& \quad a_1?(y_1).y_1?(z_1).\overline{c_3}!\langle z_1\rangle \mid c_3?(z_1).z_1?(x).\overline{c_4}!\langle x\rangle \mid \\
& \quad c_4?(x).(\nu\, s)\,(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle))) \mid \overline{u_1}?(y_4).\,\overline{c_{14}}!\langle y_4\rangle \mid \\
& \quad (\nu\, s_1)\,(c_{14}?(y).\overline{c_{15}}!\langle y\rangle.\overline{c_{16}}!\langle\rangle \mid \\
& \qquad c_{15}?(y_4).(\nu\, s'')\,(y_4!\langle s''\rangle.\overline{s''}!\langle s_1\rangle.\mathbf{0}) \mid \\
& \qquad c_{16}?().(\nu\, a_3)\,(s_1!\langle a_3\rangle.(\overline{c_{21}}!\langle\rangle \mid \\
& \qquad\quad c_{21}?().\mathbf{0} \mid a_3?(y_5).y_5?(x_1,x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_{\epsilon}(Q')))))
\end{aligned}$$

where $\widetilde{c} = (c_3, \ldots, c_{12}, c_{14}, \ldots, c_{25})$. Then, the broken down process R communicates with process Q through channel u_1 by passing name a_1 (highlighted above). Here we notice that the original transmission of value w is not immediately mimicked on channel u, but it is delegated to some other channel through a series of channel redirections starting with the transmission of name a_1. Further, the received name a_1 is locally propagated by c_{14} and c_{15}. This represents redundant communications on propagators induced by breaking down sequential prefixes produced by two encodings $[\![\cdot]\!]^1_g$ and $[\![\cdot]\!]^2$ (i.e., those communications are not present in P). Another synchronization occurs on c_{16}.

$$\begin{aligned}
\mathcal{F}(P) \longrightarrow^7\ & (\nu\, \widetilde{c}_*)\,(\nu\, a_1)\,(\,\overline{c_5}!\langle\rangle \mid \mathcal{A}^{5}_{\epsilon}(R') \mid a_1?(y_1).\,y_1?(z_1).\overline{c_3}!\langle z_1\rangle \mid \\
& c_3?(z_1).z_1?(x).\overline{c_4}!\langle x\rangle \mid c_4?(x).(\nu\, s)\,(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle) \mid \\
& (\nu\, s_1)\,((\nu\, s'')\,(\,a_1!\langle s''\rangle.\,\overline{s''}!\langle s_1\rangle.\mathbf{0}) \mid (\nu\, a_3)\,(s_1!\langle a_3\rangle.(\overline{c_{21}}!\langle\rangle \mid \\
& \quad c_{21}?().\mathbf{0} \mid a_3?(y_5).y_5?(x_1,x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_{\epsilon}(Q'))))))
\end{aligned}$$

where $\widetilde{c}_* = (c_3, \ldots, c_{12}, c_{17}, \ldots, c_{25})$.

The next step involves a communication on a_1: session name s'' is passed and substitutes variable y_1.

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^8\ & (\nu\, \widetilde{c}_*)\,(\nu\, s'')\,(\, \overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(R') \mid \\ & s''?(z_1).\, \overline{c_3}!\langle z_1\rangle \mid c_3?(z_1).z_1?(x).\overline{c_4}!\langle x\rangle \mid \\ & c_4?(x).(\nu\, s)\,(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle) \mid \\ & (\nu\, s_1)\,(\, \overline{s''}!\langle s_1\rangle.\, \mathbf{0} \mid (\nu\, a_3)\,(s_1!\langle a_3\rangle.(\overline{c_{21}}!\langle\rangle \mid c_{21}?().\mathbf{0} \mid \\ & \quad a_3?(y_5).y_5?(x_1, x_2).(\, \overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(Q'))))))\end{aligned}$$

After the synchronization on channel s'', name z_1 is further sent to the next parallel process through the propagator c_3:

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^{10}\ & (\nu\, \widetilde{c}_{**})\,(\nu\, s_1)\,(\, \overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(R') \mid \\ & s_1?(x).\, \overline{c_4}!\langle x\rangle \mid c_4?(x).(\nu\, s)\,(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle) \mid \\ & (\nu\, a_3)\,(\, s_1!\langle a_3\rangle.\, (\overline{c_{21}}!\langle\rangle \mid c_{21}?().\mathbf{0} \mid a_3?(y_5).y_5?(x_1, x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(Q')))))\end{aligned}$$

where $\widetilde{c}_{**} = (c_4, \ldots, c_{12}, c_{17}, \ldots, c_{25})$.

Communication on s_1 leads to variable x being substituted by name a_3, which is then passed on c_4 to the next process. In addition, inaction is simulated by synchronization on c_{21}.

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^{13}\ & (\nu\, \widetilde{c}_\bullet)\,(\nu\, a_3)\,(\, \overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(R') \mid (\nu\, s)\,(\, a_3!\langle s\rangle.\, \overline{s}!\langle w_1, w_2\rangle) \mid \\ & a_3?(y_5).\, y_5?(x_1, x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(Q'))\,)\end{aligned}$$

where $\widetilde{c}_\bullet = (c_5, \ldots, c_{12}, c_{17}, \ldots, c_{25})$.

Now, the distribution of the decomposition of w from one process to another can finally be simulated by two reductions: first, a synchronization on a_3 sends the endpoint of session s, which replaces variable y_5; afterwards, the dual endpoint is used to send the names w_1, w_2, substituting the variables x_1, x_2.

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^{14}\ & (\nu\, \widetilde{c}_{\bullet\bullet})\,(\nu\, s)\,(\, \overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(R') \mid \overline{s}!\langle w_1, w_2\rangle \mid s?(x_1, x_2).\, (\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(Q'))) \\ \longrightarrow\ & (\nu\, \widetilde{c}_{\bullet\bullet})\,(\, \overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(R') \mid \overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(Q')\{w_1 w_2/x_1 x_2\}) = V\end{aligned}$$

where $\widetilde{c}_{\bullet\bullet} = (c_5, \ldots, c_{12}, c_{17}, \ldots, c_{25})$.

Here, we remark that prefix $s?(x_1, x_2)$ binds variables x_1, x_2 in the breakdown of the continuation (i.e., $\mathcal{A}^{17}_\epsilon(Q')$). Thus, there is no need for propagators to pass contexts: propagators here only serve to enforce the ordering of actions. On the other hand, this relies on a process nesting that is induced by the application of encoding $[\![\cdot]\!]^2$ in the composition. Thus, the trio structure is lost.

Undoubtedly, the first action of the original first-order process has been simulated. We may notice that in V names w_1, w_2 substitute x_1, x_2 and the subsequent τ-action on w can be simulated on name w_1. The following reductions follow the same pattern. Thus, the outcome of our decomposition function is a behaviorally equivalent process that is typed with MSTs.

Example 5.2.2 (A Recursive Process)**.** Let r be a channel with the tail-recursive session type $S = \mu \mathtt{t}.?(\mathtt{Int});!\langle\mathtt{Int}\rangle;\mathtt{t}$. We decompose r using S and obtain two channels typed with MSTs as in Figure 5.3:

$$\begin{aligned} r_1 &: \mu\mathtt{t}.?(\langle ?(?(\langle ?(\mathtt{Int});\mathtt{end}\rangle);\mathtt{end});\mathtt{end}\rangle);\mathtt{t} \\ r_2 &: \mu\mathtt{t}.!\langle\langle ?(?(\langle ?(\mathtt{Int});\mathtt{end}\rangle);\mathtt{end});\mathtt{end}\rangle\rangle;\mathtt{t}\end{aligned}$$

$$\begin{aligned}
\mathcal{A}^1_\epsilon(P\{{}^{r_1}\!/r\})_\emptyset = (\nu\, s_1)\,(c_1?().\overline{c_2}!\langle\rangle.\overline{c_4}!\langle\rangle \mid{} \\
c_2?().(\nu\, a_1)\,(s_1!\langle a_1\rangle.(\overline{c_3}!\langle\rangle \mid c_3?().\mathbf{0} \mid{} \\
c_4?().s_1?(z_x).\overline{c_5}!\langle z_x\rangle \mid{} \\
R^5 \mid *\, a_1?(y_1').y_1'?(x_{r_1}, x_{r_2}, y_1).\widehat{P}\,)))
\end{aligned}$$

where:

$$\begin{aligned}
\widehat{P} &= (\nu\, \widetilde{c})\,(c^r?(b).(\nu\, s')\,(b!\langle s'\rangle.\overline{s'}!\langle x_{r_1}, x_{r_2}\rangle) \mid \overline{c_1}!\langle\rangle \mid{} \\
&\qquad c_1?().y_1?(z_x).\overline{c_2}!\langle z_x\rangle \mid R^2\{{}^{x_{r_1}, x_{r_2}}\!/r_1, r_2\}) \\
R^k &= c_k?(z_x). \\
&\quad (\nu\, a_1)\,(c^r!\langle a_1\rangle.((\nu\, s_1)\,(c_{k+1}?(y).\overline{c_{k+2}}!\langle y\rangle.\overline{c_{k+3}}!\langle\rangle \mid{} \\
&\quad c_{k+2}?(y).(\nu\, s)\,(y!\langle s\rangle.\overline{s}!\langle s_1\rangle) \mid{} \\
&\qquad c_{k+3}?().(\nu\, a_2)\,(s_1!\langle a_2\rangle.(\overline{c_{k+l+4}}!\langle\rangle \mid{} \\
&\qquad\quad c_{k+l+4}?().\mathbf{0} \mid a_2?(y_2).\, y_2?(w_1). \\
&\qquad\qquad (\nu\, \widetilde{c})\,(\overline{c_{k+4}}!\langle z_x\rangle \mid \mathcal{A}^{k+4}_{z_x}(r_2!\langle -w_1\rangle.X)_g))) \mid{} \\
&\qquad\qquad\quad a_1?(y_1).y_1?(z_1, z_2).\, z_1?(y). \\
&\qquad\qquad\quad \overline{c_{k+1}}!\langle y\rangle.c^r?(b).(\nu\, s')\,(b!\langle s'\rangle.\overline{s'}!\langle z_1, z_2\rangle)))) \\
\mathcal{A}^{k+4}_{z_x}(r_2!\langle -w_1\rangle.X)_g &= c_k?(z_x).(\nu\, a_1)\, c^r!\langle a_1\rangle. \\
&\qquad (\mathcal{A}^{k+7}_{z_x}(X)_g \mid a_1?(y_1).y_1?(\widetilde{z}).W) \\
W &= (\nu\, a_2)\,(\, z_2!\langle a_2\rangle.\, (\overline{c_{k+7}}!\langle z_x\rangle.c^r?(b).(\nu\, s)\,(b!\langle s\rangle.\overline{s}!\langle \widetilde{z}\rangle) \mid{} \\
&\quad a_2?(y_2).y_2?(z_1').(\nu\, \widetilde{c})\,(\overline{c_{k+5}}!\langle\rangle \mid{} \\
&\qquad\qquad c_{k+5}?().z_1'?(x).\overline{c_{k+6}}!\langle x\rangle \mid{} \\
&\qquad\qquad\quad c_{k+6}?(x).(\nu\, s')\,(x!\langle s'\rangle.\, \overline{s'}!\langle -w1\rangle\,)))) \\
\mathcal{A}^{k+7}_{z_x}(X)_g &= (\nu\, s_1)\,(c_{k+7}?(z_x).\overline{c_{k+8}}!\langle z_x\rangle.\overline{c_{k+9}}!\langle z_x\rangle \mid{} \\
&\quad c_{k+8}?(z_x).(\nu\, a_1)\,(c^r!\langle a_1\rangle.(c_{k+9}?(z_x).\overline{s}_1!\langle z_x\rangle.\overline{c_{k+10}}!\langle\rangle \mid c_{k+10}?() \mid{} \\
&\qquad a_1?(y_1).y_1?(r_1, r_2).(\nu\, s')\,(z_x!\langle s'\rangle.\overline{s'}!\langle r_1, r_2, s_1\rangle))))))
\end{aligned}$$

with $g = \{X \mapsto r_1, r_2\}$

Figure 5.4: Breakdown of a recursive process (Example 5.2.2)

Consider now the process $P = \mu X.r?(w).r!\langle -w\rangle.X$. Let us write P' to denote the "body" of P, i.e., $P' = r?(w).r!\langle -w\rangle.X$. Then, process $\mathcal{F}(P)$ is

$$(\nu\, \widetilde{c})\,(\nu\, c^r)\,(c^r?(b).(\nu\, s)\,(b!\langle s\rangle.\overline{s}!\langle r_1, r_2\rangle) \mid \overline{c_1}!\langle\rangle \mid \mathcal{A}^1_\epsilon(P\{{}^{r_1}\!/r\})_\emptyset)$$

where $\widetilde{c} = (c_1, \ldots, c_{\lfloor P \rceil})$ and $\mathcal{A}^1_\epsilon(P\{{}^{r_1}\!/r\})_\emptyset$ is in Figure 5.4.

In Figure 5.4, $\mathcal{A}^1_\epsilon(P\{{}^{r_1}\!/r\})$ simulates recursion in P using replication. Given some index k, process R^k mimics actions of the recursive body. It first gets a decomposition of r by interacting with the process providing recursive names on c^r (for the first instance, this is a top-level process in $\mathcal{F}(P)$). Then, it mimics the first input action on the channel received for z_1 (that is, r_1): the input of actual names for w_1 is delegated through channel redirections to name y_2 (both prefixes are highlighted in Figure 5.4). Once the recursive name is used, the decomposition of recursive name is made available for the breakdown of the continuation by a communication on c^r. Similarly, in the continuation, the second action on r, output, is mimicked by r_2 (received for z_2), with the output of actual name w_1 delegated to $\overline{s}'$ (both prefixes are highlighted in Figure 5.4).

Subprocess R^5 is a breakdown of the first instance of the recursive body. The replication guarded by a_1 produces a next instance, i.e., process $R^2\{x_{r_1}, x_{r_2}/r_1, r_2\}$ in $\widehat{P}$. By communication on a_1 and a few reductions on propagators, it gets activated: along a_1 it first receives a name for y_1' along which it also receives: (i) recursive names r_1, r_2 for variables x_{r_1}, x_{r_2}, and (ii) a name for y_1 along which it will receive a_1 again, for future instances, as it can be seen in $\mathcal{A}^{k+7}_{\tilde{z_x}}(X)_g$.

5.2.4 Results

We establish the minimality result for π using the typability of $\mathcal{F}(\cdot)$. We need some auxiliary definitions to characterize the propagators required to decompose recursive processes.

Lemma 4.3.1 state typability results by introducing two typing environments, denoted Θ and Φ. While environment Θ is used to type linear propagators (e.g., $c_k, c_{k+1}, \ldots$) generated by the breakdown function $\mathcal{B}^{-}_{-}(\cdot)$, environment Φ types shared propagators used in trios that propagate breakdown of recursive names (e.g., $c^r, c^v, \ldots$ where r and v are recursive names).

Definition 5.2.8 (Session environment for propagators)**.** Let Θ be the session environment and Φ be the recursive propagator environment defined in Lemma 4.3.1. Then, by applying the encoding $\langle\!\langle \cdot \rangle\!\rangle^2$, we define Θ' and Φ' as follows: $\Theta' = \langle\!\langle \Theta \rangle\!\rangle^2$, $\Phi' = \langle\!\langle \Phi \rangle\!\rangle^2$.

We can use $\Theta' = \langle\!\langle \Theta \rangle\!\rangle^2$ in the following statement, where we state the typability result for the breakdown function. The proof composes previously known results:

Lemma 5.2.1 (Typability of Breakdown)**.** *Let P be an initialized π process. If $\Gamma; \Delta, \Delta_\mu \vdash P \triangleright \diamond$, then $\mathcal{H}(\Gamma'), \Phi'; \mathcal{H}(\Delta), \Theta' \vdash \mathcal{A}^k_\epsilon(P)_g \triangleright \diamond$, where:*

- $k > 0$;
- $\widetilde{r} = \mathtt{dom}(\Delta_\mu)$;
- $\Phi' = \prod_{r \in \widetilde{r}} c^r : \langle\langle ?(\mathcal{R}'^{\star}(\Delta_\mu(r)));\mathtt{end}\rangle\rangle$;
- $\mathsf{balanced}(\Theta')$ *with*

$$\mathtt{dom}(\Theta') = \{c_k, c_{k+1}, \ldots, c_{k+\lfloor P \rceil - 1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\lfloor P \rceil - 1}}\}$$

such that $\Theta'(c_k) = ?(\cdot);\mathtt{end}$.

Proof. Directly from Lemma 4.3.1 and from Theorems 5.1 and 5.2 from [41]. See Appendix B.2.1 for details. □

We now consider typability for the decomposition function, using $\Phi' = \langle\!\langle \Phi \rangle\!\rangle^2$ as in Definition 5.2.8.

Theorem 5.2.1 (Minimality Result for π)**.** *Let P be a closed π process, with $\widetilde{u} = \mathtt{fn}(P)$ and $\widetilde{v} = \mathtt{rn}(P)$. If $\Gamma; \Delta, \Delta_\mu \vdash P \triangleright \diamond$, where Δ_μ only involves recursive session types, then $\mathcal{H}(\Gamma\sigma); \mathcal{H}(\Delta\sigma), \mathcal{H}(\Delta_\mu\sigma) \vdash \mathcal{F}(P) \triangleright \diamond$, where $\sigma = \{\mathsf{init}(\widetilde{u})/\widetilde{u}\}$.*

Proof. Directly by using Lemma 5.2.1; see Appendix B.2.2 for details. □

While Theorem 5.2.1 gives a useful *static* guarantee about the correctness of our decomposition of P into $\mathcal{F}(P)$, a *dynamic* guarantee that confirms that P and $\mathcal{F}(P)$ are behaviorally equivalent is most relevant. Before establishing such a dynamic guarantee, we explore to what extent $\mathcal{F}(P)$ can be optimized, i.e., whether redundancies induced by the "decompose by composition" approach can be safely eliminated.

5.3 Optimizations

Although conceptually simple, the “decompose by composition” approach to our minimality result induces redundancies. Here we propose $\mathcal{F}^*(\,\cdot\,)$, an optimization of $\mathcal{F}(\,\cdot\,)$ with less synchronizations, and establish its correctness properties, in terms of its corresponding minimality result (static correctness, Theorem 5.3.1, Page 140) but also *dynamic correctness* (Theorem 5.3.2, Page 145).

5.3.1 Motivation

To motivate our insights, consider the process $\mathcal{A}^k_{\tilde{x}}(u_i?(w).Q)_g$ as presented in Section 5.2.1 and Table 5.1. We identify some suboptimal features of a decomposition based on $\mathcal{A}^k_{\tilde{x}}(\,\cdot\,)_g$:

Channel redirections While the given process receives a name for variable w along u_i, its breakdown does not input a breakdown of w directly, but does so through a series of channel redirections: u_i receives a name along which it sends the restricted name s, along which it sends the restricted name s_1 and so on. Finally, the name received for y' receives $\widetilde{w}$, the breakdown of w. This redundancy is perhaps more evident in Definition 5.2.5, which gives the translation of types by composition: the mimicked input action is five-level nested for the original name. This resulting type is due to the composition of $[\![\,\cdot\,]\!]^1_g$ and $[\![\,\cdot\,]\!]^2$.

Redundant synchronizations on propagators Also, $\mathcal{A}^k_{\tilde{x}}(u_i?(w).Q)_g$ features redundant communications on propagators. For example, the bound name y is locally propagated by c_{k+1} and c_{k+2}. This is the result of breaking down sequential prefixes induced by $[\![\,\cdot\,]\!]^1_g$ (not present in the original process).

Trio structure is lost Last but not least, the trio structure is lost as subprocess $\widehat{Q}_{\tilde{x}}$ is guarded and nested, and it inductively invokes the function on continuation Q. This results in an arbitrary level of process nesting, which is induced by the final application of encoding $[\![\,\cdot\,]\!]^2$ in the composition.

The shortcomings of $\mathcal{A}^k_{\tilde{x}}(\,\cdot\,)_g$ are also evident in the treatment of recursive processes and recursive names. Because HO does not feature recursion constructs, $[\![\,\cdot\,]\!]^1_g$ encodes them by relying on abstraction passing and shared abstractions. Then, going back to π via $[\![\,\cdot\,]\!]^2_g$, these representations are translated to a process involving a replicated subprocess. But going through this path the encoding of recursive process becomes convoluted. On top of that, all non-optimal features already discussed for the case of input are also present in the decomposition of recursion.

Based on these observations, here we develop an optimized decomposition function, denoted $\mathcal{F}^*(\,\cdot\,)$ (Definition 5.3.7), that avoids the redundancies described above. The optimized decomposition relies on a streamlined breakdown function, denoted $\mathcal{A}^k_{\tilde{x}}(\,\cdot\,)$, which produces a composition of trios processes, with a fixed maximum number of nested prefixes. The decomposed process avoids channel redirections and only introduces propagators that codify the sequentiality of the original process.

Roadmap To facilitate reading, we summarize notations and definitions related to $\mathcal{F}(\,\cdot\,)$ and their corresponding notions for the optimized decomposition $\mathcal{F}^*(\,\cdot\,)$—see Table 5.3.

5.3.2 Preliminaries

As before, we need to decompose a session type into a *list* of minimal session types:

	Section 5.2	This section
Degree of P	$\lfloor P \rceil$ (Definition 5.2.2)	$\lfloor P \rceil^*$ (Definition 5.3.3)
Decomposition of S	$\mathcal{H}(S)$ and $\mathcal{R}'(S)$ (Figure 5.3)	$\mathcal{H}^*(S)$ and $\mathcal{R}(S)$ (Figure 5.5)
Breakdown of P	$\mathcal{A}^k_{\tilde{x}}(P)_g$ (Tables 5.1 and 5.2)	$\mathcal{A}^k_{\tilde{x}}(P)$ and $\widehat{\mathcal{A}}^k_{\tilde{x}}(P)_g$ (Tables 5.4 and 5.5)
Decomposition of P	$\mathcal{F}(P)$ (Definition 5.2.7)	$\mathcal{F}^*(P)$ (Definition 5.3.7)

Table 5.3: Summary of notations used in our two decompositions.

Definition 5.3.1 (Decomposing Types)**.** Let S and C be a session and a channel type, resp. (cf. Figure 2.4). The *type decomposition function* $\mathcal{H}^*(\,\cdot\,)$ is defined in Figure 5.5.

As before, we need two auxiliary functions for decomposing recursive types, denoted $\mathcal{R}(\,\cdot\,)$ and $\mathcal{R}^\star(\,\cdot\,)$.

A comparison between $\mathcal{H}^*(\,\cdot\,)$ and $\mathcal{H}(\,\cdot\,)$ (Figure 5.3) is already useful to understand the intent and scope of the optimized decomposition. Consider, e.g., the decompositions of the session type $!\langle C\rangle;S$ (with $S \neq \texttt{end}$):

$$\begin{aligned}\mathcal{H}(!\langle C\rangle;S) &= !\langle\langle ?(?(\langle ?(\mathcal{H}(C));\texttt{end}\rangle);\texttt{end});\texttt{end}\rangle\rangle;\texttt{end}, \mathcal{H}(S)\\ \mathcal{H}^*(!\langle C\rangle;S) &= !\langle \mathcal{H}^*(C)\rangle;\texttt{end}\,, \mathcal{H}^*(S)\end{aligned}$$

These differences at the level of induced MSTs will be useful in our formally comparison of the two decompositions, in Section 5.3.5.

Example 5.3.1 (Decomposing a Recursive Type)**.** Let $S = \mu\mathsf{t}.S'$ be a recursive session type, with $S' = ?(\mathsf{Int});?(\mathsf{Bool});!\langle\mathsf{Bool}\rangle;\mathsf{t}$. By Figure 5.5, since S is tail-recursive, $\mathcal{H}^*(S) = \mathcal{R}(S')$. Further,

$$\mathcal{R}(S') = \mu\mathsf{t}.?(\mathcal{H}^*(\mathsf{Int}));\mathsf{t}, \mathcal{R}(?(\mathsf{Bool});!\langle\mathsf{Bool}\rangle;\mathsf{t})$$

By definition of $\mathcal{R}(\,\cdot\,)$, we obtain

$$\mathcal{H}^*(S) = \mu\mathsf{t}.?(\mathsf{Int});\mathsf{t}, \mu\mathsf{t}.?(\mathsf{Bool});\mathsf{t}, \mu\mathsf{t}.!\langle\mathsf{Bool}\rangle;\mathsf{t}, \mathcal{R}(t)$$

(using $\mathcal{H}^*(\mathsf{Int}) = \mathsf{Int}$ and $\mathcal{H}^*(\mathsf{Bool}) = \mathsf{Bool}$). Since $\mathcal{R}(\mathsf{t}) = \epsilon$, we have

$$\mathcal{H}^*(S) = \mu\mathsf{t}.?(\mathsf{Int});\mathsf{t}, \mu\mathsf{t}.?(\mathsf{Bool});\mathsf{t}, \mu\mathsf{t}.!\langle\mathsf{Bool}\rangle;\mathsf{t}$$

Example 5.3.2 (Decomposing an Unfolded Recursive Type)**.** Let $T = ?(\mathsf{Bool});!\langle\mathsf{Bool}\rangle;S$ be a derived unfolding of S from Example 5.3.1. Then, by Figure 5.5, $\mathcal{R}^\star(T)$ is the list of minimal recursive types obtained as follows: first, $\mathcal{R}^\star(T) = \mathcal{R}^\star(!\langle\mathsf{Bool}\rangle;\mu\mathsf{t}.S')$ and after one more step, $\mathcal{R}^\star(!\langle\mathsf{Bool}\rangle;\mu\mathsf{t}.S') = \mathcal{R}^\star(\mu\mathsf{t}.S')$. Finally, we have $\mathcal{R}^\star(\mu\mathsf{t}.S') = \mathcal{R}(S')$. We get the same list of minimal types as in Example 5.3.1:

$$\mathcal{R}^\star(T) = \mu\mathsf{t}.?(\mathsf{Int});\mathsf{t}, \mu\mathsf{t}.?(\mathsf{Bool});\mathsf{t}, \mu\mathsf{t}.!\langle\mathsf{Bool}\rangle;\mathsf{t}$$

Definition 5.3.2 (Decomposing Environments)**.** Given environments Γ and Δ, we define $\mathcal{H}^*(\Gamma)$ and $\mathcal{H}^*(\Delta)$ inductively as $\mathcal{H}^*(\emptyset) = \emptyset$ and

$$\begin{aligned}\mathcal{H}^*(\Delta, u_i : S) &= \mathcal{H}^*(\Delta), (u_i, \ldots, u_{i+|\mathcal{H}^*(S)|-1}) : \mathcal{H}^*(S)\\ \mathcal{H}^*(\Gamma, u_i : \langle S\rangle) &= \mathcal{H}^*(\Gamma), u_i : \mathcal{H}^*(\langle S\rangle)\end{aligned}$$

We now define the (optimized) degree of a process. A comparison with the previous definition (Definition 5.2.2) provides further indication of the improvements induced by optimized decomposition.

$$
\begin{aligned}
\mathcal{H}^*(!\langle C\rangle;S) &= \begin{cases} !\langle \mathcal{H}^*(C)\rangle;\mathtt{end} & \text{if } S = \mathtt{end} \\ !\langle \mathcal{H}^*(C)\rangle;\mathtt{end}\,, \mathcal{H}^*(S) & \text{otherwise} \end{cases} \\
\mathcal{H}^*(?(C);S) &= \begin{cases} ?(\mathcal{H}^*(C));\mathtt{end} & \text{if } S = \mathtt{end} \\ ?(\mathcal{H}^*(C));\mathtt{end}\,, \mathcal{H}^*(S) & \text{otherwise} \end{cases} \\
\mathcal{H}^*(\mathtt{end}) &= \mathtt{end} \\
\mathcal{H}^*(\langle S\rangle) &= \langle \mathcal{H}^*(S)\rangle \\
\mathcal{H}^*(S_1, \ldots, S_n) &= \mathcal{H}^*(S_1), \ldots, \mathcal{H}^*(S_n) \\
\mathcal{H}^*(\mu \mathtt{t}.S') &= \mathcal{R}(S') \\
\mathcal{H}^*(S) &= \mathcal{R}^\star(S) \quad \text{where } S \neq \mu\mathtt{t}.S' \\
\mathcal{R}(\mathtt{t}) &= \epsilon \\
\mathcal{R}(!\langle C\rangle;S) &= \mu\mathtt{t}.!\langle \mathcal{H}^*(C)\rangle;\mathtt{t}, \mathcal{R}(S) \\
\mathcal{R}(?(C);S) &= \mu\mathtt{t}.?(\mathcal{H}^*(C));\mathtt{t}, \mathcal{R}(S) \\
\mathcal{R}^\star(?(C);S) &= \mathcal{R}^\star(!\langle C\rangle;S) = \mathcal{R}^\star(S) \\
\mathcal{R}^\star(\mu\mathtt{t}.S) &= \mathcal{R}(S)
\end{aligned}
$$

Figure 5.5: Optimized decomposition of session types $\mathcal{H}^*(\,\cdot\,)$ (cf. Definition 5.3.1)

Definition 5.3.3 (Degree of a Process)**.** The *optimized degree* of a process P, denoted $\lfloor P\rceil^*$, is inductively defined as follows:

$$
\begin{cases}
\lfloor Q\rceil^* + 1 & \text{if } P = u_i!\langle y\rangle.Q \text{ or } P = u_i?(y).Q \\
\lfloor Q\rceil^* & \text{if } P = (\nu\, s : S)\, Q \\
\lfloor Q\rceil^* + 1 & \text{if } P = (\nu\, r : S)\, Q \text{ and } \mathsf{tr}(S) \\
\lfloor Q\rceil^* + \lfloor R\rceil^* + 1 & \text{if } P = Q \mid R \\
1 & \text{if } P = \mathbf{0} \text{ or } P = X \\
\lfloor Q\rceil^* + 1 & \text{if } P = \mu X.Q
\end{cases}
$$

As before, given a finite tuple of names $\widetilde{u} = (a, b, s, s', \ldots)$, we write $\mathsf{init}(\widetilde{u})$ to denote the tuple $(a_1, b_1, s_1, s'_1, \ldots)$; recall that a process is initialized if all of its names are indexed.

Given two tuples of indexed names $\widetilde{u}$ and $\widetilde{x}$, it is useful to collect those names in $\widetilde{x}$ that appear in $\widetilde{u}$.

Definition 5.3.4 (Free indexed names)**.** Let $\widetilde{u}$ and $\widetilde{x}$ be two tuples of names. We define the set $\mathtt{fnb}(\widetilde{u}, \widetilde{x})$ as $\{z_k : z_i \in \widetilde{u} \wedge z_k \in \widetilde{x}\}$.

As usual, we treat sets of names as tuples (and vice-versa). By abusing notation, given a process P, we shall write $\mathtt{fnb}(P, \widetilde{y})$ to stand for $\mathtt{fnb}(\mathtt{fn}(P), \widetilde{y})$. Then, we have that $\mathtt{fnb}(P, \widetilde{x}) \subseteq \widetilde{x}$. In the definition of the breakdown function, this notion allows us to conveniently determine a *context* for a subsequent trio.

Definition 5.3.5. Given a process P, we write $\mathsf{frv}(P)$ to denote that P has a free recursive variable.

Remark 5. Whenever $c_k?(\widetilde{y})$ (resp. $\overline{c_k}!\langle\widetilde{y}\rangle$) with $\widetilde{y} = \epsilon$, we shall write $c_k?()$ (resp. $\overline{c_k}!\langle\rangle$) to stand for $c_k?(y)$ (resp. $\overline{c_k}!\langle y\rangle$) such that $c_k : ?(\langle \mathtt{end}\rangle);\mathtt{end}$ (resp. $\overline{c_k} : !\langle\langle \mathtt{end}\rangle\rangle;\mathtt{end}$).

	P	$\mathcal{A}^k_{\tilde{x}}(P)$
1	$u_i?(y).Q$	$c_k?(\widetilde{x}).u_l?(\widetilde{y}).\overline{c_{k+1}}!\langle\widetilde{z}\rangle \mid \mathcal{A}^{k+1}_{\tilde{z}}(Q\sigma)$ $y_j : S \quad \widetilde{y} = (y_1, \ldots, y_{\lvert\mathcal{H}^*(S)\rvert})$ $\widetilde{w} = (\mathtt{lin}(u_i))\,?\,\{u_i\}\colon \epsilon \quad \widetilde{z} = \mathtt{fnb}(Q, \widetilde{x}\widetilde{y} \setminus \widetilde{w})$ $l = (\mathtt{tr}(u_i))\,?\,\iota(S)\colon i \quad \sigma = \mathsf{next}(u_i) \cdot \{{}^{y_1}/{}_{y}\}$
2	$u_i!\langle y_j\rangle.Q$	$c_k?(\widetilde{x}).u_l!\langle\widetilde{y}\rangle.\overline{c_{k+1}}!\langle\widetilde{z}\rangle \mid \mathcal{A}^{k+1}_{\tilde{z}}(Q\sigma)$ $y_j : S \quad \widetilde{y} = (y_j, \ldots, y_{j+\lvert\mathcal{H}^*(S)\rvert-1})$ $\widetilde{w} = (\mathtt{lin}(u_i))\,?\,\{u_i\}\colon \epsilon \quad \widetilde{z} = \mathtt{fnb}(Q, \widetilde{x} \setminus \widetilde{w})$ $l = (\mathtt{tr}(u_i))\,?\,\iota(S))\colon i \quad \sigma = \mathsf{next}(u_i)$
3	$(\nu\, s : C)\, Q$	$(\nu\, \widetilde{s} : \mathcal{H}^*(C))\ \mathcal{A}^k_{\tilde{x}}(Q\sigma)$ $\widetilde{s} = (s_1, \ldots, s_{\lvert\mathcal{H}^*(C)\rvert}) \quad \sigma = \{{}^{s_1\overline{s_1}}/{}_{s\overline{s}}\}$
4	$(\nu\, s : \mu\mathsf{t}.S)\, Q$	$(\nu\, \widetilde{s} : \mathcal{R}(S))\ (c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle\widetilde{z}\rangle.\mathbf{0} \mid \mathcal{A}^{k+1}_{\tilde{z}}(Q))$ $\mathsf{tr}(\mu\mathsf{t}.S) \quad \widetilde{s} = (s_1, \ldots, s_{\lvert\mathcal{R}(S)\rvert})$ $\widetilde{z} = \widetilde{x}, \widetilde{s}, \widetilde{\overline{s}} \quad \widetilde{\overline{s}} = (\overline{s_1}, \ldots, \overline{s_{\lvert\mathcal{R}(S)\rvert}})$
5	$Q_1 \mid Q_2$	$c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle\widetilde{y}\rangle.\overline{c_{k+l+1}}!\langle\widetilde{z}\rangle \mid \mathcal{A}^{k+1}_{\tilde{y}}(Q_1) \mid \mathcal{A}^{k+l+1}_{\tilde{z}}(Q_2)$ $l = \lvert Q_1\rvert \quad \widetilde{y} = \mathtt{fnb}(Q_1, \widetilde{x}) \quad \widetilde{z} = \mathtt{fnb}(Q_2, \widetilde{x})$
6	$\mathbf{0}$	$c_k?().\mathbf{0}$
7	$\mu X.P$	$(\nu\, c^r_X)\ (c_k?(\widetilde{x}).\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.\mu X.c^r_X?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{y}\rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\tilde{z}}(P)_g)$ $\widetilde{n} = \mathtt{fs}(P) \quad \lvert\widetilde{z}\rvert = \lvert\widetilde{y}\rvert \quad g = \{X \mapsto \widetilde{z}\}$ $\widetilde{n} : \widetilde{C} \quad \widetilde{z} = \mathsf{bn}(\widetilde{n} : \widetilde{C})$

Table 5.4: Optimized breakdown function $\mathcal{A}^k_{\tilde{x}}(\,\cdot\,)$. The auxiliary function $\widehat{\mathcal{A}}^k_{\tilde{x}}(\,\cdot\,)_g$ is given in Table 5.5.

Definition 5.3.6 (Index function)**.** Let S be an (unfolded) recursive session type. The function $\iota(S)$ is defined as follows:

$$\iota(S) = \begin{cases} \widehat{\iota}_0(S'\{{}^{S}/{}_{\mathsf{t}}\}) & \text{if } S = \mu\mathsf{t}.S' \\ \widehat{\iota}_0(S) & \text{otherwise} \end{cases} \qquad \widehat{\iota}_l(T) = \begin{cases} \widehat{\iota}_{l+1}(S) & \text{if } T = !\langle U\rangle;S \text{ or } T = ?(U);S \\ \lvert\mathcal{R}(S)\rvert - l + 1 & \text{if } T = \mu\mathsf{t}.S \end{cases}$$

5.3.3 The Optimized Decomposition

We define the optimized decomposition $\mathcal{F}^*(\,\cdot\,)$ by relying on the revised breakdown function $\mathcal{A}^k_{\tilde{x}}(\,\cdot\,)$ (cf. Section 5.3.3). Given a context $\widetilde{x}$ and a $k > 0$, $\mathcal{A}^k_{\tilde{x}}(\,\cdot\,)$ is defined on initialized processes. Table 5.4 gives the definition: we use an auxiliary function for recursive processes, denoted $\widehat{\mathcal{A}}^k_{\tilde{x}}(\,\cdot\,)_g$ (cf. Section 5.3.3), where parameter g maps recursive variables to a list of name variables.

In the following, to keep presentation simple, we assume recursive processes $\mu X.P$ in which P does not contain a subprocess of shape $\mu Y.P'$. The generalization of our decomposition without this assumption is not difficult, but is notationally heavy.

	P	$\widehat{\mathcal{A}}^{k}_{\widetilde{x}}(P)_g$	
8	$u_i!\langle y_j\rangle.Q$	$\mu X.c^r_k?(\widetilde{x}).u_l!\langle\widetilde{y}\rangle.\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X \mid$ $\widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(Q\sigma)_g$ (if $g \neq \emptyset$) $\mu X.c^r_k?(\widetilde{x}).(u_l!\langle\widetilde{y}\rangle.c^r_{k+1}!\langle\widetilde{z}\rangle \mid X) \mid$ $\widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(Q\sigma)_g$ (if $g = \emptyset$)	$y_j : T$ $\widetilde{y} = (y_j, \ldots, y_{j+\lvert\mathcal{H}^*(T)\rvert-1})$ $\widetilde{w} = (\mathtt{lin}(u_i))\,?\,\{u_i\}\!: \epsilon$ $\widetilde{z} = g(X) \cup \mathtt{fnb}(Q, \widetilde{x} \setminus \widetilde{w})$ $l = (\mathtt{tr}(u_i))\,?\,\iota(S)\!: i$ $\sigma = \mathsf{next}(u_i)$
9	$u_i?(y).Q$	$\mu X.c^r_k?(\widetilde{x}).u_l?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X \mid$ $\widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(Q\sigma)_g$ (if $g \neq \emptyset$) $\mu X.c^r_k?(\widetilde{x}).(u_l?(\widetilde{y}).c^r_{k+1}!\langle\widetilde{z}\rangle \mid X) \mid$ $\widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(Q\sigma)_g$ (if $g = \emptyset$)	$y_j : T$ $\widetilde{y} = (y_1, \ldots, y_{\lvert\mathcal{H}^*(T)\rvert})$ $\widetilde{w} = (\mathtt{lin}(u_i))\,?\,\{u_i\}\!: \epsilon$ $\widetilde{z} = g(X) \cup \mathtt{fnb}(Q, \widetilde{x}\widetilde{y} \setminus \widetilde{w})$ $l = (\mathtt{tr}(u_i))\,?\,\iota(S)\!: i$ $\sigma = \mathsf{next}(u_i) \cdot \{y_1/y\}$
10	$Q_1 \mid Q_2$	$\mu X.c^r_k?(\widetilde{x}).$ $(\overline{c^r_{k+1}}!\langle\widetilde{y}_1\rangle.X \mid c^r_{k+l+1}!\langle\widetilde{y}_2\rangle) \mid$ (if $g \neq \emptyset$) $\widehat{\mathcal{A}}^{k+1}_{\widetilde{y}_1}(Q_1)_g \mid \widehat{\mathcal{A}}^{k+l+1}_{\widetilde{y}_2}(Q_2)_\emptyset$ $\mu X.c^r_k?(\widetilde{x}).$ $(c^r_{k+1}!\langle\widetilde{y}_1\rangle \mid c^r_{k+l+1}!\langle\widetilde{y}_2\rangle \mid X) \mid$ $\widehat{\mathcal{A}}^{k+1}_{\widetilde{y}_1}(Q_1)_\emptyset \mid \widehat{\mathcal{A}}^{k+l+1}_{\widetilde{y}_2}(Q_2)_\emptyset$ (if $g = \emptyset$)	$\mathsf{frv}(Q_1)$ $\widetilde{y}_1 = g(X) \cup \mathtt{fnb}(Q_1, \widetilde{x})$ $\widetilde{y}_2 = \mathtt{fnb}(Q_2, \widetilde{x})$ $l = \lvert Q_1\rvert$
11	$(\nu\, s)\, Q$	$\mu X.(\nu\, \widetilde{s} : \mathcal{H}^*(C))$ $(c^r_k?(\widetilde{x}).\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X \mid$ $\widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(Q\sigma)_g)$ (if $g \neq \emptyset$) $\mu X.(\nu\, \widetilde{s} : \mathcal{H}^*(C))$ $(c^r_k?(\widetilde{x}).(c^r_{k+1}!\langle\widetilde{z}\rangle \mid X) \mid$ $\widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(Q\sigma)_g)$ (if $g = \emptyset$)	$s : C$ $\widetilde{s} = (s_1, \ldots, s_{\lvert\mathcal{H}^*(C)\rvert})$ $\widetilde{\overline{s}}' = (\overline{s_1}, \ldots, \overline{s_{\lvert\mathcal{H}^*(C)\rvert}})$ $\widetilde{\overline{s}} = (\mathtt{lin}(C))\,?\,\widetilde{\overline{s}}'\!: \epsilon$ $\widetilde{z} = \widetilde{x}, \widetilde{s}, \widetilde{\overline{s}}$ $\sigma = \{s_1\overline{s_1}/s\overline{s}\}$
12	X	$\mu X.c^r_k?(\widetilde{x}).c^r_X!\langle\widetilde{x}\rangle X$ (if $g \neq \emptyset$) $\mu X.c^r_k?().(c^r_X!\langle\rangle \mid X)$ (if $g = \emptyset$)	
13	$\mathbf{0}$	$c^r_k?().\mathbf{0}$	

Table 5.5: The auxiliary function $\widehat{\mathcal{A}}^{k}_{\widetilde{x}}(\,\cdot\,)_g$.

The Optimized Breakdown Function

We describe entries 1-7 in Table 5.4, assuming the side conditions given in the table.

1. **Input** Process $\mathcal{A}^{k}_{\widetilde{x}}(u_i?(y).Q)$ consists of a leading trio that mimics the input and runs in parallel with the breakdown of Q. In the trio, a context $\widetilde{x}$ is expected along c_k. Then, an input on u_l mimics the input action: it expects the *decomposition* of name y, denoted $\widetilde{y}$. To decompose y we use its type: if $y : S$ then $\widetilde{y} = (y_1, \ldots, y_{\lvert\mathcal{H}^*(S)\rvert})$. The index l of u_l depends on the type of u_i. Intuitively, if u_i is tail-recursive then $l = \iota(S)$ (Definition 5.3.6) and we do not increment it, as the same decomposition of u_i should be used to mimic a new instance in the continuation. Otherwise, if u_i is linear then we use the substitution $\sigma = \{u_{i+1}/u_i\}$ to increment the index in Q. Next, the context $\widetilde{z} = \mathtt{fnb}(Q, \widetilde{x}\widetilde{y} \setminus \widetilde{w})$ is propagated, where $\widetilde{w} = (u_i)$ or $\widetilde{w} = \epsilon$.

2. **Output** Process $\mathcal{A}^k_{\widetilde{x}}(u_i!\langle y_j\rangle.Q)$ sends the decomposition of y on u_l, with l as in the input case. We decompose name y_j based on its type S: $\widetilde{y} = (y_j, \ldots, y_{j+|\mathcal{H}^*(S)|-1})$. The context to be propagated is $\widetilde{z} = \texttt{fnb}(P, \widetilde{x} \setminus \widetilde{w})$, where $\widetilde{w}$ and σ are as in the input case.

3. **Restriction (Non-recursive name)** The breakdown of process $(\nu\, s : C)\, Q$ is

$$(\nu\, \widetilde{s} : \mathcal{H}^*(C))\ \mathcal{A}^k_{\widetilde{x}}(Q\sigma)$$

   where s is decomposed using C: $\widetilde{s} = (s_1, \ldots, s_{|\mathcal{H}^*(C)|})$. Since $(\nu\, s)$ binds s and $\overline{s}$ (or only s if C is a shared type) the substitution σ is simply $\{s_1\overline{s}_1/s\overline{s}\}$ and initializes indexes in Q.

4. **Restriction (Recursive name)** As in the previous case, in the breakdown of $(\nu\, s : \mu \mathrm{t}.S)\, Q$ the name s is decomposed into $\widetilde{s}$ by relying on $\mu\mathrm{t}.S$. Here the breakdown consists of the breakdown of Q running in parallel with a control trio, which appends restricted (recursive) names $\widetilde{s}$ and $\widetilde{\overline{s}}$ to the context, i.e., $\widetilde{z} = \widetilde{x}, \widetilde{s}, \widetilde{\overline{s}}$.

5. **Composition** The breakdown of process $Q_1 \mid Q_2$ uses a control trio to trigger the breakdowns of Q_1 and Q_2, similarly as before.

6. **Inaction** The breakdown of $\mathbf{0}$ is an input prefix that receives an empty context ($\widetilde{x} = \epsilon$).

7. **Recursion** The breakdown of $\mu X.P$ is as follows:

$$(\nu\, c^r_X)\, (c_k?(\widetilde{x}).\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.\mu X.c^r_X?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{y}\rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(P)_g)$$

   We have a control trio and the breakdown of P, obtained using the auxiliary function $\widehat{\mathcal{A}}^k_{\widetilde{x}}(\,\cdot\,)_g$ (Table 5.5). The trio receives the context $\widetilde{x}$ on c_k and propagates it further. To ensure typability, we bind all session free names of P using the context $\widetilde{z}$, which contains the decomposition of those free names. This context is needed to break down P, and so we record it as $g = \{X \mapsto \widetilde{z}\}$ in the definition of $\widehat{\mathcal{A}}^{k+1}_{\widetilde{x}}(P)_g$. This way, $\widetilde{z}$ will be propagated all the way until reaching X.

   Next, the recursive trio is enabled, and receives $\widetilde{y}$ along c^r_X, with $|\widetilde{z}| = |\widetilde{y}|$ and $l = |P|$. The tuple $\widetilde{y}$ is propagated to the first trio of $\widehat{\mathcal{A}}^{k+1}_{\widetilde{x}}(P)_g$. By definition of $\widehat{\mathcal{A}}^{k+1}_{\widetilde{x}}(P)_g$, its propagator c^r_X will send the same context as received by the first trio. Hence, the recursive part of the control trio keeps sending this context to the next instances of recursive trios of $\widehat{\mathcal{A}}^{k+1}_{\widetilde{x}}(P)_g$.

   Notice that the leading trio actually has four prefixes. This simplifies our presentation: this trio can be broken down into two trios by introducing an extra propagator c_{k+1} to send over c^r_{k+2}.

Handling P in $\mu X.P$

As already mentioned, we use the auxiliary function $\widehat{\mathcal{A}}^k_{\widetilde{x}}(\,\cdot\,)_g$ in Table 5.5 to generate *recursive trios*. We concentrate on discussing entries 8-11 in Table 5.5; the other entries are similar as before. A key observation is that parameter g can be empty. To see this, consider a process like $P = \mu X.(Q_1 \mid Q_2)$ where X occurs free in Q_1 but not in Q_2. If X occurs free in Q_1 then its decomposition will have a non-empty g, whereas the g for Q_2 will be empty. In the recursive trios of Table 5.5, the difference between $g \neq \emptyset$ and $g = \emptyset$ is subtle: in the former case, X appears guarded by a propagator; in the latter case, it appears unguarded in a parallel composition. This way, trios in the breakdown of Q_2 replicate themselves on a trigger from the breakdown of Q_1.

Having discussed this difference, we only describe the cases when $g \neq \emptyset$:

8 / 9. Output and Input The breakdown of $u_i!\langle y_j\rangle.Q$ consists of the breakdown of Q in parallel with a leading trio, a recursive process whose body is defined as in $\mathcal{B}(\cdot)$. As names $g(X)$ may not appear free in Q, we must ensure that a context $\widetilde{z}$ for the recursive body is propagated. The breakdown of $r?(y).Q$ is defined similarly.

10. Parallel Composition We discuss the breakdown of $Q_1 \mid Q_2$ assuming $\mathsf{frv}(Q_1)$. We take $\widetilde{y}_1 = g(X) \cup \mathtt{fnb}(Q_1, \widetilde{x})$ to ensure that $g(X)$ is propagated to the breakdown of X. The role of c^r_{k+l+1} is to enact a new instance of the breakdown of Q_2; it has a shared type to enable replication. In a running process, the number of these triggers in parallel denotes the number of available instances of Q_2.

11. Recursive Variable In this case, the breakdown is a control trio that receives the context $\widetilde{x}$ from a preceding trio and propagates it again to the first control trio of the breakdown of a recursive process along c^r_X. Notice that by construction we have $\widetilde{x} = g(X)$.

We may now define the optimized process decomposition $\mathcal{F}^*(\,\cdot\,)$:

Definition 5.3.7 (Decomposing Processes, Optimized)**.** Let P be a process with $\widetilde{u} = \mathtt{fn}(P)$ and $\widetilde{v} = \mathtt{rn}(P)$. Given the breakdown function $\mathcal{A}^k_{\widetilde{x}}(\cdot)$ in Table 5.4, the *optimized decomposition* of P, denoted $\mathcal{F}^*(P)$, is defined as

$$\mathcal{F}^*(P) = (\nu\, \widetilde{c})\,(\overline{c_k}!\langle\widetilde{r}\rangle.\mathbf{0} \mid \mathcal{A}^k_{\widetilde{r}}(P\sigma))$$

where: $k > 0$; $\widetilde{c} = (c_k, \ldots, c_{k+\lfloor P\rceil^*-1})$; $\widetilde{r}$ such that for $v \in \widetilde{v}$ and $v : S$ $(v_1, \ldots, v_{|\mathcal{R}(S)|}) \subseteq \widetilde{r}$; and $\sigma = \{\mathsf{init}(\widetilde{u})/\widetilde{u}\}$.

The definition of $\mathcal{F}^*(\,\cdot\,)$ is similar to the definition of $\mathcal{F}(\,\cdot\,)$ given in Definition 5.2.7. The optimizations are internal to the definition of the breakdown function; notice that the handling of recursive names needed in $\mathcal{F}(\,\cdot\,)$ is not needed in $\mathcal{F}^*(\,\cdot\,)$, as they are now handled by the auxiliary function $\widehat{\mathcal{A}}^k_{\widetilde{x}}(\,\cdot\,)_g$.

5.3.4 Examples

We now illustrate $\mathcal{F}^*(\,\cdot\,)$, $\mathcal{H}^*(\,\cdot\,)$, $\mathcal{A}^k_{\widetilde{x}}(\,\cdot\,)$, and $\widehat{\mathcal{A}}^k_{\widetilde{x}}(\,\cdot\,)_g$ by revisiting examples from Section 5.2.

Example 5.3.3 (Example 5.2.1, Revisited)**.** Consider again the process $P = (\nu\, u)\,(A \mid B)$ as in Example 5.2.1. Recall that P implements session types $S = !\langle T\rangle;\mathtt{end}$ and $T = ?(\mathsf{Int});!\langle\mathsf{Bool}\rangle;\mathtt{end}$.

By Definition 5.3.3, $\lfloor P\rceil^* = 9$. The optimized decomposition of P is:

$$\mathcal{F}^*(P) = (\nu\, \widetilde{c})\,(\overline{c_1}!\langle\rangle \mid (\nu\, u_1)\,\mathcal{A}^1_\epsilon((A \mid B)\sigma'))$$

where $\sigma' = \mathsf{init}(\mathtt{fn}(P)) \cdot \{u_1\overline{u_1}/u\overline{u}\}$ and $\widetilde{c} = (c_1, \ldots, c_9)$. We have:

$$\mathcal{A}^1_\epsilon((A \mid B)\sigma')) = c_1?().\overline{c_2}!\langle\rangle.\overline{c_6}!\langle\rangle \mid \mathcal{A}^2_\epsilon(A\sigma') \mid \mathcal{A}^6_\epsilon(B\sigma')$$

The breakdowns of sub-processes A and B are as follows:

$$\begin{aligned}
\mathcal{A}^2_\epsilon(A\sigma') = {} & c_2?().u_1!\langle w_1, w_2\rangle.\overline{c_3}!\langle\rangle \mid c_3?().\overline{w}_1?(t).\overline{c_4}!\langle\rangle \mid \\
& c_4?().\overline{w}_2!\langle\mathsf{odd}(t)\rangle.\overline{c_5}!\langle\rangle \mid c_5?().\mathbf{0} \\
\mathcal{A}^6_\epsilon(B\sigma') = {} & c_6?().\overline{u}_1?(x_1, x_2).\overline{c_7}!\langle x_1, x_2\rangle \mid c_7?(x_1, x_2).x_1!\langle 5\rangle.\overline{c_8}!\langle x_2\rangle \mid \\
& c_8?(x_2).x_2?(b_1).\overline{c_9}!\langle\rangle \mid c_9?().\mathbf{0}
\end{aligned}$$

Name w is decomposed as indexed names $\overline{w}_1, \overline{w}_2$; by using $\mathcal{H}^*(\cdot)$ (Definition 5.3.1) on T, their MSTs are $M_1 = !\langle \mathsf{Int} \rangle;\mathtt{end}$ and $M_2 = ?(\mathsf{Bool});\mathtt{end}$, respectively. Name u_1 is the decomposition of name u and it is typed with $!\langle \overline{M}_1, \overline{M}_2 \rangle;\mathtt{end}$.

We discuss the reduction steps from $\mathcal{F}^*(P)$. After a few administrative reductions on c_1, c_2, and c_6, $\mathcal{F}^*(P)$ mimics the first source communication:

$$\begin{aligned}\mathcal{F}^*(P) \longrightarrow^3 (\nu\, \widetilde{c}_*)\, (\, & u_1!\langle w_1, w_2 \rangle.\, \overline{c_3}!\langle\rangle \mid \mathbf{A}^3_\epsilon(\overline{w}?(t).\overline{w}!\langle \mathsf{odd}(t) \rangle.\mathbf{0}) \mid \\ & \overline{u}_1?(x_1, x_2).\, \overline{c_7}!\langle x_1, x_2 \rangle \mid \mathbf{A}^7_{x_1,x_2}(x!\langle 5 \rangle.x?(b).\mathbf{0})) \\ \longrightarrow (\nu\, \widetilde{c}_*)\, (\overline{c_3}!\langle\rangle & \mid \mathbf{A}^2_\epsilon(\overline{w}?(t).\overline{w}!\langle \mathsf{odd}(t) \rangle.\mathbf{0}) \mid \overline{c_7}!\langle w_1, w_2 \rangle \mid \mathbf{A}^7_{x_1,x_2}(x!\langle 5 \rangle.x?(b).\mathbf{0}))\end{aligned}$$

Above, $\widetilde{c}_* = (c_3, c_4, c_5, c_7, c_8, c_9)$. After reductions on c_3 and c_7, name w_1 substitutes x_1 and the communication along w_1 can be mimicked:

$$\begin{aligned}\mathcal{F}^*(P) \longrightarrow^6 (\nu\, \widetilde{c}_{**})\, (\, & \overline{w}_1?(t).\, \overline{c_4}!\langle\rangle \mid c_4?().\overline{w}_2!\langle \mathsf{odd}(t) \rangle.\overline{c_5}!\langle\rangle \mid c_5?().\mathbf{0} \mid \\ & w_1!\langle 5 \rangle.\, \overline{c_8}!\langle w_2 \rangle \mid c_8?(x_2).x_2?(b_1).\overline{c_9}!\langle\rangle \mid c_9?().\mathbf{0}) \\ \longrightarrow (\nu\, \widetilde{c}_{**})\, (\overline{c_4}!\langle 5 \rangle & \mid c_4?(t).\overline{w}_2!\langle \mathsf{odd}(t) \rangle.\overline{c_5}!\langle\rangle \mid c_5?().\mathbf{0} \mid \\ & \overline{c_8}!\langle w_2 \rangle \mid c_8?(x_2).x_2?(b_1).\overline{c_9}!\langle\rangle \mid c_9?().\mathbf{0})\end{aligned}$$

Above, $\widetilde{c}_{**} = (c_4, c_5, c_8, c_9)$. Further reductions follow similarly.

Example 5.3.4 (Example 5.2.2, Revisited)**.** Consider again the tail-recursive session type $S = \mu\mathtt{t}.?(\mathtt{Int});!\langle \mathtt{Int} \rangle;\mathtt{t}$. Also, let R be a process implementing a channel r with type S:

$$R = \mu X.r?(z).r!\langle -z \rangle.X$$

We decompose name r using S and obtain two channels typed with MSTs as in Figure 5.5. We have: $r_1 : \mu\mathtt{t}.?(\mathtt{Int});\mathtt{t}$ and $r_2 : \mu\mathtt{t}.!\langle \mathtt{Int} \rangle;\mathtt{t}$.

The trios produced by $\mathcal{A}^k_\epsilon(R)$ satisfy two properties: they (1) mimic the recursive behavior of R and (2) use the same decomposition of channel r (i.e., r_1,r_2) in every instance.

To accomplish (1), each trio of the breakdown of the recursion body is a recursive trio. For (2), we need two things. First, we expect to receive all recursive names in the context $\widetilde{x}$ when entering the decomposition of the recursion body; further, each trio should use one recursive name from the names received and propagate *all* of them to subsequent trio. Second, we need an extra control trio when breaking down prefix μX: this trio (i) receives recursive names from the last trio in the breakdown of the recursion body and (ii) activates another instance with these recursive names.

Using these ideas, process $\mathcal{A}^1_{r_1,r_2}(R)$ is as follows (we write R' to stand for $r?(z).r!\langle -z \rangle.X$):

$$c_1?(r_1, r_2).\overline{c^r_2}!\langle r_1, r_2 \rangle.\mu X.c^r_X?(y_1, y_2).\overline{c^r_2}!\langle y_1, y_2 \rangle.X \mid \widehat{\mathcal{A}}^2_{r_1,r_2}(R')$$

where $\widehat{\mathcal{A}}^2_{r_1,r_2}(R')$ is the composition of three recursive trios:

$$\begin{aligned}&\mu X.c^r_2?(y_1, y_2).r_1?(z_1).\overline{c^r_3}!\langle y_1, y_2, z_1 \rangle.X \mid \\ &\quad \mu X.c^r_3?(y_1, y_2, z_1).r_2?(-z_1).\overline{c^r_4}!\langle y_1, y_2 \rangle.X \mid \mu X.c^r_4?(y_1, y_2).\overline{c^r_X}!\langle y_1, y_2 \rangle.X\end{aligned}$$

c^r_2 will first activate the recursive trios with context (r_1, r_2). Next, each trio uses one of r_1, r_2 and propagate them both mimicking the recursion body. The last recursive trio sends r_1, r_2 back to the top-level control trio, so it can enact another instance of the decomposition of the recursion body by activating the first recursive trio.

5.3.5 Measuring the Optimization

Before discussing the static and dynamic correctness $\mathcal{F}^*(\cdot)$, here we measure the improvements over $\mathcal{F}(\cdot)$. A key metric for comparison is the number of prefixes/sychronizations induced by each decomposition. This includes (1) the number of prefixes involved in *channel redirections* and (2) the *number of propagators*; both can be counted by already defined notions:

1. Channel redirections can be counted by the levels of nesting in the decompositions of types (cf. Figures 5.3 and 5.5).
2. The number of propagators is determined by the degree of a process (cf. Definitions 5.2.2 and 5.3.3).

These two metrics are related; let us discuss them in detail.

Channel redirections The decompositions of types for $\mathcal{F}(\cdot)$ and $\mathcal{F}^*(\cdot)$ abstractly describe the respective channel redirections. The type decomposition for $\mathcal{F}(\cdot)$ (Figure 5.3) defines 5 levels of nesting for the translation of input/output types. Then, at the level of (decomposed) processes, channels with these types implement redirections: the nesting levels correspond to 5 additional prefixes in the decomposed process that mimic a source input/output action. In contrast, the type decomposition for $\mathcal{F}^*(\cdot)$ (Figure 5.5) induces no nesting, and so at the level of processes there are no additional prefixes.

Number of propagators We define auxiliary functions to count the number of propagators induced by $\mathcal{F}(\cdot)$ and $\mathcal{F}^*(\cdot)$. These functions, denoted $\#(\cdot)$ and $\#^*(\cdot)$, respectively, are defined using the degree functions ($\lfloor \cdot \rceil$ and $\lfloor \cdot \rceil^*$) given by Definitions 5.2.2 and 5.3.3, respectively.

Remarkably, $\lfloor \cdot \rceil$ and $\#(\cdot)$ are not equal. The difference lies in the number of tail-recursive names in a process. In $\mathcal{F}(\cdot)$ there are propagators c_k but also propagators c^r for recursive names. Definition 5.2.2, however, only counts the former kind of propagators. For any P, the number of propagators c^r in $\mathcal{F}(P)$ is the number of free and bound tail-recursive names in P. We remark that, by definition, there may be more than one occurrence of a propagator c^r in $\mathcal{F}(P)$: there is at least one prefix with subject c^r; further occurrences depend on the sequentiality structure of the (recursive) type assigned to r. On the other hand, in $\mathcal{F}^*(P)$ there are propagators c_k and propagators c^r_X, whose number corresponds to the number of recursive variables in the process. To define $\#(\cdot)$ and $\#^*(\cdot)$, we write $\mathsf{brn}(P)$ to denote bound occurrences of recursive names and $\#_X(P)$ to denote the number of occurrences of recursive variables.

Definition 5.3.8 (Propagators in $\mathcal{F}(P)$ and $\mathcal{F}^*(P)$)**.** Given a process P, the number of propagators in each decomposition is given by

$$\#(P) = \lfloor P \rceil + 2 \cdot |\mathsf{brn}(P)| + |\mathtt{rn}(P)| \qquad \#^*(P) = \lfloor P \rceil^* + \#_X(P)$$

Notice that $\#^*(P)$ gives the exact number of actions induced by propagators in $\mathcal{F}^*(P)$; in contrast, due to propagators c^r, $\#(P)$ gives the *least* number of such actions in $\mathcal{F}(P)$.

In general, we have $\#(P) \geq \#^*(P)$, but we can be more precise for a broad class of processes. We say that a process $P \not\equiv \mathbf{0}$ is in *normal form* if $P = (\nu\, \widetilde{n})\,(Q_1 \mid \ldots \mid Q_n)$, where each Q_i (with $i \in \{1, \ldots, n\}$) is not $\mathbf{0}$ and does not contain restriction nor parallel composition at the top-level. We have the following result; see Appendix B.3.1 for details.

Lemma 5.3.1. *If P is in normal form then $\#(P) \geq \frac{5}{3} \cdot \#^*(P)$.*

This result implies that the number of (extra) synchronizations induced by propagators in $\mathcal{F}(P)$ is larger than in $\mathcal{F}^*(P)$.

5.3.6 Static Correctness

We rely on an auxiliary predicate:

Definition 5.3.9 (Indexed Names)**.** Suppose some typing environments Γ, Δ. Let $\widetilde{x}$, $\widetilde{y}$ be two tuples of indexed names. We write $\texttt{indexed}_{\Gamma,\Delta}(\widetilde{y}, \widetilde{x})$ for the predicate

$$\forall z_i.(z_i \in \widetilde{x} \Leftrightarrow ((z_i, \dots, z_{i+m-1}) \subseteq \widetilde{y}) \wedge m = |\mathcal{H}^*((\Gamma,\Delta)(z_i))|)$$

We now state our static correctness results (typability with respect to MSTs) for the auxiliary function $\widehat{\mathcal{A}}^k_{\widetilde{y}}(\,\cdot\,)_g$, the breakdown function $\mathcal{A}^k_{\widetilde{y}}(P)$, and the optimized decomposition $\mathcal{F}^*(\,\cdot\,)$:

Lemma 5.3.2 (Typability of Auxiliary Breakdown: $\widehat{\mathcal{A}}^k_{\widetilde{y}}(\,\cdot\,)_g$)**.** *Let P be an initialized process. If $\Gamma \cdot X : \Delta_\mu; \Delta \vdash P \triangleright \diamond$ then:*

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \Theta \vdash \widehat{\mathcal{A}}^k_{\widetilde{y}}(P)_g \triangleright \diamond \quad (k > 0)$$

where

- $\widetilde{x} \subseteq \texttt{fn}(P)$ *such that* $\Delta \setminus \widetilde{x} = \emptyset$*;*
- $\widetilde{y} = \widetilde{v} \cdot \widetilde{m}$*, where* $\widetilde{m} = \texttt{codom}(g)$ *and* $\widetilde{v}$ *is such that* $\texttt{indexed}_{\Gamma,\Delta}(\widetilde{v}, \widetilde{x})$ *holds;*
- $\Theta = \Theta_\mu, \Theta_X(g)$ *where*
  - $\texttt{dom}(\Theta_\mu) = \{c^r_k, c^r_{k+1}, \dots, c^r_{k+\lfloor P\rceil^*-1}\} \cup \{\overline{c^r_{k+1}}, \dots, \overline{c^r_{k+\lfloor P\rceil^*-1}}\}$
  - *Let* $\widetilde{N} = (\mathcal{G}(\Gamma), \mathcal{G}(\Delta_\mu \cdot \Delta))(\widetilde{y})$*. Then*
  $$\Theta_\mu(c^r_k) = \begin{cases} \mu \mathsf{t}.?(\widetilde{N});\mathsf{t} & \textit{if } g \neq \emptyset \\ \langle \widetilde{N} \rangle & \textit{otherwise} \end{cases}$$
  - $\mathsf{balanced}(\Theta_\mu)$
  - $\Theta_X(g) = \bigcup_{X \in \texttt{dom}(g)} c^r_X : \langle \widetilde{M}_X \rangle$ *where* $\widetilde{M}_X = (\mathcal{G}(\Gamma), \mathcal{G}(\Delta_\mu))(g(X))$.

Proof. By induction on the structure of P; see Appendix B.3.2 for details. □

Lemma 5.3.3 (Typability of Breakdown)**.** *Let P be an initialized process. If $\Gamma; \Delta \vdash P \triangleright \diamond$ then*

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta \setminus \widetilde{x}), \Theta \vdash \mathcal{A}^k_{\widetilde{y}}(P) \triangleright \diamond \quad (k > 0)$$

where

- $\widetilde{x} \subseteq \texttt{fn}(P)$ *and* $\widetilde{y}$ *such that* $\texttt{indexed}_{\Gamma,\Delta}(\widetilde{y}, \widetilde{x})$ *holds.*
- *dom*$(\Theta) = \{c_k, c_{k+1}, \dots, c_{k+\lfloor P\rceil^*-1}\} \cup \{\overline{c_{k+1}}, \dots, \overline{c_{k+\lfloor P\rceil^*-1}}\}$
- $\Theta(c_k) = ?(\widetilde{M});\mathsf{end}$*, where* $\widetilde{M} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta))(\widetilde{y})$.
- $\mathsf{balanced}(\Theta)$

Proof. By induction of the structure of P and Lemma 5.3.2; see Appendix B.3.3 for details. □

We now (re)state the minimality result, now based on the decomposition $\mathcal{F}^*(\,\cdot\,)$.

Theorem 5.3.1 (Minimality Result for π, Optimized)**.** *Let P be a process with $\widetilde{u} = \texttt{fn}(P)$. If $\Gamma; \Delta \vdash P \triangleright \diamond$ then $\mathcal{H}^*(\Gamma\sigma); \mathcal{H}^*(\Delta\sigma) \vdash \mathcal{F}^*(P) \triangleright \diamond$, where $\sigma = \{\mathsf{init}(\widetilde{u})/\widetilde{u}\}$.*

Proof. Direct by Definition 5.3.7 and Lemma 5.3.3; see Appendix B.3.4 for details. □

5.3.7 Dynamic Correctness

As a complement to the minimality result, we have established that P and $\mathcal{F}^*(P)$ are *behaviorally equivalent* (Theorem 5.3.2). The behavioral equivalence that we consider is *MST-bisimilarity* (cf. Definition 5.3.19, $\approx^{\mathtt{M}}$), a variant of *characteristic bisimilarity*, one of the session-typed behavioral equivalences studied in [40].

Preliminaries

We require some auxiliary definitions and notations from [40].

Typed Labeled Transition System Our typed LTS is obtained by coupling the untyped LTS given in Figure 2.10 with a labeled transition relation on typing environments, given in Figure 5.6. Building upon the reduction relation for session environments in Definition 2.2.1, such a relation is defined on triples of environments by extending the LTSs in [43, 42]; it is denoted

$$(\Gamma_1, \Lambda_1, \Delta_1) \xrightarrow{\ell} (\Gamma_2, \Lambda_2, \Delta_2)$$

Recall that Γ admits weakening. Using this principle (not valid for Λ and Δ), we have $(\Gamma', \Lambda_1, \Delta_1) \overset{\ell}{\mapsto} (\Gamma', \Lambda_2, \Delta_2)$ whenever $(\Gamma, \Lambda_1, \Delta_1) \overset{\ell}{\mapsto} (\Gamma', \Lambda_2, \Delta_2)$. Some intuitions follow.

Input Actions are defined by Rules [SRv] and [ShRv]. In Rule [SRv] the type of value V and the type of the object associated to the session type on s should coincide. The resulting type tuple must contain the environments associated to V. The dual endpoint $\overline{s}$ cannot be present in the session environment: if it were present the only possible communication would be the interaction between the two endpoints (cf. Rule [Tau]). Following similar principles, Rule [ShRv] defines input actions for shared names.

Output Actions are defined by Rules [SSnd] and [ShSnd]. Rule [SSnd] states the conditions for observing action $(\nu\,\widetilde{m})\,s!\langle V\rangle$ on a type tuple $(\Gamma, \Lambda, \Delta \cdot s:S)$. The session environment $\Delta\,,\, s:S$ should include the session environment of the sent value V (denoted Δ' in the rule), *excluding* the session environments of names m_j in $\widetilde{m}$ which restrict the scope of value V (denoted Δ_j in the rule). Analogously, the linear variable environment Λ' of V should be included in Λ. The rule defines the scope extrusion of session names in $\widetilde{m}$; consequently, environments associated to their dual endpoints (denoted Δ'_j in the rule) appear in the resulting session environment. Similarly for shared names in $\widetilde{m}$ that are extruded. All free values used for typing V (denoted Λ' and Δ' in the rule) are subtracted from the resulting type tuple. The prefix of session s is consumed by the action. Rule [ShSnd] follows similar ideas for output actions on shared names: the name must be typed with $\langle U\rangle$; conditions on value V are identical to those on Rule [SSnd].

Other Actions Rules [Sel] and [Bra] describe actions for select and branch. Rule [Tau] defines internal transitions: it reduces the session environment (cf. Definition 2.2.1) or keeps it unchanged.

We illustrate Rule [SSnd] by means of an example:

Example 5.3.5. Consider environment tuple $(\Gamma;\, \emptyset;\, s :!\langle(!\langle S\rangle;\mathtt{end}) \multimap \diamond\rangle;\mathtt{end}, s' : S)$ and typed value $V = \lambda x.\, x!\langle s'\rangle.m?(z).\mathbf{0}$ with

$$\Gamma; \emptyset; s' : S, m :?(\mathtt{end});\mathtt{end} \vdash V \triangleright (!\langle S\rangle;\mathtt{end}) \multimap \diamond$$

Then, by Rule [SSnd], we can derive:

$$(\Gamma; \emptyset; s :!\langle(!\langle S\rangle;\mathtt{end}) \multimap \diamond\rangle;\mathtt{end}, s' : S) \xrightarrow{(\nu\, m)\, s!\langle V\rangle} (\Gamma; \emptyset; s : \mathtt{end}, \overline{m} :!\langle\mathtt{end}\rangle;\mathtt{end})$$

[SRv]

$$\frac{\overline{s} \notin \mathtt{dom}(\Delta) \qquad \Gamma;\Lambda';\Delta' \vdash V \triangleright U}{(\Gamma;\Lambda;\Delta, s:?(U);S) \xrightarrow{s?\langle V\rangle} (\Gamma;\Lambda,\Lambda';\Delta,\Delta', s:S)}$$

[ShRv]

$$\frac{\Gamma;\emptyset;\emptyset \vdash a \triangleright \langle U\rangle \qquad \Gamma;\Lambda';\Delta' \vdash V \triangleright U}{(\Gamma;\Lambda;\Delta) \xrightarrow{a?\langle V\rangle} (\Gamma;\Lambda,\Lambda';\Delta,\Delta')}$$

[SSnd]

$$\frac{\begin{array}{c}\Gamma,\Gamma';\Lambda';\Delta' \vdash V \triangleright U \qquad \Gamma';\emptyset;\Delta_j \vdash m_j \triangleright U_j \qquad \overline{s} \notin \mathtt{dom}(\Delta)\\ \Delta'\backslash(\cup_j\Delta_j) \subseteq (\Delta, s:S) \qquad \Gamma';\emptyset;\Delta'_j \vdash \overline{m}_j \triangleright U'_j \qquad \Lambda' \subseteq \Lambda\end{array}}{(\Gamma;\Lambda;\Delta, s:!\langle U\rangle;S) \xrightarrow{(\nu\,\widetilde{m})\,s!\langle V\rangle} (\Gamma,\Gamma';\Lambda\backslash\Lambda';(\Delta, s:S, \cup_j\Delta'_j)\backslash\Delta')}$$

[ShSnd]

$$\frac{\begin{array}{c}\Gamma,\Gamma';\Lambda';\Delta' \vdash V \triangleright U \qquad \Gamma';\emptyset;\Delta_j \vdash m_j \triangleright U_j \qquad \Gamma;\emptyset;\emptyset \vdash a \triangleright \langle U\rangle\\ \Delta'\backslash(\cup_j\Delta_j) \subseteq \Delta \qquad \Gamma';\emptyset;\Delta'_j \vdash \overline{m}_j \triangleright U'_j \qquad \Lambda' \subseteq \Lambda\end{array}}{(\Gamma;\Lambda;\Delta) \xrightarrow{(\nu\,\widetilde{m})\,a!\langle V\rangle} (\Gamma,\Gamma';\Lambda\backslash\Lambda';(\Delta, \cup_j\Delta'_j)\backslash\Delta')}$$

[Sel]

$$\frac{\overline{s} \notin \mathtt{dom}(\Delta) \qquad j \in I}{(\Gamma;\Lambda;\Delta, s:\oplus\{l_i:S_i\}_{i\in I}) \xrightarrow{s\oplus l_j} (\Gamma;\Lambda;\Delta, s:S_j)}$$

[Bra]

$$\frac{\overline{s} \notin \mathtt{dom}(\Delta) \quad j \in I}{(\Gamma;\Lambda;\Delta, s:\&\{l_i:T_i\}_{i\in I}) \xrightarrow{s\&l_j} (\Gamma;\Lambda;\Delta, s:S_j)}$$

[Tau]

$$\frac{\Delta_1 \longrightarrow \Delta_2 \vee \Delta_1 = \Delta_2}{(\Gamma;\Lambda;\Delta_1) \xrightarrow{\tau} (\Gamma;\Lambda;\Delta_2)}$$

Figure 5.6: Labeled Transition System for Typed Environments.

Observe how the protocol along s is partially consumed; also, the resulting session environment is extended with $\overline{m}$, the dual endpoint of the extruded name m.

Notation 5.3.1. *Given a value V of type U, we sometimes annotate the output action $(\nu\,\widetilde{m})\,n!\langle V\rangle$ with the type of V as $(\nu\,\widetilde{m})\,n!\langle V:U\rangle$.*

The typed LTS combines the LTSs in Figure 2.10 and Figure 5.6.

Definition 5.3.10 (Typed Transition System)**.** A *typed transition relation* is a typed relation $\Gamma;\Delta_1 \vdash P_1 \xrightarrow{\ell} \Delta_2 \vdash P_2$ where:

1. $P_1 \xrightarrow{\ell} P_2$ and
2. $(\Gamma,\emptyset,\Delta_1) \xrightarrow{\ell} (\Gamma,\emptyset,\Delta_2)$ with $\Gamma;\emptyset;\Delta_i \vdash P_i \triangleright \diamond$ $(i = 1,2)$.

We write $\Rightarrow$ for the reflexive and transitive closure of $\rightarrow$, $\overset{\ell}{\Rightarrow}$ for the transitions $\Rightarrow\xrightarrow{\ell}\Rightarrow$, and $\overset{\hat{\ell}}{\Rightarrow}$ for $\overset{\ell}{\Rightarrow}$ if $\ell \neq \tau$ otherwise $\Rightarrow$.

A typed transition relation requires type judgements with an empty Λ, i.e., an empty environment for linear higher-order types. Notice that for open process terms (i.e., with free variables), we can always apply Rule [EProm] (cf. Figure 2.5) and obtain an empty Λ. As it will be clear below (cf. Definition 5.3.12), we will be working with closed process terms, i.e., processes without free variables.

Typed Relations We now define *typed relations* and *contextual equivalence* (i.e., barbed congruence). To define typed relations, we first define *confluence* over session environments Δ. Recall that Δ captures session communication, which is deterministic. The notion of confluence allows us to abstract away from alternative computation paths that may arise due to non-interfering reductions of session names.

Definition 5.3.11 (Session Environment Confluence)**.** Two session environments Δ_1 and Δ_2 are *confluent*, denoted $\Delta_1 \rightleftharpoons \Delta_2$, if there exists a Δ such that: i) $\Delta_1 \longrightarrow^* \Delta$ and ii) $\Delta_2 \longrightarrow^* \Delta$ (here we write $\longrightarrow^*$ for the multi-step reduction in Definition 2.2.1).

We illustrate confluence by means of an example:

Example 5.3.6 (Session Environment Confluence)**.** Consider the (balanced) session environments:

$$\begin{array}{rcl}\Delta_1 &=& \{s_1 : T_1, s_2 :?(U_2);\mathtt{end}, \overline{s_2} :!\langle U_2\rangle;\mathtt{end}\}\\ \Delta_2 &=& \{s_1 : T_1, s_2 :!\langle U_1\rangle;?(U_2);\mathtt{end}, \overline{s_2} :?(U_1);!\langle U_2\rangle;\mathtt{end}\}\end{array}$$

Following Definition 2.2.1, we have that $\Delta_1 \longrightarrow \{s_1 : T_1, s_2 : \mathtt{end}, \overline{s_2} : \mathtt{end}\}$ and $\Delta_2 \longrightarrow\longrightarrow \{s_1 : T_1, s_2 : \mathtt{end}, \overline{s_2} : \mathtt{end}\}$. Therefore, Δ_1 and Δ_2 are confluent. $\square$

Typed relations relate only closed processes whose session environments are balanced and confluent:

Definition 5.3.12 (Typed Relation)**.** We say that a binary relation over typing judgements

$$\Gamma; \emptyset; \Delta_1 \vdash P_1 \triangleright \diamond \;\Re\; \Gamma; \emptyset; \Delta_2 \vdash P_2 \triangleright \diamond$$

is a *typed relation* whenever:

1. P_1 and P_2 are closed;
2. Δ_1 and Δ_2 are balanced (cf. Definition 2.2.1); and
3. $\Delta_1 \rightleftharpoons \Delta_2$ (cf. Definition 5.3.11).

Notation 5.3.2 (Typed Relations)**.** *We write*

$$\Gamma; \Delta_1 \vdash P_1 \;\Re\; \Delta_2 \vdash P_2$$

to denote the typed relation $\Gamma; \emptyset; \Delta_1 \vdash P_1 \triangleright \diamond \;\Re\; \Gamma; \emptyset; \Delta_2 \vdash P_2 \triangleright \diamond$.

Characteristic Bisimilarity Characteristic bisimilarity equates typed processes by relying on *characteristic processes* and *trigger processes*. These notions, which we recall below, need to be adjusted for our purposes.

Definition 5.3.13 (Characteristic trigger process [40])**.** The characteristic trigger process for type C is

$$t \Leftarrow_{\mathtt{C}} v : C \overset{\mathsf{def}}{=} t?(x).(\nu\, s)(s?(y).\{\!|C|\!\}^y \mid \overline{s}!\langle v\rangle.\mathbf{0})$$

where $\{\!|C|\!\}^y$ is the characteristic process for C on name y (cf. Figure 2.12).

Definition 5.3.14 (Characteristic Bisimilarity)**.** A typed relation $\Re$ is a *characteristic bisimulation* if for all $\Gamma; \Delta_1 \vdash P_1 \;\Re\; \Delta_2 \vdash Q_1$,

1) Whenever $\Gamma; \Delta_1 \vdash P_1 \xmapsto{(\nu\, \widetilde{m_1})\, n!\langle V_1 : U_1\rangle} \Delta_1' \vdash P_2$ then there exist Q_2, V_2, Δ_2' such that $\Gamma; \Delta_2 \vdash Q_1 \overset{(\nu\, \widetilde{m_2})\, n!\langle V_2 : U_2\rangle}{\Longmapsto} \Delta_2' \vdash Q_2$ and, for a fresh t,
$$\Gamma; \Delta_1'' \vdash (\nu\, \widetilde{m_1})(P_2 \mid t \Leftarrow_{\mathtt{C}} V_1 : U_1) \;\Re\; \Delta_2'' \vdash (\nu\, \widetilde{m_2})(Q_2 \mid t \Leftarrow_{\mathtt{C}} V_2 : U_2)$$

2) For all $\Gamma; \Delta_1 \vdash P_1 \overset{\ell}{\mapsto} \Delta_1' \vdash P_2$ such that ℓ is not an output, there exist Q_2, Δ_2' such that $\Gamma; \Delta_2 \vdash Q_1 \overset{\hat{\ell}}{\Longmapsto} \Delta_2' \vdash Q_2$ and $\Gamma; \Delta_1' \vdash P_2 \;\Re\; \Delta_2' \vdash Q_2$; and

3) The symmetric cases of 1 and 2.

The largest such bisimulation is called *characteristic bisimilarity*, denoted by $\approx^{\mathtt{C}}$.

MST-bisimilarity and Main Result

We introduce MST-bisimilarity (denoted $\approx^{\mathtt{M}}$), and discuss key differences with respect to characteristic bisimilarity. We let an action along a name n to be mimicked by an action on a possibly indexed name n_i, for some i.

Definition 5.3.15 (Indexed name)**.** Given a name n, we write $\breve{n}$ to either denote n or any indexed name n_i, with $i > 0$.

Suppose we wish to relate P and Q using $\approx^{\mathtt{M}}$, and that P performs an output action involving name v. In our setting, Q should send a *tuple* of names: the decomposition of v. The second difference is that output objects should be related by the relation $\diamond$:

Definition 5.3.16 (Relating names)**.** We define the relation on names $\diamond$ as follows:

$$\frac{}{\epsilon \diamond \epsilon} \qquad \frac{\Gamma; \Lambda; \Delta \vdash n_i \triangleright C}{n_i \diamond (n_i, \ldots, n_{i+|\mathcal{G}(C)|-1})} \qquad \frac{\tilde{n} \diamond \tilde{m}_1 \quad n_i \diamond \tilde{m}_2}{\tilde{n}, n_i \diamond \tilde{m}_1, \tilde{m}_2}$$

where ϵ denotes the empty list.

Our variant of trigger processes is defined as follows:

Definition 5.3.17 (Minimal characteristic trigger process)**.** Given a type C, the trigger process is
$$t \Leftarrow_{\mathtt{m}} v_i : C \overset{\mathsf{def}}{=} t_1?(x).(\nu\, s_1)(s_1?(\tilde{y}).\langle C\rangle_i^y \mid \overline{s_1}!\langle\tilde{v}\rangle.\mathbf{0})$$

where $v_i \diamond \tilde{v}$, $y_i \diamond \tilde{y}$, and $\langle C\rangle_i^y$ is a minimal characteristic process for type C on name y (see Definition 5.3.18 for a definition).

Definition 5.3.18 (Minimal characteristic processes)**.**

$$\begin{aligned}
\langle ?(C);S\rangle_i^u &\overset{\mathsf{def}}{=} u_i?(x).(t_1!\langle u_{i+1}, \ldots, u_{i+|\mathcal{G}(S)|}\rangle.\mathbf{0} \mid \langle C\rangle_i^x) \\
\langle !\langle C\rangle;S\rangle_i^u &\overset{\mathsf{def}}{=} u_i!\langle\langle C\rangle_{\mathtt{c}}\rangle.t_1!\langle u_{i+1}, \ldots, u_{i+|\mathcal{G}(S)|}\rangle.\mathbf{0} \\
\langle \mathsf{end}\rangle_i^u &\overset{\mathsf{def}}{=} \mathbf{0} \\
\langle\langle C\rangle\rangle_i^u &\overset{\mathsf{def}}{=} u_1!\langle\langle C\rangle_{\mathtt{c}}\rangle.t_1!\langle u_1\rangle.\mathbf{0} \\
\langle \mu \mathsf{t}.S\rangle_i^u &\overset{\mathsf{def}}{=} \langle S\{\mathsf{end}/\mathsf{t}\}\rangle_i^u \\
\langle S\rangle_{\mathtt{c}} &\overset{\mathsf{def}}{=} \tilde{s} \quad (|\tilde{s}| = |\mathcal{G}(S)|, \tilde{s} \text{ fresh}) \\
\langle\langle C\rangle\rangle_{\mathtt{c}} &\overset{\mathsf{def}}{=} a_1 \quad (a_1 \text{ fresh})
\end{aligned}$$

where t_1 is a fresh (indexed) name.

We are now ready to define MST-bisimilarity:

Definition 5.3.19 (MST-Bisimilarity)**.** A typed relation $\Re$ is an *MST bisimulation* if for all $\Gamma_1; \Delta_1 \vdash P_1 \ \Re \ \Gamma_2; \Delta_2 \vdash Q_1$,

1. Whenever $\Gamma_1; \Delta_1 \vdash P_1 \xrightarrow{(\nu\, \widetilde{m_1})\, n!\langle v{:}C_1\rangle} \Delta_1'; \Lambda_1' \vdash P_2$ then there exist Q_2, Δ_2', and σ_v such that $\Gamma_2; \Delta_2 \vdash Q_1 \xRightarrow{(\nu\, \widetilde{m_2})\, \breve{n}!\langle \tilde{v}{:}\mathcal{H}^*(C)\rangle} \Delta_2' \vdash Q_2$ where $v\sigma_v \diamond \tilde{v}$ and, for a fresh t,
$$\begin{aligned}\Gamma; \Delta_1'' \vdash (\nu\, \widetilde{m_1})(P_2 \mid t \Leftarrow_{\mathtt{C}} v : C_1) \Re \\ \Delta_2'' \vdash (\nu\, \widetilde{m_2})(Q_2 \mid t \Leftarrow_{\mathtt{m}} v\sigma : C_1)\end{aligned}$$
2. Whenever $\Gamma_1; \Delta_1 \vdash P_1 \xrightarrow{n?(v)} \Delta_1' \vdash P_2$ then there exist Q_2, Δ_2', and σ_v such that $\Gamma_2; \Delta_2 \vdash Q_1 \xRightarrow{\breve{n}?(\tilde{v})} \Delta_2' \vdash Q_2$ where $v\sigma_v \diamond \tilde{v}$ and $\Gamma_1; \Delta_1' \vdash P_2 \ \Re \ \Gamma_2; \Delta_2' \vdash Q_2$,
3. Whenever $\Gamma_1; \Delta_1 \vdash P_1 \xrightarrow{\ell} \Delta_1' \vdash P_2$, with ℓ not an output or input, then there exist Q_2 and Δ_2' such that $\Gamma_2; \Delta_2 \vdash Q_1 \xRightarrow{\hat{\ell}} \Delta_2' \vdash Q_2$ and $\Gamma_1; \Delta_1' \vdash P_2 \ \Re \ \Gamma_2; \Delta_2' \vdash Q_2$ and $\mathsf{sub}(\ell) = n$ implies $\mathsf{sub}(\hat{\ell}) = \breve{n}$.
4. The symmetric cases of 1, 2, and 3.

The largest such bisimulation is called *MST bisimilarity* ($\approx^{\mathtt{M}}$).

We can now state our dynamic correctness result:

Theorem 5.3.2 (Operational Correspondence)**.** *Let P be a process such that $\Gamma; \Delta \vdash P$. We have*
$$\Gamma; \Delta \vdash P \ \approx^{\mathtt{M}} \ \mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta) \vdash \mathcal{F}^*(P)$$

The proof of this theorem is by coinduction: we exhibit a binary relation $\mathcal{S}$ such that $(P, \mathcal{F}^*(P)) \in \mathcal{S}$ and is an MST bisimulation. The proof that $\mathcal{S}$ is an MST bisimulation is given by Lemma 5.3.7 and Lemma 5.3.10 in the next sub-section.

5.3.8 Proof of Theorem 5.3.2

Preliminaries

In this section we define the relation $\mathcal{S}$ used in the proof of Theorem 5.3.2. We require some auxiliary notions.

First, we define a relation $\diamond$ on processes, which corresponds to the extension of the relation on indexed names given by Definition 5.3.16:

Definition 5.3.20 (Relation $\diamond$ on processes)**.** Given the relation $\diamond$ on names (Definition 5.3.16), we define the relation $\diamond$ on processes as follows:

$$[\text{IPSnd}]\ \frac{P\sigma \diamond P' \qquad v\sigma \diamond \tilde{v} \qquad \sigma = \mathsf{next}(n_i)}{n_i!\langle v\rangle.P \diamond n_i!\langle \tilde{v}\rangle.P'} \qquad [\text{IPInact}]\ \frac{}{\mathbf{0} \diamond \mathbf{0}}$$

$$[\text{IPRcv}]\ \frac{P\sigma \diamond P' \qquad v\sigma \diamond \tilde{v} \qquad \sigma = \mathsf{next}(n_i)}{n_i?(y).P \diamond n_i?(\tilde{y}).P'} \qquad [\text{IPNews}]\ \frac{P \diamond P' \qquad \tilde{m}_1 \diamond \tilde{m}_2}{(\nu\, \tilde{m}_1)\, P \diamond (\nu\, \tilde{m}_2)\, P'}$$

We have the following properties for $\diamond$:

Lemma 5.3.4. *We have:* $(t \Leftarrow_{\mathtt{C}} v : C\{^{t_1, v_i}/_{t, v}\},\ t_1 \Leftarrow_{\mathtt{m}} v_i : C) \in \diamond$ *for* $i > 0$.

Proof. Directly by Definitions 5.3.13, 5.3.17 and 5.3.20. ☐

Lemma 5.3.5. *Relation* $\diamond$ *is an MST bisimulation.*

Proof. Straightforward by transition induction. ☐

As we have seen, the output clause of MST bisimilarity "appends" trigger processes in parallel to the processes under comparison. The following definitions introduce notations which are useful for distinguishing the triggers included in a process.

Definition 5.3.21 (Trigger Collections)**.** We let H, H' to range over *trigger collections*: processes of the form $P_1 \mid \cdots \mid P_n$ (with $n \geq 1$), where each P_i is a trigger process or a process that originates from a trigger process.

Example 5.3.7. Let $H_1 = t \Leftarrow_{\mathtt{C}} v : C \mid (\!|C|\!)^u \mid t'!\langle n \rangle.\mathbf{0}$ where v, t, t', u, n are channel names, C a channel type. We could see that $(\!|C|\!)^u$ and $t'!\langle n \rangle.\mathbf{0}$ originate from a trigger process. Thus, H_1 is a trigger collection.

Definition 5.3.22 (Process in parallel with a trigger or a characteristic process)**.** We write $P \parallel Q$ to stand for $P \mid Q$ where either P or Q is a trigger collection.

Definition 5.3.23 (Propagators of P)**.** We define $\mathsf{fpn}(P)$ to denote the set of free propagator names in P.

The following definition is useful when constructing an MST bisimulation for *recursive processes*, i.e., processes P such that $\mathsf{frv}(P)$ holds (cf. Definition 5.3.5). The *depth* of a recursive variable denotes the number of prefixes preceding the occurrence of a recursive variable X or a subprocess of the form $\mu X.Q$; in turn, this depth will be related to the current trio mimicking a given process.

Definition 5.3.24 (Depth of recursive variable)**.** Let P be a recursive process. The functions $\delta(P)$ and $\widehat{\delta}(P)$ are mutually defined as follows:

$$\delta(P) = \begin{cases} \widehat{\delta}(P) & \text{if } \mathsf{frv}(P) \\ 0 & \text{otherwise} \end{cases} \qquad \widehat{\delta}(P) = \begin{cases} 0 & \text{if } P = \mu X.Q \text{ or } P = X \\ \delta(Q) + 1 & \text{if } P = \alpha.Q \\ \delta(Q) + \delta(R) & \text{if } P = Q \mid R \\ \delta(Q) & \text{if } P = (\nu\, s)\, Q \end{cases}$$

Definition 5.3.25. The predicate $\mathcal{D}_X(P, Q, d)$ holds whenever there exist Q, Q' such that (i) $P \equiv Q'\{^{\mu X.Q}/_X\}$ and (ii) $\delta(Q') = d$.

Finally, we introduce notations on (indexed) names and substitutions:

Definition 5.3.26 (Name breakdown)**.** Let $u_i : C$ be an indexed name with its session type. We write $\mathsf{bn}(u_i : C)$ to denote

$$\mathsf{bn}(u_i : C) = (u_i, \ldots, u_{i+|\mathcal{H}^*(C)|-1})$$

We extend $\mathsf{bn}(\cdot)$ to work on lists of assignments (name, type), as follows:

$$\mathsf{bn}((u_i^1, \ldots, u_j^n) : (C_1, \ldots, C_n)) = \mathsf{bn}(u_i^1 : C_1) \cdot \ldots \cdot \mathsf{bn}(u_j^n : C_n)$$

Definition 5.3.27 (Indexed names substitutions)**.** Let $\widetilde{u} = (a, b, s, \overline{s}, \overline{s}', s', r, r', \ldots)$ be a finite tuple of names. We write $\mathsf{index}(\widetilde{u})$ to denote

$$\mathsf{index}(\widetilde{u}) = \{\{^{a_1, b_1, s_i, \overline{s}_i, s'_j, \overline{s}'_j, r_1, r'_1, \cdots}/_{a, b, s, s', r, r', \ldots}\} : i, j, \ldots > 0\}$$

P	$\mathcal{C}^{\tilde{u}}_{\tilde{x}}(P)$	
$Q_1 \parallel Q_2$	$\{R_1 \parallel R_2 : R_1 \in \mathcal{C}^{\tilde{u}_1}_{\tilde{y}}(Q_1), R_2 \in \mathcal{C}^{\tilde{u}_2}_{\tilde{z}}(Q_2)\}$	$\widetilde{y} = \mathtt{fnb}(Q_1, \tilde{x})$ $\widetilde{z} = \mathtt{fnb}(Q_2, \tilde{x})$ $\{\tilde{u}/\tilde{x}\} = \{\tilde{u}_1/\tilde{y}\} \cdot \{\tilde{u}_2/\tilde{z}\}$
$(\nu\, s : C)\, Q$	$\{(\nu\, \widetilde{s} : \mathcal{G}(C))\ R : \mathcal{C}^{\tilde{u}}_{\tilde{x}}(Q\sigma)\}$	$\widetilde{s} = (s_1, \ldots, s_{\lvert\mathcal{H}^*(C)\rvert})$ $\sigma = \{s_1\overline{s_1}/s\overline{s}\}$
Q	if $\mathcal{D}_X(Q, Q', d)$: $\widehat{\mathcal{C}}^{\tilde{u}}_{\tilde{x}}(\mu X.Q')^d$ otherwise: $\{(\nu\, \widetilde{c})\, R\ :\ R = \overline{c_k}!\langle\widetilde{u}\rangle \mid \mathcal{A}^k_{\tilde{x}}(Q)\}$ $\cup$ $\{(\nu\, \widetilde{c})\, R\ :\ R \in \mathcal{J}^{\tilde{u}}_{\tilde{x}}(Q)\}$	$\widetilde{c} = \mathsf{fpn}(R)$ $d \geq 1$ (cf. Definition 5.3.25 for $\mathcal{D}_{-}(\cdot, \cdot, \cdot)$)
H	$\{H' : H\{\tilde{u}/\tilde{x}\} \diamond H'\}$	
P	$\mathcal{J}^{\tilde{u}}_{\tilde{x}}(P)$	
$n_i?(y).Q$	$\{n_l\rho?(\widetilde{y}).\overline{c_{k+1}}!\langle\widetilde{z}\rho\rangle \mid \mathcal{A}^{k+1}_{\tilde{z}}(Q\sigma)\}$	$y : S$ $\widetilde{y} = (y_1, \ldots, y_{\lvert\mathcal{H}^*(S)\rvert})$ $\widetilde{w} = (\mathtt{lin}(n_i))\, ?\, \{n_i\} \colon \epsilon$ $\widetilde{z} = \mathtt{fnb}(Q, \widetilde{x}\widetilde{y} \setminus \widetilde{w})$ $\rho = \{\tilde{u}/\tilde{x}\}$ $\sigma = \mathsf{next}(n_i) \cdot \{y_1/y\}$ $l = (\mathsf{tr}(S))\, ?\, \iota(S) \colon i$
$n_i!\langle y_j\rangle.Q$	$\{n_l\rho!\langle\widetilde{y}\rho\rangle.\overline{c_{k+1}}!\langle\widetilde{z}\rho\rangle \mid \mathcal{A}^{k+1}_{\tilde{z}}(Q\sigma)\}$	$y_j : S$ $\widetilde{y} = (y_j, \ldots, y_{j+\lvert\mathcal{H}^*(S)\rvert-1})$ $\widetilde{w} = (\mathtt{lin}(n_i))\, ?\, \{n_i\} \colon \epsilon$ $\widetilde{z} = \mathtt{fnb}(Q, \widetilde{x} \setminus \widetilde{w})$ $\rho = \{\tilde{u}/\tilde{x}\}$ $\sigma = \mathsf{next}(n_i)$ $l = (\mathsf{tr}(S))\, ?\, \iota(S) \colon i$
$Q_1 \mid Q_2$	$\{\overline{c_k}!\langle\widetilde{y}\rho\rangle.\overline{c_{k+l}}!\langle\widetilde{z}\rho\rangle \mid \mathcal{A}^k_{\tilde{y}}(Q_1) \mid \mathcal{A}^{k+l}_{\tilde{z}}(Q_2)\}$ $\cup$ $\{(R_1 \mid R_2) : R_1 \in \mathcal{C}^{\tilde{u}_1}_{\tilde{y}}(Q_1), R_2 \in \mathcal{C}^{\tilde{u}_2}_{\tilde{z}}(Q_2)\}$	$\widetilde{y} = \mathtt{fnb}(Q_1, \widetilde{x})$ $\widetilde{z} = \mathtt{fnb}(Q_2, \widetilde{x})$ $\rho = \{\tilde{u}/\tilde{x}\}$ $l = \lfloor Q_1 \rceil^*$
$\mu X.P$	$\left\{\overline{c^r_{k+1}}!\langle\widetilde{z}\rho\rangle.\mu X.c^r_X?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{y}\rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\tilde{x}}(P)_g\right\}$	$\widetilde{n} = \mathtt{fn}(P)$ $\widetilde{n} : \widetilde{C} \wedge \widetilde{m} = \mathsf{bn}(\widetilde{n} : \widetilde{C})$ $\widetilde{z} = \widetilde{x} \cup \widetilde{m},\ \lvert\widetilde{z}\rvert = \lvert\widetilde{y}\rvert$ $g = \{X \mapsto \widetilde{m}\}$
$\mathbf{0}$	$\{\mathbf{0}\}$	

Table 5.6: The sets $\mathcal{C}^{\tilde{u}}_{\tilde{x}}(P)$ (upper part) and $\mathcal{J}^{\tilde{u}}_{\tilde{x}}(P)$ (lower part).

The relation $\mathcal{S}$: Ingredients and Properties

Having all auxiliary notions in place, we are ready to define $\mathcal{S}$:

P	$\widehat{\mathcal{C}}^{\tilde{u}}_{\tilde{x}}(P)^d$
$\mu X.Q$	$N \cup \Big\{(\nu\, \widetilde{c})\, (\mu X.c^r_X?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{y}\rangle.X \mid c^r_k?(\widetilde{x}).c^r_X!\langle\widetilde{x}\rangle.X \mid R) \,:\, R \in \widehat{\mathcal{J}}^k_{\tilde{x}}(Q)^d_g\Big\}$ *where:* $N = \begin{cases} M \cup \Big\{(\nu\, \widetilde{c})\, (\overline{c^r_k}!\langle\widetilde{z}\rho\rangle.\mu X.c^r_X?(\widetilde{y}).\overline{c^r_k}!\langle\widetilde{y}\rangle.X \mid \widehat{\mathcal{A}}^k_{\tilde{x}}(Q)_g)\Big\} & \text{if } d = 0 \\ \emptyset & \text{otherwise} \end{cases}$ $M = \begin{cases} \{(\nu\, \widetilde{c})\, (\mu X.c^r_X?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{y}\rangle.X \mid R \mid \\ \qquad c^r_X!\langle\widetilde{x}\rho\rangle.\mu X.c^r_k?(\widetilde{x}).c^r_X!\langle\widetilde{x}\rangle.X) \,:\, R \in \widehat{\mathcal{J}}^k_{\tilde{x}}(Q)^0_g\} & \text{if } g \neq \emptyset \\ \{(\nu\, \widetilde{c})\, (\mu X.c^r_X?().\overline{c^r_{k+1}}!\langle\rangle.X \mid R \mid \\ \qquad c^r_X!\langle\rangle \mid \mu X.c^r_k?().(c^r_X!\langle\rangle \mid X)) \,:\, R \in \widehat{\mathcal{J}}^k_{\tilde{x}}(Q)^0_g\} & \text{otherwise} \end{cases}$ *Also:* $\widetilde{c} = \mathsf{fpn}(R) \quad \widetilde{x} = \mathtt{fs}(Q) \quad g = \{X \mapsto \widetilde{x}\} \quad \rho = \{\tilde{u}/\tilde{x}\}$

Table 5.7: The set $\widehat{\mathcal{C}}^{\tilde{u}}_{\tilde{x}}(P)$. The set $\widehat{\mathcal{J}}^k_{\tilde{x},\rho}(P)^d_g$ is defined in Tables 5.8 and 5.9.

Definition 5.3.28 (Relation $\mathcal{S}$). Let $P\{\tilde{u}/\tilde{x}\}$ be a well-typed process, with $\tilde{u} : \widetilde{C}$. We define the relation $\mathcal{S}$ as follows:

$$\begin{aligned} \mathcal{S} = \{(P\{\tilde{u}/\tilde{x}\}, R) \,:\, & R \in \mathcal{C}^{\tilde{w}}_{\tilde{y}}(P\sigma), \\ & \sigma \in \mathsf{index}(\tilde{u} \cup \widetilde{x} \cup \mathtt{fn}(P)),\ \widetilde{w} = \mathsf{bn}(\tilde{u}\sigma : \widetilde{C}),\ \widetilde{y} = \mathsf{bn}(\widetilde{x}\sigma : \widetilde{C})\} \end{aligned}$$

where the set $\mathcal{C}^{-}_{-}(\,\cdot\,)$ as in Table 5.6.

The definition of $\mathcal{S}$ follows Parrow's proof of dynamic correctness for the trios in the untyped π-calculus. Intuitively, processes in the set $\mathcal{C}^{\tilde{w}}_{\tilde{y}}(P)$ represent processes that are "correlated" to P, up to synchronizations induced by propagators. In our case, the presence of trigger processes and recursive processes induces significant differences with respect to Parrow's definition. Given a process P, we have two mutually defined sets: $\mathcal{C}^{\tilde{u}}_{\tilde{x}}(P)$ and $\mathcal{J}^{\tilde{u}}_{\tilde{x}}(P)$, which are both given in Table 5.6. The idea is that $\mathcal{C}^{\tilde{u}}_{\tilde{x}}(P)$ deals with trigger processes and top-level activation of propagators, whereas $\mathcal{J}^{\tilde{u}}_{\tilde{x}}(P)$ collects processes without triggers that are involved in the overall activation of propagators. Handling recursive processes requires dedicated treatment; this is formalized by the auxiliary sets $\widehat{\mathcal{C}}^{\tilde{u}}_{\tilde{x}}(P)$ (defined in Table 5.7) and $\widehat{\mathcal{J}}^k_{\tilde{x},\rho}(P)^d_g$ (defined in Tables 5.8 and 5.9).

Properties We prove a series of lemmas that establish a form of *operational correspondence*, divided in completeness and soundness properties. We first need the following result. Following Parrow [50], we refer to prefixes that do not correspond to prefixes of the original process (i.e. prefixes on propagators c_i), as *non-essential prefixes.* The relation $\mathcal{S}$ is closed under reductions that involve non-essential prefixes.

Lemma 5.3.6. *Given an indexed process $P_1\{\tilde{u}/\tilde{x}\}$, the set $\mathcal{C}^{\tilde{u}}_{\tilde{x}}(P_1)$ is closed under τ transitions on non-essential prefixes. That is, if $R_1 \in \mathcal{C}^{\tilde{u}}_{\tilde{x}}(P_1)$ and $R_1 \xrightarrow{\tau} R_2$ is inferred from the actions on non-essential prefixes, then $R_2 \in \mathcal{C}^{\tilde{u}}_{\tilde{x}}(P_1)$.*

Proof (Sketch). By the induction on the structure of P_1 and the inspection of definition of $\mathcal{C}^{-}_{-}(\,\cdot\,)$ and $\mathcal{J}^{-}_{-}(\,\cdot\,)$ in Table 5.6 and Table 5.7. □

P	$\widehat{\mathfrak{J}}^{k}_{\tilde{x},\rho}(P)^{d}_{g}$ when $g \neq \emptyset$
$n_i!\langle y_j\rangle.Q$	$\left\{n_l!\langle\widetilde{y}\rho\rangle.\overline{c^r_{k+1}}!\langle\widetilde{z}\rho\rangle.\mu X.c^r_k?(\widetilde{x}).n_l!\langle\widetilde{y}\rangle.\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(Q\sigma)_g\right\}$ if $\delta(Q) = d$ $N \cup \left\{\mu X.c^r_k?(\widetilde{x}).n_l!\langle\widetilde{y}\rangle.\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X \mid R \,:\, R \in \widehat{\mathfrak{J}}^{k+1}_{\tilde{z},\rho}(Q\sigma)^d_g\right\}$ otherwise $N = \begin{cases}\{B \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(Q\sigma)_g\} & \text{if } \delta(Q) = d+1\\ \emptyset & \text{otherwise}\end{cases}$ $B = \overline{c^r_{k+1}}!\langle\widetilde{z}\rho\rangle.\mu X.c^r_k?(\widetilde{x}).n_l!\langle\widetilde{y}\rangle.\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X$ $y_j : T$ $\quad\widetilde{y} = (y_j, \ldots, y_{j+\lvert\mathcal{H}^*(T)\rvert-1})$ $\quad\widetilde{w} = (\mathtt{lin}(u_i))\,?\,\{u_i\}\colon\epsilon$ $\widetilde{z} = g(X) \cup \mathtt{fnb}(Q, \widetilde{x} \setminus \widetilde{w})$ $\quad l = (\mathtt{tr}(u_i))\,?\,\iota(S)\colon i$ $\quad\sigma = \mathsf{next}(u_i)$
$n_i?(y).Q$	$\left\{n_l?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{z}\rho\rangle.\mu X.c^r_k?(\widetilde{x}).n_l?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(Q\sigma)_g\right\}$ if $\delta(Q) = d$ $N \cup \left\{\mu X.c^r_k?(\widetilde{x}).n_l?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X \mid R : R \in \widehat{\mathfrak{J}}^{k+1}_{\tilde{z},\rho}(Q\sigma)^d_g\right\}$ otherwise $N = \begin{cases}\{B \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(Q\sigma)_g\} & \text{if } \delta(Q) = d+1\\ \emptyset & \text{otherwise}\end{cases}$ $B = \overline{c^r_{k+1}}!\langle\widetilde{z}\rho\rangle.\mu X.c^r_k?(\widetilde{x}).n_l?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X$ $y : T$ $\quad\widetilde{y} = (y_1, \ldots, y_{\lvert\mathcal{H}^*(T)\rvert})$ $\quad\widetilde{w} = (\mathtt{lin}(u_i))\,?\,\{u_i\}\colon\epsilon$ $\widetilde{z} = g(X) \cup \mathtt{fnb}(Q, \widetilde{x}\widetilde{y} \setminus \widetilde{w})$ $\quad l = (\mathtt{tr}(u_i))\,?\,\iota(S)\colon i$ $\quad\sigma = \mathsf{next}(u_i) \cdot \{y_1/y\}$
$Q_1 \mid Q_2$	$N \cup \{\mu X.c^r_k?(\widetilde{x}).(\overline{c^r_{k+1}}!\langle\widetilde{y}_1\rangle.X \mid c^r_{k+l+1}!\langle\widetilde{y}_2\rangle) \mid R_1 \mid R_2 \,:$ $R_1 \in \widehat{\mathfrak{J}}^{k+1}_{\tilde{y}_1,\rho}(Q_1)^d_{g_1}, R_2 \in \widehat{\mathfrak{J}}^{k+l+1}_{\tilde{y}_2,\rho}(Q_2)^d_{g_2}\}$ $N = \begin{cases}\{B \mid \widehat{\mathcal{A}}^{k+1}_{\tilde{y}_1}(Q_1)_{g_1} \mid \\ \widehat{\mathcal{A}}^{k+l+1}_{\tilde{y}_2}(Q_2)_{g_2}\} & \text{if } \delta(Q_1) = d\\ \emptyset & \text{otherwise}\end{cases}$ $B = \overline{c^r_{k+1}}!\langle\widetilde{y}_1\rho\rangle.\mu X.c^r_k?(\widetilde{x}).(\overline{c^r_{k+1}}!\langle\widetilde{y}_1\rangle.X \mid c^r_{k+l+1}!\langle\widetilde{y}_2\rangle)$ $\mathsf{frv}(Q_1)$ $\quad\widetilde{y}_1 = g(X) \cup \mathtt{fnb}(Q_1, \widetilde{x})$ $\quad\widetilde{y}_2 = \mathtt{fnb}(Q_2, \widetilde{x})$ $\quad l = \lfloor Q_1 \rceil^*$
$(\nu\, s)\, Q$	$N \cup \left\{\mu X.(\nu\, \widetilde{s} : \mathcal{H}^*(C))\, c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle\widetilde{z}\rangle.X \mid R : R \in \widehat{\mathfrak{J}}^{k+1}_{\tilde{z},\rho}(Q\sigma)^d_g\right\}$ $N = \begin{cases}\{(\nu\, \widetilde{s} : \mathcal{H}^*(C))\, B \mid \\ \widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(Q\sigma)_g\} & \text{if } \delta(Q_1) = d\\ \emptyset & \text{otherwise}\end{cases}$ $B = \overline{c_{k+1}}!\langle\widetilde{z}\rho\rangle.\mu X.(\nu\, \widetilde{s} : \mathcal{H}^*(C))\, c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle\widetilde{z}\rangle.X$ $s : S$ $\quad\widetilde{s} = (s_1, \ldots, s_{\lvert\mathcal{H}^*(S)\rvert})$ $\quad\widetilde{\overline{s}} = (\overline{s_1}, \ldots, \overline{s_{\lvert\mathcal{H}^*(S)\rvert}})$ $\widetilde{n} = (\mathtt{lin}(s))\,?\,\widetilde{n}\colon\epsilon$ $\quad\widetilde{z} = \widetilde{x}, \widetilde{s}, \widetilde{n}$ $\quad\sigma = \{s_1\overline{s_1}/s\overline{s}\}$
X	$\{\mathbf{0}\}$

Table 5.8: The set $\widehat{\mathfrak{J}}^{k}_{\tilde{x},\rho}(P)^{d}_{g}$ when $g \neq \emptyset$. (Table 5.9 covers the case $g = \emptyset$.)

Operational Completeness We first consider transitions using the untyped LTS; in Lemma 5.3.8 we will consider transitions with the typed LTS.

Lemma 5.3.7. *Assume $P_1\{\tilde{u}/\tilde{x}\}$ is a process and $P_1\{\tilde{u}/\tilde{x}\}\, \mathcal{S}\, Q_1$.*

P	$\widehat{\mathfrak{J}}^{k}_{\tilde{x},\rho}(P)^{d}_{g}$ when $g = \emptyset$
$n_i!\langle y_j\rangle.Q$	$\{n_l!\langle\widetilde{y}\rho\rangle.c^r_{k+1}!\langle\widetilde{z}\rho\rangle \mid \mu X.c^r_k?(\widetilde{x}).(n_l!\langle\widetilde{y}\rangle.c^r_{k+1}!\langle\widetilde{z}\rangle \mid X) \mid \widehat{\mathcal{A}}^{k+1}_{\tilde{z}}(Q\sigma)_g\}$
$n_i?(y).Q$	$\{n_l?(\widetilde{y}).c^r_{k+1}!\langle\widetilde{z}\rho\rangle \mid \mu X.c^r_k?(\widetilde{x}).(n_l?(\widetilde{y}).c^r_{k+1}!\langle\widetilde{z}\rangle \mid X) \mid \widehat{\mathcal{A}}^{k+1}_{\tilde{z}}(Q\sigma)_g\}$
$Q_1 \mid Q_2$	$\{c^r_{k+1}!\langle\widetilde{y}_1\rho\rangle \mid c^r_{k+l+1}!\langle\widetilde{y}_2\rho\rangle \mid$ $\mu X.c^r_k?(\widetilde{x}).(c^r_{k+1}!\langle\widetilde{y}_1\rangle \mid c^r_{k+l+1}!\langle\widetilde{y}_2\rangle \mid X) \mid$ $\widehat{\mathcal{A}}^{k+1}_{\tilde{y}_1}(Q_1)_{g_1} \mid \widehat{\mathcal{A}}^{k+l+1}_{\tilde{y}_2}(Q_2)_{g_2}\}$
$(\nu\, s : C)\, Q$	$\{c^r_{k+1}!\langle\widetilde{z}\rho\rangle \mid \mu X.(\nu\,\widetilde{s} : \mathcal{G}(C))\, c_k?(\widetilde{x}).(c^r_{k+1}!\langle\widetilde{z}\rangle \mid X) \mid \widehat{\mathcal{A}}^{k+1}_{\tilde{z}}(Q)_g\}$

Table 5.9: The set $\widehat{\mathfrak{J}}^{k}_{\tilde{x},\rho}(P)^{d}_{g}$ when $g = \emptyset$. Side conditions are the same as for corresponding cases from Table 5.8 (when $g \neq \emptyset$).

1. *Whenever* $P_1\{^{\tilde{u}}/_{\tilde{x}}\}\xrightarrow{(\nu\,\widetilde{m_1})\,n!\langle v:C_1\rangle}P_2$, *such that* $\overline{n} \notin P_1\{^{\tilde{u}}/_{\tilde{x}}\}$, *then there exist* Q_2 *and* σ_v *such that* $Q_1\xRightarrow{(\nu\,\widetilde{m_2})\,\breve{n}!\langle\widetilde{v}:\mathcal{H}^*(C_1)\rangle}Q_2$ *where* $v\sigma_v \diamond \widetilde{v}$ *and, for a fresh* t,
$$(\nu\,\widetilde{m_1})(P_2 \parallel t \Leftarrow_{\mathtt{C}} v : C_1) \;\; \mathcal{S} \;\; (\nu\,\widetilde{m_2})(Q_2 \parallel t_1 \Leftarrow_{\mathtt{m}} v\sigma_v : C_1)$$
2. *Whenever* $P_1\{^{\tilde{u}}/_{\tilde{x}}\}\xrightarrow{n?(v)}P_2$, *such that* $\overline{n} \notin P_1\{^{\tilde{u}}/_{\tilde{x}}\}$, *then there exist* Q_2 *and* σ_v *such that* $Q_1\xRightarrow{\breve{n}?(\tilde{v})}Q_2$ *where* $v\sigma_v \diamond \widetilde{v}$ *and* $P_2 \;\mathcal{S}\; Q_2$,
3. *Whenever* $P_1\xrightarrow{\tau}P_2$ *then there exists* Q_2 *such that* $Q_1\xRightarrow{\tau}Q_2$ *and* $P_2 \;\mathcal{S}\; Q_2$.

Proof. By transition induction. See Appendix B.3.6 for details. $\square$

The following statement builds upon the previous one to the case of typed LTS (Section 5.3.7):

Lemma 5.3.8. *Assume* $P_1\{^{\tilde{u}}/_{\tilde{x}}\}$ *is a process and* $P_1\{^{\tilde{u}}/_{\tilde{x}}\}\,\mathcal{S}\,Q_1$.

1. *Whenever* $\Gamma_1; \Delta_1 \vdash P_1 \xrightarrow{(\nu\,\widetilde{m_1})\,n!\langle v:C_1\rangle} \Delta_1'; \Lambda_1' \vdash P_2$ *then there exist* Q_2, Δ_2', *and* σ_v *such that* $\Gamma_2; \Delta_2 \vdash Q_1 \xRightarrow{(\nu\,\widetilde{m_2})\,\breve{n}!\langle\tilde{v}:\mathcal{H}^*(C)\rangle} \Delta_2' \vdash Q_2$ *where* $v\sigma_v \diamond \widetilde{v}$ *and, for a fresh* t,
$$(\nu\,\widetilde{m_1})(P_2 \parallel t \Leftarrow_{\mathtt{C}} v : C_1) \;\; \mathcal{S} \;\; (\nu\,\widetilde{m_2})(Q_2 \parallel t \Leftarrow_{\mathtt{m}} v\sigma_v : C_1)$$
2. *Whenever* $\Gamma_1; \Delta_1 \vdash P_1 \xrightarrow{n?(v)} \Delta_1' \vdash P_2$ *then there exist* Q_2, Δ_2', *and* σ_v *such that* $\Gamma_2; \Delta_2 \vdash Q_1 \xRightarrow{\breve{n}?(\tilde{v})} \Delta_2' \vdash Q_2$ *where* $v\sigma_v \diamond \widetilde{v}$ *and* $P_2 \;\mathcal{S}\; Q_2$,
3. *Whenever* $\Gamma_1; \Delta_1 \vdash P_1 \xrightarrow{\tau} \Delta_1' \vdash P_2$ *then there exist* Q_2 *and* Δ_2' *such that* $\Gamma_2; \Delta_2 \vdash Q_1 \xRightarrow{\tau} \Delta_2' \vdash Q_2$ *and* $P_2 \;\mathcal{S}\; Q_2$.

Proof. The proof uses results of Lemma 5.3.7. We consider the first case, the other two are similar. By the definition of the typed LTS we have:

$$\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{^{\tilde{u}}/_{\tilde{x}}\} \tag{5.1}$$
$$(\Gamma_1; \emptyset; \Delta_1) \xrightarrow{(\nu\,\widetilde{m_1})\,n!\langle v:C_1\rangle} (\Gamma_1; \emptyset; \Delta_2) \tag{5.2}$$

By (5.2) we further have

$$\text{[SSnd]}\ \frac{\begin{array}{ccc}\Gamma,\Gamma';\Lambda';\Delta' \vdash V \triangleright U & \Gamma';\emptyset;\Delta_j \vdash m_j \triangleright U_j & \overline{n} \notin \mathtt{dom}(\Delta) \\ \Delta' \backslash (\cup_j \Delta_j) \subseteq (\Delta, n : S) & \Gamma';\emptyset;\Delta'_j \vdash \overline{m}_j \triangleright U'_j & \Lambda' \subseteq \Lambda\end{array}}{(\Gamma;\Lambda;\Delta, n :!\langle C_1\rangle;S) \xrightarrow{(\nu\,\widetilde{m_1})\,n!\langle v:C_1\rangle} (\Gamma,\Gamma';\Lambda\backslash\Lambda';(\Delta, n : S, \cup_j \Delta'_j)\backslash\Delta')}$$

By (5.1) and the condition $\overline{n} \notin \mathtt{dom}(\Delta)$ we have $\overline{n} \notin \mathtt{fn}(P_1\{\tilde{u}/\tilde{x}\})$. Therefore, we can apply Item 1 of Lemma 5.3.7. □

Operational Soundness For the proof of operational soundness we follow the same strategy as before: we first establish a lemma for the untyped LTS, then we extend it to the case of the typed LTS.

Lemma 5.3.9. *Assume $P_1\{\tilde{u}/\tilde{x}\}$ is a well-formed process and $P_1\{\tilde{u}/\tilde{x}\}\,\mathcal{S}\,Q_1$.*

1. *Whenever $Q_1 \xrightarrow{(\nu\,\widetilde{m_2})\,n_i!\langle \tilde{v}:\mathcal{H}^*(C_1)\rangle} Q_2$, such that $\overline{n}_i \notin \mathtt{fn}(Q_1)$, then there exist P_2 and σ_v such that $P_1\{\tilde{u}/\tilde{x}\} \xrightarrow{(\nu\,\widetilde{m_1})\,n!\langle v:C_1\rangle} P_2$ where $v\sigma_v \diamond \tilde{v}$ where $v\sigma_v \diamond \tilde{v}$ and, for a fresh t,*
$$(\nu\,\widetilde{m_1})(P_2 \parallel t \Leftarrow_{\mathtt{C}} v : C_1)\ \ \mathcal{S}\ \ (\nu\,\widetilde{m_2})(Q_2 \parallel t_1 \Leftarrow_{\mathtt{m}} v\sigma_v : C_1)$$
2. *Whenever $Q_1 \xrightarrow{\breve{n}?(\tilde{v})} Q_2$ then there exist P_2 and σ_v such that $P_1\{\tilde{u}/\tilde{x}\} \xrightarrow{n?(v)} P_2$ where $v\sigma_v \diamond \tilde{v}$ and $P_2\ \mathcal{S}\ Q_2$,*
3. *Whenever $Q_1 \xrightarrow{\tau} Q_2$ either (i) $P_1\{\tilde{u}/\tilde{x}\}\ \mathcal{S}\ Q_2$ or (ii) there exists P_2 such that $P_1 \xrightarrow{\tau} P_2$ and $P_2\ \mathcal{S}\ Q_2$.*

Proof (Sketch). See Appendix B.3.7. □

Lemma 5.3.10. *Assume $P_1\{\tilde{u}/\tilde{x}\}$ is a process and $P_1\{\tilde{u}/\tilde{x}\}\,\mathcal{S}\,Q_1$.*

1. *Whenever $\Gamma_2;\Lambda_2;\Delta_2 \vdash Q_1 \xrightarrow{(\nu\,\widetilde{m_2})\,\breve{n}!\langle \tilde{v}:\mathcal{H}^*(C_1)\rangle} \Lambda'_2;\Delta'_2 \vdash Q_2$ then there exist P_2, σ_v, Δ'_1, and Λ'_1 such that $\Gamma_1;\Lambda_1;\Delta_1 \vdash P_1\{\tilde{u}/\tilde{x}\} \overset{(\nu\,\widetilde{m_1})\,n!\langle v:C_1\rangle}{\Longrightarrow} \Lambda'_1;\Delta'_1 \vdash P_2$ and, for a fresh t,*
$$(\nu\,\widetilde{m_1})(P_2 \parallel t \Leftarrow_{\mathtt{C}} v : C_1)\ \ \mathcal{S}\ \ (\nu\,\widetilde{m_2})(Q_2 \parallel t_1 \Leftarrow_{\mathtt{m}} v\sigma_v : C_1)$$
2. *Whenever $\Gamma_2;\Lambda_2;\Delta_2 \vdash Q_1 \xrightarrow{\breve{n}?(\tilde{v})} \Lambda'_2;\Delta'_2 \vdash Q_2$ then there exist P_2, σ_v, Λ'_1, and Δ'_1 such that $\Gamma_1;\Lambda_1;\Delta_1 \vdash P_1\{\tilde{u}/\tilde{x}\} \overset{n?(v)}{\Longrightarrow} \Lambda'_1,\Delta'_1 \vdash P_2$ where $v\sigma_v \diamond \tilde{v}$ and $P_2\ \mathcal{S}\ Q_2$,*
3. *Whenever $\Gamma_2;\Lambda_2;\Delta_2 \vdash Q_1 \xrightarrow{\tau} \Lambda'_2;\Delta'_2 \vdash Q_2$ either (i) $P_1\{\tilde{u}/\tilde{x}\}\ \mathcal{S}\ Q_2$ or (ii) there exist P_2, Λ'_1, and Δ'_1 such that $\Gamma_1;\Lambda_1;\Delta_1 \vdash P_1\{\tilde{u}/\tilde{x}\} \overset{\tau}{\Longrightarrow} \Lambda'_1;\Delta'_1 \vdash P_2$ and $P_2\ \mathcal{S}\ Q_2$.*

Proof (Sketch). Analogous to the proof of Lemma 5.3.8, using Lemma 5.3.9. □

We close this section by stating again Theorem 5.3.2 and giving its proof.

Theorem 5.3.2 (Operational Correspondence)**.** *Let P be a process such that $\Gamma;\Delta \vdash P$. We have*
$$\Gamma;\Delta \vdash P\ \ \approx^{\mathtt{M}}\ \ \mathcal{H}^*(\Gamma);\mathcal{H}^*(\Delta) \vdash \mathcal{F}^*(P)$$

Proof. By Lemma 5.3.10 and Lemma 5.3.9 we know $\mathcal{S}$ is an MST bisimulation. So, we need to show $(P, \mathcal{F}^*(P)) \in \mathcal{S}$. Let P_1 be such that $P_1\{\tilde{r}/\tilde{x}\} = P$ where $\tilde{r} = \mathtt{rn}(P)$. Further, let $\sigma = \bigcup_{v\in\tilde{r}}\{v_1/v\}$, so we have $\sigma \in \mathsf{index}(\tilde{r})$. Then, let $\widetilde{r_*} = \mathsf{bn}(\tilde{r} : \widetilde{S})$ and $\widetilde{x_*} = \mathsf{bn}(\tilde{x} : \widetilde{S})$ where $\tilde{r} : \widetilde{S}$. Therefore, by Definition 5.3.7 and Table 5.6 we have $\mathcal{F}^*(P) \in \mathcal{C}^{\widetilde{r_*}}_{\widetilde{x_*}}(P_1)$. Finally, by Definition 5.3.28 we have $(P, \mathcal{F}^*(P)) \in \mathcal{S}$. □

5.4 Extension with Labeled Choices

In session-based concurrency, labeled choices are implemented via branching and selection constructs, denoted $u_i \triangleright \{l_j : P_j\}_{j \in I}$ and $u_i \triangleleft l_j.Q$, respectively. Intuitively, a branching construct specifies the offer of a finite number of alternative behaviors $(P_1, P_2, \ldots)$, whereas a selection construct signals the choice of one of them. Put differently, branching and selection implement deterministic choices via the input and output of labels; they come with dedicated session types, denoted $\&\{l_i : S_i\}_{i \in I}$ and $\oplus\{l_i : S_i\}_{i \in I}$, respectively, for some finite index set I.

Here we illustrate how our decomposition strategy can be extended to account for labeled choice (and their types). This entails the extension of the type decomposition function $\mathcal{H}^*(\,\cdot\,)$ (Definition 5.3.1) and of the breakdown function $\mathcal{A}^k_{\tilde{x}}(P)$ (Table 5.4); this latter extension will be denoted $\mathcal{A}^{\boxplus k}_{\tilde{x}}(\,\cdot\,)$. For simplicity, we concentrate on the case of *finite* processes, without recursion / recursive session types. We base our presentation on the corresponding breakdown for the case of the higher-order calculus HO, as presented in [5].

Notice that in a branching type $\&\{l_i : S_i\}_{i \in I}$ each S_i can be different. To uniformly handle these differences, we break down branching and selection types as follows:

Definition 5.4.1 (Decomposing Types: Labeled Choices)**.** We extend the type decomposition function $\mathcal{H}^*(\,\cdot\,)$ (Definition 5.3.1) as follows:

$$\mathcal{H}^*(\&\{l_i : S_i\}_{i \in I}) = \&\{l_i : ?(\mathcal{H}^*(S_i));\mathtt{end}\}_{i \in I}$$
$$\mathcal{H}^*(\oplus\{l_i : S_i\}_{i \in I}) = \oplus\{l_i : !\langle\mathcal{H}^*(S_i)\rangle;\mathtt{end}\}_{i \in I}$$

This decomposition follows the intuition that branching and selection correspond to the input and output of labels, respectively. For example, in the case of branching, once a particular branch l_i has been selected, we would like to input names on which to provide sessions from the branch $\mathcal{H}^*(S_i)$; the corresponding selection process should provide the names for the chosen branch. By binding names of selected branch, we account for the different session types of the branches S_i.

The decomposition of branching and selection processes is defined accordingly:

Definition 5.4.2 (Decomposing Processes: Labeled Choice)**.** Given a $k \geq 0$ and a tuple of names $\tilde{x}$, the decomposition function $\mathcal{A}^{\boxplus k}_{\tilde{x}}(\,\cdot\,)$ is defined as

$$\mathcal{A}^{\boxplus k}_{\tilde{x}}(u_i \triangleright \{l_j : P_j\}_{j \in I}) = c_k?(\tilde{x}).u_i \triangleright \{l_j : (\nu\, \tilde{c}_j)\, u_i?(\tilde{y}).\overline{c_{k+1}}!\langle\tilde{x}\tilde{y}\rangle \mid \mathcal{A}^{\boxplus k+1}_{\tilde{x}\tilde{y}}(P_j\{y_1/u_i\})\}_{j \in I}$$
$$\mathcal{A}^{\boxplus k}_{\tilde{x}}(u_i \triangleleft l_j.Q) = c_k?(\tilde{x}).u_i \triangleleft l_j.(\nu\, \tilde{u} : \mathcal{H}^*(S_j))\, u_i!\langle\tilde{\overline{u}}\rangle.\overline{c_{k+1}}!\langle\tilde{x}\rangle \mid \mathcal{A}^{\boxplus k+1}_{\tilde{x}}(Q\{s_1/u_i\})$$

where $\widetilde{u} = (u_{i+1}, \ldots, u_{i+|\mathcal{H}^*(S_j)|})$ and $\widetilde{\overline{u}} = (\overline{u_{i+1}}, \ldots, \overline{u_{i+|\mathcal{H}^*(S_j)|}})$. For the remaining constructs, $\mathcal{A}^{\boxplus k}_{\tilde{x}}(\,\cdot\,)$ corresponds to $\mathcal{A}^{\boxplus k}_{\tilde{x}}(\,\cdot\,)$ (Table 5.4).

We illustrate the workings of $\mathcal{A}^{\boxplus k}_{\tilde{x}}(\cdot)$ by means of the following example, adapted from [5]:

Example 5.4.1. Consider a mathematical server Q that offers clients two operations: addition and negation of integers. The server uses name u to implement the following session type:

$$S = \&\{\mathsf{add} : \underbrace{?(\mathsf{int});?(\mathsf{int});!\langle\mathsf{int}\rangle;\mathtt{end}}_{S_{\mathsf{add}}}\ ,\ \mathsf{neg} : \underbrace{?(\mathsf{int});!\langle\mathsf{int}\rangle;\mathtt{end}}_{S_{\mathsf{neg}}}\}$$

This way, the add branch receives two integers and sends over their sum; the neg branch has a single input of an integer followed by an output of its negation.

Let us consider a possible implementation for the server Q and for a client R that selects the first branch to add integers 16 and 26:

$$
\begin{aligned}
Q &\triangleq u \triangleright \{\mathsf{add} : \underbrace{u?(a).u?(b).u!\langle a+b\rangle}_{Q_{\mathsf{add}}},\ \mathsf{neg} : \underbrace{u?(a).u!\langle -a\rangle}_{Q_{\mathsf{neg}}}\}\\
R &\triangleq \overline{u} \triangleleft \mathsf{add}.\overline{u}!\langle \mathbf{16}\rangle.\overline{u}!\langle \mathbf{26}\rangle.\overline{u}?(r)
\end{aligned}
$$

The composed process $P \triangleq (\nu\, u)\,(Q \mid R)$ can reduce as follows:

$$
\begin{aligned}
P &\longrightarrow (\nu\, u)\,(u?(a).u?(b).u!\langle a+b\rangle \mid \overline{u}!\langle \mathbf{16}\rangle.\overline{u}!\langle \mathbf{26}\rangle.\overline{u}?(r))\\
&\longrightarrow^2 (\nu\, u)\,(u!\langle \mathbf{16}+\mathbf{26}\rangle \mid \overline{u}?(r)) = P'
\end{aligned}
$$

First, following Definition 5.4.1, the decomposition of S is the minimal session type M, defined as follows:

$$
\begin{aligned}
M = \mathcal{H}^*(S) = \&\{&\mathsf{add} : ?((?(\mathsf{int}), ?(\mathsf{int}), !\langle \mathsf{int}\rangle)),\\
&\mathsf{neg} : ?((?(\mathsf{int}), !\langle \mathsf{int}\rangle))\}
\end{aligned}
$$

Let us now discuss the decomposition of P. Using Definition 5.4.2 we have:

$$
D = (\nu\, c_1, \dots, c_7)\,(c_1?().\overline{c_2}!\langle\rangle.\overline{c_3}!\langle\rangle \mid \mathcal{A}^{\boxplus 2}_{\epsilon}(Q\sigma_2) \mid \mathcal{A}^{\boxplus 3}_{\epsilon}(R\sigma_2)) \tag{5.3}
$$

where $\sigma_2 = \{u_1\overline{u_1}/u\overline{u}\}$. The breakdown of process Q is as follows:

$$
\begin{aligned}
\mathcal{A}^{\boxplus 2}_{\epsilon}(Q\sigma_2) = c_2?().u_1 \triangleright \{&\mathsf{add} : (\nu\, \tilde{c}_j)\, u_1?(y_1, y_2, y_3).\overline{c_1}!\langle y_1, y_2, y_3\rangle \mid \mathcal{A}^{\boxplus 1}_{y_1,y_2,y_3}(Q_{\mathsf{add}}\{y_1/u_1\}),\\
&\mathsf{neg} : (\nu\, \tilde{c}_j)\, u_1?(\tilde{y}).\overline{c_1}!\langle y_1, y_2\rangle \mid \mathcal{A}^{\boxplus 1}_{y_1,y_2}(Q_{\mathsf{neg}}\{y_1/u_1\})\}
\end{aligned}
$$

where

$$
\begin{aligned}
\mathcal{A}^{\boxplus 1}_{y_1,y_2,y_3}(Q_{\mathsf{add}}\{y_1/u_1\}) &= c_1?(y_1, y_2, y_3) \mid c_1?(y_1, y_2, y_3).y_1?(a).\overline{c_2}!\langle y_2, y_3, a\rangle \mid\\
&\quad c_2?(y_2, y_3, a).y_2?(b).\overline{c_3}!\langle y_3, a, b\rangle \mid c_3?(y_3, a, b).y_3!\langle a+b\rangle.c_4?() \mid c_4?()\\
\mathcal{A}^{\boxplus 1}_{y_1,y_2}(Q_{\mathsf{neg}}\{y_1/u_1\}) &= c_1?(y_1, y_2) \mid c_1?(y_1, y_2).y_1?(a).\overline{c_2}!\langle y_2, a\rangle \mid\\
&\quad c_2?(y_2, a).y_2!\langle -a\rangle.\overline{c_3}!\langle\rangle \mid c_3?()
\end{aligned}
$$

In process $\mathcal{A}^{\boxplus 1}_{y_1,y_2,y_3}(Q_{\mathsf{add}}\{y_1/u_1\})$, name u_1 implements M. Following the common trio structure, the first prefix awaits activation on c_2. The next prefix mimics the branching action of Q on u_1. Then, each branch consists of the input of the breakdown of the continuation of channel u, that is y_1, y_2, y_3. This input does not have a counterpart in Q; it is meant to synchronize with process $\mathcal{A}^{\boxplus 3}_{\epsilon}(R\sigma_2)$, the breakdown of the corresponding selection process.

In the bodies of the abstractions we break down Q_{add} and Q_{neg}, but not before adjusting the names on which the broken down processes provide the sessions. For this, we substitute u with y_1 in both processes, ensuring that the broken down names are bound by the input. By binding decomposed names in the input we account for different session types of the original name in branches, while preserving typability: this way the decomposition of different branches can use (i) the same names but typed with different minimal types and (ii) a different number of names, as it is the case in this example.

The decomposition of the client process R, which implements the selection, is as follows:

$$
\mathcal{A}^{\boxplus 3}_{\epsilon}(R\sigma_2) = c_3?(\epsilon).\overline{u}_1 \triangleleft \mathsf{add}.(\nu\, u_2, u_3, u_4)\,\overline{u}_1!\langle u_2, u_3, u_4\rangle.\overline{c_4}!\langle\rangle \mid \mathbf{A}^{\boxplus 4}_{\epsilon}(\overline{u}_2!\langle \mathbf{16}\rangle.\overline{u}_2!\langle \mathbf{26}\rangle.\overline{u}_2?(r))
$$

where:

$$
\mathbf{A}^{\boxplus 4}_{\epsilon}(\overline{u}_2!\langle \mathbf{16}\rangle.\overline{u}_2!\langle \mathbf{26}\rangle.\overline{u}_2?(r)) = c_4?().\overline{u}_2!\langle \mathbf{16}\rangle.\overline{c_5}!\langle\rangle \mid c_5?().\overline{u}_3!\langle \mathbf{26}\rangle.\overline{c_6}!\langle\rangle \mid c_6?().\overline{u}_4?(r).\overline{c_7}!\langle\rangle \mid c_7?()
$$

After receiving the context on c_3 (empty in this case), the selection action on u_1 is mimicked; then, the breakdown of channel continuation, u_2, u_3, u_4, that are locally bound, are sent along name u_1. The intention is to use these names to connect the selected branch and the continuation of a selection process: the subprocess encapsulated in the branch will use (u_2, u_3, u_4), while the dual names $(\overline{u}_2, \overline{u}_3, \overline{u}_4)$ are present in the breakdown of the continuation.

Let us briefly examine the reductions of the decomposed process D in (5.3). First, c_1, c_2, and c_3 will synchronize. We have $D \longrightarrow^4 D_1$, where

$$\begin{aligned} D_1 = (\nu\, c_4 \ldots c_7)\,(\nu\, u_1)\,(u_1 \triangleright \{&\mathsf{add} : (\nu\, \tilde{c}_j)\, u_1?(y_1, y_2, y_3).\overline{c_1}!\langle \tilde{y}\rangle \mid \mathcal{A}^{\boxplus 1}_{y_1,y_2,y_3}(Q_{\mathsf{add}}\{y_1/u_1\}), \\ &\mathsf{neg} : (\nu\, \tilde{c}_j)\, u_1?(\tilde{y}).\overline{c_1}!\langle y_1, y_2\rangle \mid \mathcal{A}^{\boxplus 1}_{y_1,y_2}(Q_{\mathsf{neg}}\{y_1/u_1\})\} \\ &\mid \overline{u}_1 \triangleleft \mathsf{add}.(\nu\, u_2, u_3, u_4)\, \overline{u}_1!\langle u_2, u_3, u_4\rangle.\overline{c_4}!\langle\rangle \mid \mathbf{A}^{\boxplus 4}_{\epsilon}(\overline{u}_2!\langle \mathbf{16}\rangle.\overline{u}_2!\langle \mathbf{26}\rangle.\overline{u}_2?(r))) \end{aligned}$$

Now, the processes chooses the label add on u_1. The process D_1 will reduce further as $D_1 \longrightarrow D_2 \longrightarrow^2 D_3$, where:

$$\begin{aligned} D_2 = \;& (\nu\, \tilde{c}_j)\, u_1?(y_1, y_2, y_3).\overline{c_1}!\langle \tilde{y}\rangle \mid \mathcal{A}^{\boxplus 1}_{y_1,y_2,y_3}(Q_{\mathsf{add}}\{y_1/u_1\}) \mid \\ & (\nu\, u_2, u_3, u_4)\, \overline{u}_1!\langle u_2, u_3, u_4\rangle.\overline{c_4}!\langle\rangle \mid \mathbf{A}^{\boxplus 4}_{\epsilon}(\overline{u}_2!\langle \mathbf{16}\rangle.\overline{u}_2!\langle \mathbf{26}\rangle.\overline{u}_2?(r)) \\ D_3 = \;& \overline{u}_2?(a).\overline{c_2}!\langle u_3, u_4, a\rangle \mid c_2?(y_2, y_3, a).y_2?(b).\overline{c_3}!\langle y_3, a, b\rangle \mid \\ & c_3?(y_3, a, b).y_3!\langle a + b\rangle.\overline{c_4}!\langle\rangle \mid \\ & c_4?().\overline{u}_2!\langle \mathbf{16}\rangle.\overline{c_5}!\langle\rangle \mid c_5?().\overline{u}_3!\langle \mathbf{26}\rangle.\overline{c_6}!\langle\rangle \mid c_6?().\overline{u}_4?(r).\overline{c_7}!\langle\rangle \mid c_7?() \end{aligned}$$

Now, process D_3 can mimic the original transmission of the integer 16 on channel u_2 as follows:

$$\begin{aligned} D_3 \longrightarrow\; & \overline{c_2}!\langle \overline{u}_3, \overline{u}_4, \mathbf{16}\rangle \mid c_2?(y_2, y_3, a).y_2?(b).\overline{c_3}!\langle y_3, a, b\rangle \mid \\ & c_3?(y_3, a, b).y_3!\langle a + b\rangle.\overline{c_4}!\langle\rangle \mid c_4?() \mid \\ & \overline{c_5}!\langle\rangle \mid c_5?().\overline{u}_3!\langle \mathbf{26}\rangle.\overline{c_6}!\langle\rangle \mid c_6?().\overline{u}_4?(r).\overline{c_7}!\langle\rangle \mid c_7?() = D_4 \end{aligned}$$

Finally, D_4 reduces to D_5 as follows:

$$D_4 \longrightarrow^5 \overline{u}_4!\langle a + b\rangle.\overline{c_4}!\langle\rangle \mid c_4?() \mid u_4?(r).\overline{c_7}!\langle\rangle \mid c_7?() = D_5$$

This way, the steps from D to D_5 attest to the fact that our extended decomposition correctly simulates the steps from P to P'.

5.5 Related Work

A source of inspiration for our developments is the trios decomposition by Parrow [50], which he studied for an untyped π-calculus with replication; in contrast, π processes are typed and feature recursion. We stress that our goal is to clarify the role of sequentiality in session types by using processes with MSTs, which lack sequentiality. While Parrow's approach elegantly induces processes typable with MSTs (and suggests a clean approach to establish dynamic correctness), defining trios decompositions for π is just one path towards our goal.

Our work aims to understand session types in terms of themselves, by considering to the sub-class of session types without sequentiality, as defined by MSTs. Prior works have related session types with *different* type systems—see, e.g., [37, 15, 17, 20, 27]. Kobayashi [37] was the first to define a formal relationship between session types and *usage types*, expressed as typed encodings of processes; this relationship was thoroughly studied by Dardha et al. [15, 17, 14] (see below). Demangeon and Honda [20] connect a linearly-typed π-calculus with subtyping and a session-typed calculus via a full abstraction result. Both works rely on convenient constructs in the target language (e.g., case constructors and variant types) to achieve correct encodability. The work of Gay et al. [27] addresses a similar problem but by adopting a π-calculus without such additional features: they consider the correct encodability of session types into a generic type system for a simple π-calculus. Their work demonstrates that the translation of session types for branching and selection in the presence of subtyping is a challenging endeavor.

The work by Dardha et al. [15, 17, 14] develops further the translation first suggested by Kobayashi. They compile a session π-calculus down into a π-calculus with the linear type system of [38] extended with variant types. They represent sequentiality using a continuation-passing style (CPS): a session type is interpreted as a linear type carrying a pair consisting of the original payload type and a new linear channel type, to be used for ensuing interactions. The differences between this CPS approach and our work and are also technical: the approach in [15] thus involves translations connecting *two* different π-calculi and *two* different type systems. In contrast, our approach based on MSTs justifies sequentiality using a single typed process framework. Another difference concerns process recursion and recursive session types: while the works [15, 17] consider only finite processes (no recursion nor recursive types), the work [14] considers recursive session types as a way of supporting replicated processes (a specific class of unrestricted behaviors). In contrast, the decompositions we have considered here support full process recursion.

Despite these concrete differences, it is interesting to compare our approach and the CPS approach. To substantiate our comparisons, we introduce some selected notions for the linearly-typed π-calculus considered by Dardha et al. (the reader is referred to [17] for a thorough description and technical results).

Definition 5.5.1 (Linearly-Typed π-calculus Processes [17])**.** The syntax of processes $P, Q, \ldots$, values $v, v', \ldots$, and of linear types $\tau, \tau', \ldots$ is as follows:

$$\begin{array}{rl}
\text{Processes} & P, Q, := x!\langle \tilde{v} \rangle.P \;\mid\; x?(\tilde{y}).P \;\mid\; (\nu\, v)\, P \;\mid\; \mathbf{0} \;\mid\; P \mid Q \;\mid\; \textbf{case}\ v\ \textbf{of}\ \{l_i\_(x_i) \triangleright P_i\}_{i\in I} \\
\text{Values} & v, v' := x \;\mid\; \star \;\mid\; l\_v \\
\text{Types} & \tau, \tau' := l_{\mathsf{o}}[\widetilde{\tau}] \;\mid\; l_{\mathtt{i}}[\widetilde{\tau}] \;\mid\; l_{\#}[\widetilde{\tau}] \;\mid\; \varnothing[] \;\mid\; \#[\widetilde{\tau}] \;\mid\; \langle l_i : \tau_i \rangle_{i\in I} \;\mid\; \mathtt{Unit}
\end{array}$$

At the level of processes, the main difference with respect to the syntax of $\mathsf{HO}\pi$ in Figure 2.1 is the case construct '**case** v **of** $\{l_i\_(x_i) \triangleright P_i\}_{i\in I}$', which together with the variant value 'l_v' implement a form of deterministic choice. This choice is decoupled from a synchronization: indeed, the process **case** l_j_v **of** $\{l_i\_(x_i) \triangleright P_i\}_{i\in I}$ (with $j \in I$) autonomously reduces to $P_j\{v/x_j\}$. At the level of types, the grammar above first defines types $l_{\mathsf{o}}[\widetilde{\tau}]$, $l_{\mathtt{i}}[\widetilde{\tau}]$, and $l_{\#}[\widetilde{\tau}]$ for the linear exchange messages of type $\widetilde{\tau}$: output, input, and both output and input, respectively. Then, the types $\varnothing[]$ and $\#[\widetilde{\tau}]$ are assigned to channels without any capabilities and with arbitrary capabilities, respectively. Finally, we have the variant type $\langle l_i : \tau_i \rangle_{i\in I}$ associated to the case construct and the unit type $\mathtt{Unit}$.

As already mentioned, the key idea of the CPS approach is to represent one message exchange using *two* channels: one is the message itself, the other is a continuation for the next sequential action in the session. We illustrate this approach by example.

Example 5.5.1 (The CPS Approach, By Example)**.** Consider the session type S and the processes Q and R from Example 5.4.1. Under the CPS approach, the session type S is represented as linear types as follows:

$$\begin{array}{ll}
[\![S]\!] = l_{\mathtt{i}}[\langle \mathsf{add} : [\![S_{\mathsf{add}}]\!],\ \mathsf{neg} : [\![S_{\mathsf{neg}}]\!] \rangle] & [\![S_{\mathsf{add}}]\!] = l_{\mathtt{i}}[\mathsf{int}, l_{\mathtt{i}}[\mathsf{int}, l_{\mathsf{o}}[\mathsf{int}, \varnothing[]]]] \\
 & [\![S_{\mathsf{neg}}]\!] = l_{\mathtt{i}}[\mathsf{int}, l_{\mathsf{o}}[\mathsf{int}, \varnothing[]]]
\end{array}$$

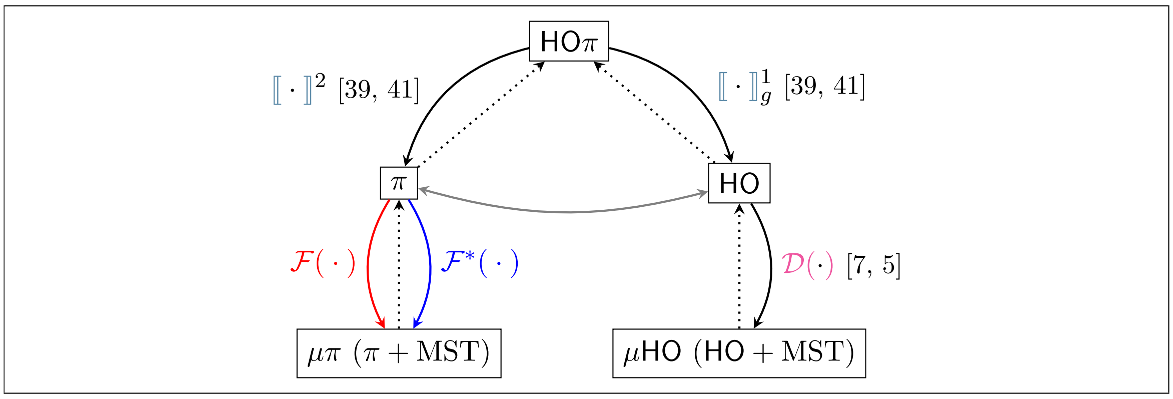


Figure 5.7: Summary of expressiveness results for HOπ. Solid lines indicate correct encodings and decompositions; dotted lines indicate sub-languages.

This way, sequentiality in S arises in $[\![S]\!]$ as nesting of pairs—the later that an action appears in a session type, the deeper it will appear in the corresponding linear type. Accordingly, the encoding of session-typed processes as linear processes, denoted $[\![\cdot]\!]_f$, is illustrated below; the parameter f records continuations:

$$\begin{aligned}
[\![P]\!]_\emptyset &= (\nu\, u)\, [\![Q \mid R]\!]_\emptyset = (\nu\, u)\, [\![Q]\!]_\emptyset \mid [\![R]\!]_\emptyset \\
[\![Q]\!]_\emptyset &= u?(y).\mathbf{case}\ y\ \mathbf{of}\ \{\mathsf{add\_}c_1 \triangleright [\![Q_{\mathsf{add}}]\!]_{u\mapsto c_1},\ \mathsf{neg\_}c_1 \triangleright [\![Q_{\mathsf{neg}}]\!]_{u\mapsto c_1}\} \\
[\![Q_{\mathsf{add}}]\!]_{u\mapsto c_1} &= c_1?(a, c_2).c_2?(b, c_3).(\nu\, c_4)\, c_3!\langle a+b, c_4\rangle.\mathbf{0} \\
[\![Q_{\mathsf{neg}}]\!]_{u\mapsto c_1} &= c_1?(a, c_2).(\nu\, c_3)\, c_2!\langle -a, c_3\rangle.\mathbf{0} \\
[\![R]\!]_\emptyset &= (\nu\, c_1)\, u!\langle \mathsf{add\_}c_1\rangle.(\nu\, c_2)\, c_1!\langle \mathbf{16}, c_2\rangle.(\nu\, c_3)\, c_2!\langle \mathbf{26}, c_3\rangle.c_3?(r).\mathbf{0}
\end{aligned}$$

Observe how the branching construct in Q is encoded using two constructs: an input prefix immediately followed by a case construct. Also, notice the use of restrictions $(\nu\, c_i)$ (with $i \in \{1, 2, 3\}$) in the encoding of output prefixes: this is crucial to avoid interferences and ensure that session communications are mimicked in the intended order.

Based on the above example, we may identify two sources of comparison between our decompositions into MSTs and the CPS approach. At the level of types, there is clear resemblance between MSTs and the class of linear types needed to encode finite session types in the CPS approach. At the level of processes, both approaches use a similar principle, namely to generate fresh names to encode the sequential structure of sessions. While the CPS approach follows a *dynamic* discipline to generate such names (i.e., they are generated by the encoding of output-like actions), the decomposition approach follows a *static discipline*, i.e., fresh names are generated based on the length of the source session types.

5.6 Concluding Remarks

We studied minimal session types (MSTs) for π—the sub-language of HOπ [39, 41] with first-order communication, recursion, and recursive types—and obtained new minimality results based on two different decompositions of processes. Introduced in [7], MSTs express a specific form of minimality, which eschews from sequentiality in types; hence, our minimality results for π mean that sequentiality in types is a convenient and yet not indispensable feature, as it can be represented by name-passing processes while remaining in a session-typed setting.

Following the approach for HO Chapter 4, which is inspired by Parrow's trios processes [50], we defined two process decompositions for π. They use type information to transform processes typable with standard session types into processes typable with MSTs. The first decomposition, denoted $\mathcal{F}(\,\cdot\,)$, is obtained by composing existing encodability results and the decomposition / minimality result for HO; the second decomposition, denoted $\mathcal{F}^*(\,\cdot\,)$, optimizes the first one by (i) removing redundant synchronizations and (ii) using the native support of recursion in π. For both decompositions we establish the minimality result (cf. Theorem 5.2.1 and Theorem 5.3.1), which is actually a result of static correctness (i.e., preservation of typability). The gains in moving from $\mathcal{F}(\,\cdot\,)$ to $\mathcal{F}^*(\,\cdot\,)$ can be accounted for in very precise terms (Lemma 5.3.1). For the optimized decomposition $\mathcal{F}^*(\,\cdot\,)$ we proved also a dynamic correctness result (Theorem 5.3.2), which formally attests that a process and its decomposition are behaviorally equivalent. This result thus encompasses a result of operational correspondence; its proof leverages existing characterizations of contextual equivalence for π, and adapts them to the case of MSTs. This way, our technical results together confirm the significance of MSTs; they also indicate that a minimality result is independent from the kind of communicated objects, either abstractions (functions from names to processes, as studied in [7, 5]) or names (as studied here).

Sequentiality is the key distinguishing feature in the specification of message-passing programs using session types. Our minimality results for π and HO should not be interpreted as meaning that sequentiality in session types is redundant in *modeling and specifying* processes; rather, we claim that it is not an essential notion to *verifying* them. Because we can type-check session typed processes using type systems that do not directly support sequentiality in types, our decompositions suggest a technique for implementing session types into languages whose type systems do not support sequentiality.

All in all, besides settling a question left open in Chapter 4, our work deepens our understanding about the essential mechanisms in session-based concurrency and about the connection between the first-order and higher-order paradigms in the typed setting. Figure 5.7 depicts the formal connections between $\mathsf{HO}\pi$ and its sub-languages, based on: (i) the mutual encodability results between π and HO [39, 41]; (ii) the minimality result for HO [7, 5]; and (iii) the decompositions and minimality results for π obtained here.

There are several interesting items for future work. First, it would be interesting to consider asynchronous communication and subtyping in the context of our decompositions, both first-order and higher-order. Second, it would be insightful to formulate a "hybrid" approach that combines and unifies the CPS approach [37, 15, 17] and our approach based on MSTs. Strengths of our approach include direct support for recursion and recursive types, and dynamic correctness guarantees given in terms of well-studied behavioral equivalences; on the other hand, the CPS approach offers a simple alternative to represent non-uniform session structures, such as those present in labeled choices.

6 Concluding Remarks

Conclusions

In this thesis we studied minimal formulations for typestates and session types, two important classes of formal structures used in the specification and analysis of temporal properties in programs. Our study aimed at establishing to what extent these structures admit simpler (or even minimal) formulations, and what are their associated benefits and ramifications.

As mentioned in the introduction, we view typestates and session types as the same concept but applied to two different settings: namely, object-oriented languages and communication-based software. In fact, they share common traits, which exhibit different characteristics in those two contexts. In particular, in this thesis we focused on *sequencing*—the essential feature for both structures— and its manifestations in those two settings.

In the case of session types, sequencing is explicitly used as a native construct. We have argued that sequencing represents a burden for the practice of session types, as it is not commonly supported in typing systems of mainstream programming languages. Still, sequencing information is essential in describing and enforcing communication protocols (as we have seen in our e-commerce example in the introduction). Therefore, it very important to retain the ability of expressing sequencing. Our insight is that for accomplishing this goal we can leverage the sequencing features already present in the underlying programming models. In our study, we rely on the π-calculus (in both first-order and higher-order paradigms), a formalism that allows us to rigorously reason about the properties of our developments. Our approach is also justified by the fact that sequencing is usually supported by programming models of many mainstream languages (i.e. in imperative languages).

This way, in the case of session types we accomplish our aim of identifying a minimal formulation by codifying sequencing information onto the process level. We indeed consider our formulation as *minimal*, in the sense that MSTs have features that are, arguably, truly indispensable. That is, MSTs prescribe a single communication action by a payload type and channel direction. It is hard to imagine a useful formulation that is even simpler (has less constructs) than MSTs.

Sequencing is also one of the essential features of typestates; indeed, sequencing is implicitly given in DFA-based approach. Interestingly, sequencing is also the feature that represent a burden in the case of typestates analysis, albeit for different reasons. Roughly speaking, a DFA is able to precisely describe a set of valid sequences. That is, a validity of the next label (i.e., method call) can depend on the entire prior sequence: loosely speaking, any permutation of prior sequences can impact the validity of the next method call (as a DFA state is determined by the exact sequence that leads to it from the initial state). However, in many common code contracts this precise description of valid sequences is not needed; in fact, it is arguably too low-level and as such it can cause a hurdle in practical specifications. Our main insight is that many representative code contracts are mostly about temporal (or causal) relationships between pairs of methods. Based on this insight, we provide a high-level language to specify dependencies that are local to pairs of methods.

While for session types it is clear that our proposed formulations are indeed minimal, in the case of BFA it is not so evident why and whether our formalisms are minimal. We argue that some form of sequencing must be present in a minimal formulation of typestates. In our developments, this is accomplished by specifying only local method dependencies which express temporal relationships, as just discussed. Even though different/alternative sub-classes of DFA-based typestates are conceivable, our developments based on a bit-vector representation provide a solid argument in favor of the minimality of BFAs, as bit-vectors arguably represent the least amount of information and operations needed for analysis.

In summary, our central finding is that sequencing, which is the essential feature of in specifying temporal properties, can be dispensed with by different means in two settings. Crucially, this does not mean that we remove sequencing from specifications altogether. Rather, both of our approaches preserve sequencing, although in different ways. In our BFA approach we are able to specify sequencing by using our high-level language, thereby relieving programmers from producing low-level specifications, which can be tedious or even unfeasible for many typical code contracts. Put differently, our developments enable us to succinctly describe a *family of sequences*, whereas in DFA-based typestates each sequence must be specified explicitly. In the case of session types, we retain full expressivity, as our confirmed by our technical results (static and dynamic correctness); indeed, our decomposition can be used to transform a source program into program that does not require sequencing for type-checking.

Future Work

Our developments enable many interesting strands for future investigation. We discuss some of them.

For our work on BFA-based typestates, it would be relevant to investigate how to "safely" expand on the expressivity of our annotation language without compromising the performance/efficiency of the compositional algorithm and subtyping checking. Put differently, it would useful to determine whether it is feasible to increase the expressivity of BFAs, and for what cost in terms of performance and usability. Moreover, it would be interesting to explore whether our BFA formalism can be effectively used in settings where BFA-based methods are typically used, such as, for example, automata learning, code synthesis, and automatic program repair. Furthermore, it is worth investigating how BFA and DFA-based analyses can be bundled into a single analysis, thus inhering the benefits of both. Finally, we plan to integrate aliasing in our approach, leveraging the fact that Infer already comes with aliasing checkers.

Concerning session types, as mentioned at the end of Chapter 4, we developed a prototype implementation of our decomposition technique in a tool called MISTY [6]. This tool can be used to demonstrate the workings of the encodings: given a source program typable with session types it produces its corresponding decomposition and reduction sequences, which demonstrate tight connection between the two. In future work, we would like to build upon MISTY in order to integrate our decomposition techniques in into mainstream programming languages such as Go and Akka Scala. Going further in this direction, we would like to assess the implementations of our techniques. Finally, from the theoretical side, it would be worth investigating what would minimal formulations mean in the case of *asynchronous* session types, i.e., communication in which output is a non-blocking operation. This can be particularly a interesting research avenue, as asynchronous session types impose a "looser" form of sequencing.

Bibliography

[1] Infer TOPL. `https://fbinfer.com/`, 2021.

[2] Infer TOPL. `https://fbinfer.com/docs/checker-topl/`, 2021.

[3] Rail model. `https://web.dev/rail/`, 2021. Accessed: 2021-09-30.

[4] A. Arslanagic, A. Palamariuc, and J. A. Pérez. Minimal session types for the π-calculus. In N. Veltri, N. Benton, and S. Ghilezan, editors, *PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6-8, 2021*, pages 12:1–12:15. ACM, 2021.

[5] A. Arslanagic, J. A. Pérez, and D. Frumin. A minimal formulation of session types. *CoRR*, abs/2301.05301, 2023.

[6] A. Arslanagic, J. A. Pérez, and E. Voogd. Minimal session types (artifact). *Dagstuhl Artifacts Ser.*, 5(2):05:1–05:3, 2019.

[7] A. Arslanagic, J. A. Pérez, and E. Voogd. Minimal session types (pearl). In A. F. Donaldson, editor, *33rd European Conference on Object-Oriented Programming, ECOOP 2019, July 15-19, 2019, London, United Kingdom*, volume 134 of *LIPIcs*, pages 23:1–23:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[8] A. Arslanagić, P. Subotić, and J. A. Pérez. Lfa checker: Scalable typestate analysis for low-latency environments. Mar 2022.

[9] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM SIGPLAN Notices*, 49, 06 2014.

[10] K. Bierhoff and J. Aldrich. Modular Typestate Checking of Aliased Objects. In *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications*, OOPSLA '07, pages 301–320, New York, NY, USA, 2007. Association for Computing Machinery.

[11] E. Bodden and L. Hendren. The Clara framework for Hybrid Typestate Analysis. *Int. J. Softw. Tools Technol. Transf.*, 14(3):307–326, jun 2012.

[12] C. Calcagno and D. Distefano. Infer: An Automatic Program Verifier for Memory Safety of C Programs. In M. Bobaru, K. Havelund, G. J. Holzmann, and R. Joshi, editors, *NASA Formal Methods*, pages 459–465, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[13] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The astreé analyzer. In M. Sagiv, editor, *Programming Languages and Systems*, pages 21–30, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[14] O. Dardha. Recursive session types revisited. In M. Carbone, editor, *Proceedings Third Workshop on Behavioural Types, BEAT 2014, Rome, Italy, 1st September 2014*, volume 162 of *EPTCS*, pages 27–34, 2014.

[15] O. Dardha, E. Giachino, and D. Sangiorgi. Session types revisited. In *Proc. of PPDP 2012*, pages 139–150. ACM, 2012.

[16] O. Dardha, E. Giachino, and D. Sangiorgi. Session types revisited. *Inf. Comput.*, 256:253–286, 2017.

[17] O. Dardha, E. Giachino, and D. Sangiorgi. Session Types Revisited. *Information and Computation*, 256:253 – 286, Oct. 2017.

[18] R. Deline and M. Fähndrich. The Fugue protocol checker: Is your software Baroque? Technical Report MSR-TR-2004-07, Microsoft Research, 04 2004.

[19] R. DeLine and M. Fähndrich. Typestates for objects. In M. Odersky, editor, *ECOOP 2004 - Object-Oriented Programming, 18th European Conference, Oslo, Norway, June 14-18, 2004, Proceedings*, volume 3086 of *Lecture Notes in Computer Science*, pages 465–490. Springer, 2004.

[20] R. Demangeon and K. Honda. Full abstraction in a subtyped pi-calculus with linear types. In *Proc. of CONCUR 2011*, volume 6901 of *LNCS*, pages 280–296. Springer, 2011.

[21] M. Dezani-Ciancaglini and U. de' Liguoro. Sessions and Session Types: an Overview. In *WS-FM'09*, volume 6194 of *LNCS*, pages 1–28. Springer, 2010.

[22] M. Emmi, L. Hadarean, R. Jhala, L. Pike, N. Rosner, M. Schäf, A. Sengupta, and W. Visser. RAPID: Checking API Usage for the Cloud in the Cloud. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, pages 1416–1426, New York, NY, USA, 2021. Association for Computing Machinery.

[23] M. Fahndrich and R. Leino. Heap Monotonic Typestate. In *Proceedings of the first International Workshop on Alias Confinement and Ownership (IWACO)*, July 2003.

[24] M. Fähndrich and F. Logozzo. Static Contract Checking with Abstract Interpretation. In *Proceedings of the 2010 International Conference on Formal Verification of Object-Oriented Software*, FoVeOOS'10, pages 10–30, Berlin, Heidelberg, 2010. Springer-Verlag.

[25] M. Felderer, D. Gurov, M. Huisman, B. Lisper, and R. Schlick. Formal methods in industrial practice - bridging the gap (track summary). In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV*, volume 11247 of *Lecture Notes in Computer Science*, pages 77–81. Springer, 2018.

[26] C. Flanagan, K. R. M. Leino, M. Lillibridge, G. Nelson, J. B. Saxe, and R. Stata. Extended static checking for java. *SIGPLAN Not.*, 37(5):234–245, may 2002.

[27] S. J. Gay, N. Gesbert, and A. Ravara. Session types as generic process types. In J. Borgström and S. Crafa, editors, *Proceedings Combined 21st International Workshop on Expressiveness in Concurrency and 11th Workshop on Structural Operational Semantics, EXPRESS 2014, and 11th Workshop on Structural Operational Semantics, SOS 2014, Rome, Italy, 1st September 2014.*, volume 160 of *EPTCS*, pages 94–110, 2014.

[28] S. J. Gay, P. Thiemann, and V. T. Vasconcelos. Duality of session types: The final cut. In S. Balzer and L. Padovani, editors, *Proceedings of the 12th International Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric*

Software, PLACES@ETAPS 2020, Dublin, Ireland, 26th April 2020, volume 314 of *EPTCS*, pages 23–33, 2020.

[29] D. Gorla. Towards a unified approach to encodability and separation results for process calculi. *Inf. Comput.*, 208(9):1031–1053, 2010.

[30] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.

[31] K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM, 2008.

[32] H. Hüttel, I. Lanese, V. T. Vasconcelos, L. Caires, M. Carbone, P. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, H. T. Vieira, and G. Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3, 2016.

[33] J. Jacobs. A self-dual distillation of session types. In K. Ali and J. Vitek, editors, *36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany*, volume 222 of *LIPIcs*, pages 23:1–23:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[34] M. Jakobsen, A. Ravier, and O. Dardha. Papaya: Global Typestate Analysis of Aliased Objects. In *23rd International Symposium on Principles and Practice of Declarative Programming*, PPDP 2021, New York, NY, USA, 2021. Association for Computing Machinery.

[35] M. Kellogg, M. Ran, M. Sridharan, M. Schäf, and M. D. Ernst. Verifying Object Construction. In *ICSE 2020, Proceedings of the 42nd International Conference on Software Engineering*, Seoul, Korea, May 2020.

[36] U. Khedker, A. Sanyal, and B. Sathe. *Data Flow Analysis: Theory and Practice.* CRC Press, 2017.

[37] N. Kobayashi. Type systems for concurrent programs. In *Formal Methods at the Crossroads*, volume 2757 of *LNCS*, pages 439–453. Springer, 2003.

[38] N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the Pi-Calculus. *TOPLAS*, 21(5):914–947, Sept. 1999.

[39] D. Kouzapas, J. A. Pérez, and N. Yoshida. On the relative expressiveness of higher-order session processes. In P. Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 446–475. Springer, 2016. Extended version to appear in Information and Computation (Elsevier).

[40] D. Kouzapas, J. A. Pérez, and N. Yoshida. Characteristic bisimulation for higher-order session processes. *Acta Inf.*, 54(3):271–341, 2017.

[41] D. Kouzapas, J. A. Pérez, and N. Yoshida. On the relative expressiveness of higher-order session processes. *Inf. Comput.*, 268, 2019.

[42] D. Kouzapas and N. Yoshida. Globally governed session semantics. *LMCS*, 10(4), 2014.

[43] D. Kouzapas, N. Yoshida, R. Hu, and K. Honda. On asynchronous eventful session semantics. *MSCS*, 2015.

[44] P. Lam, V. Kuncak, and M. Rinard. Generalized Typestate Checking Using Set Interfaces and Pluggable Analyses. *SIGPLAN Not.*, 39(3):46–55, Mar. 2004.

[45] J. Lange, N. Ng, B. Toninho, and N. Yoshida. A static verification framework for message passing in go using behavioural types. In M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, editors, *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 1137–1148. ACM, 2018.

[46] J. Lerch, J. Späth, E. Bodden, and M. Mezini. Access-path abstraction: Scaling field-sensitive data-flow analysis with unbounded access paths. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE '15, page 619–629. IEEE Press, 2015.

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

[48] J. Mota, M. Giunti, and A. Ravara. Java typestate checker. In F. Damiani and O. Dardha, editors, *Coordination Models and Languages*, pages 121–133, Cham, 2021. Springer International Publishing.

[49] N. Ng and N. Yoshida. Static deadlock detection for concurrent go by global session graph synthesis. In A. Zaks and M. V. Hermenegildo, editors, *Proceedings of the 25th International Conference on Compiler Construction, CC 2016, Barcelona, Spain, March 12-18, 2016*, pages 174–184. ACM, 2016.

[50] J. Parrow. Trios in concert. In G. D. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 623–638. The MIT Press, 2000. Online version, dated July 22, 1996, available at `http://user.it.uu.se/~joachim/trios.pdf`.

[51] R. Paul, A. K. Turzo, and A. Bosu. Why Security Defects Go Unnoticed during Code Reviews? A Case-Control Study of the Chromium OS project. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 1373–1385. IEEE, 2021.

[52] T. Reps, S. Horwitz, and M. Sagiv. Precise Interprocedural Dataflow Analysis via Graph Reachability. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, pages 49–61, New York, NY, USA, 1995. Association for Computing Machinery.

[53] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher Order Paradigms.* PhD thesis, University of Edinburgh, 1992.

[54] J. Späth, K. Ali, and E. Bodden. Context-, flow-, and field-sensitive data-flow analysis using synchronized pushdown systems. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019.

[55] R. E. Strom and S. Yemini. Typestate: A Programming Language Concept for Enhancing Software Reliability. *IEEE Trans. Software Eng.*, 12(1):157–171, 1986.

[56] P. Subotić, L. Milikić, and M. Stojić. A Static analysis Framework for Data Science Notebooks. In *ICSE'22: The 44th International Conference on Software Engineering*, May 21-May 29 2022.

[57] T. Szabó, S. Erdweg, and M. Voelter. IncA: A DSL for the Definition of Incremental Program Analyses. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE 2016, pages 320–331, New York, NY, USA, 2016. Association for Computing Machinery.

[58] P. Thiemann and V. T. Vasconcelos. Context-free session types. In J. Garrigue, G. Keller, and E. Sumii, editors, *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, pages 462–475. ACM, 2016.

[59] P. Wadler. Propositions as sessions. In P. Thiemann and R. B. Findler, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, pages 273–286. ACM, 2012.

[60] E. Yahav and S. Fink. *The SAFE Experience*, pages 17–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

A Appendix to Chapter 4

A.1 Appendix to Section 4.3.4

We use the following auxiliary lemma:

Lemma A.1.1. *Let $\widetilde{z}$ be tuple of channel names, U a higher-order type, and S a recursive session type. If $\widetilde{z} : \mathcal{R}^{\star}(!\langle U\rangle;S)$ and $k = [!\langle U\rangle;S\rangle$ then $z_k : \mu\mathsf{t}.!\langle \mathcal{G}(U)\rangle;\mathsf{t}$.*

A.1.1 Proof of Lemma 4.3.1

Lemma 4.3.1. *Let P be an indexed* HO *process and V be a value.*

1. *If $\Gamma;\Lambda;\Delta \circ \Delta_\mu \vdash P \triangleright \diamond$ then $\mathcal{G}(\Gamma_1),\Phi;\emptyset;\mathcal{G}(\Delta),\Theta \vdash \mathcal{B}^k_{\widetilde{x}}(P) \triangleright \diamond$, where:*
   - $k > 0$
   - $\widetilde{r} = dom(\Delta_\mu)$
   - $\Phi = \prod_{r\in\widetilde{r}} c^r : \langle \mathcal{R}^{\star}(\Delta_\mu(r)) \multimap \diamond\rangle$
   - $\widetilde{x} = \mathtt{fv}(P)$
   - $\Gamma_1 = \Gamma \setminus \widetilde{x}$
   - $dom(\Theta) = \{c_k,\dots,c_{k+\wr P\wr-1}\} \cup \{\overline{c_{k+1}},\dots,\overline{c_{k+\wr P\wr-1}}\}$
   - $\Theta(c_k) = ?(U_1,\dots,U_n)$*, where* $(\mathcal{G}(\Gamma),\mathcal{G}(\Lambda))(\widetilde{x}) = (x_1 : U_1,\dots,x_n : U_n)$
   - $\mathsf{balanced}(\Theta)$
2. *If $\Gamma;\Lambda;\Delta \circ \Delta_\mu \vdash V \triangleright \widetilde{T} \multimap \diamond$ then $\mathcal{G}(\Gamma),\Phi;\mathcal{G}(\Lambda);\mathcal{G}(\Delta) \vdash \mathcal{V}_{\widetilde{x}}(V) \triangleright \mathcal{G}(\widetilde{T}) \multimap \diamond$, where:*
   - $\widetilde{x} = \mathtt{fv}(V)$
   - $\Phi = \prod_{r\in\widetilde{r}} c^r : \langle \mathcal{R}^{\star}(\Delta_\mu(r)) \multimap \diamond\rangle$

Proof. By mutual induction on the structure of P and V. Here, we analyze only Part (1) of the theorem, as Part (2) and Part (3) are proven similarly:

1. By assumption $\Gamma;\Lambda;\Delta,\Delta_\mu \vdash P \triangleright \diamond$. There are four cases, depending on the shape of P. We consider two representative cases. We omit other cases as they are similar.
   a) Case $P = \mathbf{0}$. The only rule that can be applied here is Nil. By inversion of this rule, we have: $\Gamma;\emptyset;\emptyset \vdash \mathbf{0}$. We shall then prove the following judgment:

   $$\mathcal{G}(\Gamma);\emptyset;\Theta \vdash \mathcal{B}^k_{\widetilde{x}}(\mathbf{0}) \triangleright \diamond \tag{A.1}$$

   where $\widetilde{x} = \mathtt{fv}(\mathbf{0}) = \emptyset$ and $\Theta = \{c_k : ?(\mathsf{end} \multimap \diamond)\}$. By Table 4.1: $\mathcal{B}^k_{\epsilon}(\mathbf{0}) = c_k?().\mathbf{0}$. By convention we know that $c_k?().\mathbf{0}$ stands for $c_k?(y).\mathbf{0}$ with $c_k : ?(\mathsf{end} \to \diamond)$. The following tree proves this case:

   $$\dfrac{\dfrac{\dfrac{}{\Gamma';\emptyset;\emptyset \vdash \mathbf{0} \triangleright \diamond}\,\text{Nil} \qquad c_k \notin \mathtt{dom}(\Gamma)}{\Gamma';\emptyset;c_k : \mathsf{end} \vdash \mathbf{0} \triangleright \diamond}\,\text{End} \qquad \dfrac{\dfrac{\dfrac{}{\mathcal{G}(\Gamma);y \triangleright \mathsf{end} \multimap \diamond;\emptyset \vdash y \triangleright \mathsf{end} \multimap \diamond}\,\text{LVar}}{\Gamma';\emptyset;\emptyset \vdash y \triangleright \mathsf{end} \multimap \diamond}\,\text{EProm}}{\Gamma';\emptyset;\emptyset \vdash y \triangleright \mathsf{end} \to \diamond}\,\text{Prom}}{\mathcal{G}(\Gamma);\emptyset;\Theta \vdash c_k?().\mathbf{0} \triangleright \diamond}\,\text{Rcv}$$

where $\Gamma' = \mathcal{G}(\Gamma), y : \mathsf{end} \to \diamond$. We know $c_k \notin \mathtt{dom}(\Gamma)$ since we use reserved names for propagators channels.

b) Case $P = u_i!\langle V \rangle.P'$. We distinguish three sub-cases: (i) $u_i \in \mathtt{dom}(\Delta)$ and (ii) $u_i \in \mathtt{dom}(\Gamma)$, and (iii) $u_i \in \mathtt{dom}(\Delta_\mu)$.

We consider sub-case (i) first. For this case Rule Send can be applied:

$$\text{Send}\ \frac{\Gamma; \Lambda_1; \Delta_1, \Delta_{\mu_1} \vdash P' \triangleright \diamond \qquad \Gamma; \Lambda_2; \Delta_2, \Delta_{\mu_2} \vdash V \triangleright U \qquad u_i : S \in \Delta_1, \Delta_2}{\Gamma; \Lambda_1, \Lambda_2; ((\Delta_1, \Delta_2) \setminus \{u_i : S\}), u_i :!\langle U \rangle;S, \Delta_{\mu_1}, \Delta_{\mu_2} \vdash u_i!\langle V \rangle.P' \triangleright \diamond} \tag{A.2}$$

Let $\widetilde{w} = \mathtt{fv}(P')$. Also, let $\Gamma'_1 = \Gamma \setminus \widetilde{w}$ and Θ_1 be a balanced environment such that

$$\mathtt{dom}(\Theta_1) = \{c_{k+1}, \ldots, c_{k+\wr P' \wr}\} \cup \{\overline{c_{k+2}}, \ldots, \overline{c_{k+\wr P' \wr}}\}$$

and $\Theta_1(c_{k+1}) = ?(\widetilde{M}_1)$ where $\widetilde{M}_1 = (\mathcal{G}(\Gamma), \mathcal{G}(\Lambda_1))(\widetilde{w})$. We define:

$$\Phi_i = \prod_{r \in \mathtt{dom}(\Delta_{\mu_i})} c^r : \langle \mathcal{R}^\star(\Delta_{\mu_i}(r)) {\multimap} \diamond \rangle \text{ for } i \in \{1, 2\} \tag{A.3}$$

Then, by IH on the first assumption of (A.2) we have:

$$\mathcal{G}(\Gamma'_1), \Phi_1; \emptyset; \mathcal{G}(\Delta_1), \Theta_1 \vdash \mathcal{B}^{k+1}_{\tilde{z}}(P') \triangleright \diamond \tag{A.4}$$

Let $\widetilde{y} = \mathtt{fv}(V)$ and $\Gamma'_2 = \Gamma \setminus \widetilde{y}$. Then, by IH (Part 2) on the second assumption of (A.2) we have:

$$\mathcal{G}(\Gamma), \Phi_2; \mathcal{G}(\Lambda_2); \mathcal{G}(\Delta_2) \vdash \mathcal{V}_{\tilde{y}}(V) \triangleright \mathcal{G}(U) \tag{A.5}$$

We may notice that if $U = C {\to} \diamond$ then $\Lambda_2 = \emptyset$ and $\Delta_2 = \emptyset$. Let $\widetilde{x} = \mathtt{fv}(P)$. We define $\Theta = \Theta_1, \Theta'$, where:

$$\Theta' = c_k :?(\widetilde{M}), \overline{c_{k+1}} :!\langle \widetilde{M}_2 \rangle$$

with $\widetilde{M} = (\mathcal{G}(\Gamma), \mathcal{G}(\Lambda_1, \Lambda_2))(\widetilde{x})$. By Definition 4.3.6, we know $\wr P \wr = \wr P' \wr + 1$, so

$$\mathtt{dom}(\Theta) = \{c_k, \ldots, c_{k+\wr P \wr - 1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\wr P \wr - 1}}\}$$

By construction Θ is balanced since $\Theta(c_{k+1})$ dual $\Theta(\overline{c_{k+1}})$ and Θ_1 is balanced. By Table 4.1, we have:

$$\mathcal{B}^k_{\tilde{x}}(u_i!\langle V \rangle.P') = c_k?(\widetilde{x}).u_i!\langle \mathcal{V}_{\tilde{y}}(V\{u_{i+1}/u_i\}) \rangle.\overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \mid \mathcal{B}^{k+1}_{\tilde{w}}(P'\{u_{i+1}/u_i\})$$

We know $\mathtt{fv}(P') \subseteq \mathtt{fv}(P)$ and $\mathtt{fv}(V) \subseteq \mathtt{fv}(P)$ that is $\widetilde{w} \subseteq \widetilde{x}$ and $\widetilde{y} \subseteq \widetilde{x}$. Let $\Gamma_1 = \Gamma \setminus \widetilde{x}$ and $\Phi = \Phi_1, \Phi_2$. We shall prove the following judgment:

$$\mathcal{G}(\Gamma_1), \Phi; \emptyset; \mathcal{G}(((\Delta_1, \Delta_2) \setminus \{u_i : S\}), u_i :!\langle U \rangle;S), \Theta \vdash \mathcal{B}^k_{\tilde{x}}(u_i!\langle V \rangle.P') \triangleright \diamond \tag{A.6}$$

Let $\sigma = \{u_{i+1}/u_i\}$. To type the left-hand side component of $\mathcal{B}^k_{\tilde{x}}(u_i!\langle V \rangle.P')$ we use some auxiliary derivations:

$$\text{End}\ \dfrac{\text{PolySend}\ \dfrac{\text{End}\ \dfrac{\text{Nil}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond}}{\mathcal{G}(\Gamma), \Phi; \emptyset; \overline{c_{k+1}} : \mathsf{end} \vdash \mathbf{0} \triangleright \diamond} \qquad \text{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1); \emptyset \vdash \widetilde{w} \triangleright \widetilde{M}_2}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1); \overline{c_{k+1}} :!\langle \widetilde{M}_2 \rangle;\mathsf{end} \vdash \overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \triangleright \diamond}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1); \overline{c_{k+1}} :!\langle \widetilde{M}_2 \rangle;\mathsf{end}, u_i : \mathsf{end} \vdash \overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \triangleright \diamond} \tag{A.7}$$

$$\text{(Lemma 2.2.3) with } \Phi_1\ \dfrac{\text{(Lemma 2.2.1) with } \{\widetilde{n}/\widetilde{u}\}\ \dfrac{(A.5)}{\mathcal{G}(\Gamma), \Phi_2; \mathcal{G}(\Lambda_2); \mathcal{G}(\Delta_2\sigma) \vdash \mathcal{V}_{\widetilde{y}}(V\sigma) \triangleright \mathcal{G}(U)}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_2); \mathcal{G}(\Delta_2\sigma) \vdash \mathcal{V}_{\widetilde{y}}(V\sigma) \triangleright \mathcal{G}(U)} \tag{A.8}$$

$$\text{End}\ \dfrac{\text{Send}\ \dfrac{u_i :!\langle \mathcal{G}(U)\rangle;\texttt{end} \in \mathcal{G}(\Delta_2\sigma), u_i :!\langle \mathcal{G}(U)\rangle;\texttt{end}, \overline{c_{k+1}} :!\langle \widetilde{M}_2\rangle;\texttt{end} \qquad (A.7) \qquad (A.8)}{\begin{array}{c}\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1, \Lambda_2); \mathcal{G}(\Delta_2\sigma), u_i :!\langle \mathcal{G}(U)\rangle;\texttt{end}, \overline{c_{k+1}} :!\langle \widetilde{M}_2\rangle;\texttt{end} \vdash \\ u_i!\langle \mathcal{V}_{\widetilde{y}}(V\sigma)\rangle.\overline{c_{k+1}}!\langle \widetilde{w}\rangle.\mathbf{0} \triangleright \diamond\end{array}}}{\begin{array}{c}\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1, \Lambda_2); \mathcal{G}(\Delta_2\sigma), u_i :!\langle \mathcal{G}(U)\rangle;\texttt{end}, \overline{c_{k+1}} :!\langle \widetilde{M}_2\rangle;\texttt{end}, c_k : \texttt{end} \vdash \\ u_i!\langle \mathcal{V}_{\widetilde{y}}(V\sigma)\rangle.\overline{c_{k+1}}!\langle \widetilde{w}\rangle.\mathbf{0} \triangleright \diamond\end{array}} \tag{A.9}$$

$$\text{PolyRcv}\ \dfrac{(A.9) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_2); \emptyset \vdash \widetilde{x} : \widetilde{M}}}{\begin{array}{c}\mathcal{G}(\Gamma_1), \Phi; \emptyset; \mathcal{G}(\Delta_2\sigma), u_i :!\langle \mathcal{G}(U)\rangle;\texttt{end}, \Theta' \vdash \\ c_k?(\widetilde{x}).u_i!\langle \mathcal{V}_{\widetilde{y}}(V\sigma)\rangle.\overline{c_{k+1}}!\langle \widetilde{w}\rangle.\mathbf{0} \triangleright \diamond\end{array}} \tag{A.10}$$

The following tree proves this case:

$$\text{Par}\ \dfrac{(A.10) \qquad \text{(Lemma 2.2.3) with } \widetilde{x} \setminus \widetilde{w} \text{ and } \Phi_2\ \dfrac{\text{(Lemma 2.2.1) with } \{\widetilde{n}/\widetilde{u}\}\ \dfrac{(A.4)}{\mathcal{G}(\Gamma_1'), \Phi_1; \emptyset; \mathcal{G}(\Delta_1\sigma), \Theta_1 \vdash \mathcal{B}^{k+r+1}_{\widetilde{w}}(P'\sigma) \triangleright \diamond}}{\mathcal{G}(\Gamma_1), \Phi; \emptyset; \mathcal{G}(\Delta_1\sigma), \Theta_1 \vdash \mathcal{B}^{k+r+1}_{\widetilde{w}}(P'\sigma) \triangleright \diamond}}{\mathcal{G}(\Gamma_1), \Phi; \emptyset; \mathcal{G}(((\Delta_1, \Delta_2) \setminus \{u_i : S\}), u_i :!\langle U\rangle;S), \Theta \vdash \mathcal{B}^{k}_{\widetilde{x}}(u_i!\langle V\rangle.P') \triangleright \diamond} \tag{A.11}$$

where $\widetilde{n} = (u_{i+1}, \ldots, u_{i+|\mathcal{G}(S)|})$ and $\widetilde{u} = (u_i, \ldots, u_{i+|\mathcal{G}(S)|-1})$. This concludes sub-case (i).

Now, we consider sub-case (ii). For this sub-case Rule Req can be applied:

$$\text{Req}\ \dfrac{\Gamma; \emptyset; \emptyset \vdash u \triangleright \langle U\rangle \qquad \Gamma; \Lambda; \Delta_1, \Delta_{\mu_1} \triangleright P' \triangleright \diamond \qquad \Gamma; \emptyset; \Delta_2, \Delta_{\mu_2} \vdash V \triangleright U}{\Gamma; \Lambda; \Delta_1, \Delta_2, \Delta_{\mu_1}, \Delta_{\mu_2} \vdash u!\langle V\rangle.P' \triangleright \diamond} \tag{A.12}$$

Let $\widetilde{w} = \texttt{fv}(P')$. Further, let $\Gamma_1' = \Gamma \setminus \widetilde{w}$ and let Θ_1, Φ_1, and Φ_2 be environments defined as in sub-case (i). By IH on the second assumption of (A.12) we have:

$$\mathcal{G}(\Gamma_1'), \Phi_1; \emptyset; \mathcal{G}(\Delta_1), \Theta_1 \vdash \mathcal{B}^{k+1}_{\widetilde{w}}(P') \triangleright \diamond \tag{A.13}$$

Let $\widetilde{y} = \texttt{fv}(V)$. By IH on the second assumption of (A.2) we have:

$$\mathcal{G}(\Gamma), \Phi_2; \emptyset; \mathcal{G}(\Delta_2) \vdash \mathcal{V}_{\widetilde{y}}(V) \triangleright \mathcal{G}(U) \tag{A.14}$$

Let $\widetilde{x} = \texttt{fv}(P)$ and $\Gamma_1 = \Gamma \setminus \widetilde{x}$. We define $\Theta = \Theta_1, \Theta'$, where:

$$\Theta' = c_k :?(\widetilde{M}), \overline{c_{k+1}} :!\langle \widetilde{M}_2\rangle$$

with $\widetilde{M} = (\mathcal{G}(\Gamma), \mathcal{G}(\Lambda))(\widetilde{x})$. By Definition 4.3.6, we know $\lbag P \rbag = \lbag P' \rbag + 1$, so

$$\texttt{dom}(\Theta) = (c_k, \ldots, c_{k+\lbag P \rbag - 1}) \cup (\overline{c_{k+1}}, \ldots, \overline{c_{k+\lbag P \rbag - 1}})$$

By construction Θ is balanced since $\Theta(c_{k+1})$ $\mathtt{dual}$ $\Theta(\overline{c_{k+1}})$ and Θ_1 is balanced. By Table 4.1, we have:

$$\mathcal{B}^{k}_{\tilde{x}}(u_i!\langle V\rangle.P') = c_k?(\widetilde{x}).u_i!\langle \mathcal{V}_{\tilde{y}}(V)\rangle.\overline{c_{k+1}}!\langle\widetilde{w}\rangle.\mathbf{0} \mid \mathcal{B}^{k+1}_{\tilde{w}}(P')$$

We know $\mathtt{fv}(P')\subseteq\mathtt{fv}(P)$ and $\mathtt{fv}(V)\subseteq\mathtt{fv}(P)$ that is $\widetilde{w}\subseteq\widetilde{x}$ and $\widetilde{y}\subseteq\widetilde{x}$.

To prove

$$\mathcal{G}(\Gamma_1),\Phi;\emptyset;\mathcal{G}(\Delta_1,\Delta_2),\Theta\vdash\mathcal{B}^{k}_{\tilde{x}}(u_i!\langle V\rangle.P')$$

we use some auxiliary derivations:

$$\text{PolySend}\ \dfrac{\text{End}\ \dfrac{\text{Nil}\ \dfrac{}{\mathcal{G}(\Gamma),\Phi;\emptyset;\emptyset\vdash\mathbf{0}\triangleright\diamond}}{\mathcal{G}(\Gamma),\Phi;\emptyset;\overline{c_{k+1}}:\mathsf{end}\vdash\mathbf{0}\triangleright\diamond}\qquad \text{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma),\Phi;\mathcal{G}(\Lambda);\emptyset\vdash\widetilde{w}:\widetilde{M_2}}}{\mathcal{G}(\Gamma),\Phi;\mathcal{G}(\Lambda);\overline{c_{k+1}}:!\langle\widetilde{M_2}\rangle;\mathsf{end}\vdash\overline{c_{k+1}}!\langle\widetilde{w}\rangle.\mathbf{0}\triangleright\diamond} \tag{A.15}$$

$$\text{Req}\ \dfrac{(\text{A.15})\qquad (\text{Lemma 2.2.3) with }\Phi_1\ \dfrac{(\text{A.14})}{\mathcal{G}(\Gamma),\Phi;\emptyset;\mathcal{G}(\Delta_2)\vdash\mathcal{V}_{\tilde{y}}(V)\triangleright\mathcal{G}(U)}}{\mathcal{G}(\Gamma),\Phi;\mathcal{G}(\Lambda);\mathcal{G}(\Delta_2),\overline{c_{k+1}}:!\langle\widetilde{M_2}\rangle;\mathsf{end}\vdash u_i!\langle\mathcal{V}_{\tilde{y}}(V)\rangle.\overline{c_{k+1}}!\langle\widetilde{w}\rangle.\mathbf{0}\triangleright\diamond} \tag{A.16}$$

$$\text{PolyRcv}\ \dfrac{(\text{A.16})\qquad \text{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma),\Phi;\mathcal{G}(\Lambda);\emptyset\vdash\widetilde{x}\triangleright\widetilde{M}}}{\mathcal{G}(\Gamma_1),\Phi;\emptyset;\mathcal{G}(\Delta_2),\Theta'\vdash c_k?(\widetilde{x}).u_i!\langle\mathcal{V}_{\tilde{y}}(V)\rangle.\overline{c_{k+1}}!\langle\widetilde{w}\rangle.\mathbf{0}\triangleright\diamond} \tag{A.17}$$

The following tree proves this case:

$$\text{Par}\ \dfrac{(\text{A.17})\qquad (\text{Lemma 2.2.3) with }\tilde{x}\setminus\tilde{w}\text{ and }\Phi_2\ \dfrac{(\text{A.13})}{\mathcal{G}(\Gamma_1),\Phi;\emptyset;\mathcal{G}(\Delta_1),\Theta_1\vdash\mathcal{B}^{k+1}_{\tilde{w}}(P')\triangleright\diamond}}{\mathcal{G}(\Gamma_1),\Phi;\emptyset;\mathcal{G}(\Delta_1,\Delta_2),\Theta\vdash\mathcal{B}^{k}_{\tilde{x}}(u_i!\langle V\rangle.P')\triangleright\diamond} \tag{A.18}$$

This concludes sub-case (ii).

Now, we consider sub-case (iii). Here we know $P = u_i!\langle V\rangle.P'$ and $u_i : S\in\Delta_\mu$. For this case Rule Send can be applied:

$$\text{Send}\ \dfrac{\Gamma;\Lambda_1;\Delta_1,\Delta_{\mu_1}\vdash P'\triangleright\diamond\qquad \Gamma;\Lambda_2;\Delta_2,\Delta_{\mu_2}\vdash V\triangleright U\qquad u_i:S'\in\Delta_{\mu_1},\Delta_{\mu_2}}{\Gamma;\Lambda_1,\Lambda_2;\Delta_1,\Delta_2,((\Delta_{\mu_1},\Delta_{\mu_2})\setminus\{u_i:S'\}),u_i:!\langle U\rangle;S'\vdash u_i!\langle V\rangle.P'\triangleright\diamond} \tag{A.19}$$

Let $\widetilde{w}=\mathtt{fv}(P')$. Let Θ_1, Θ', Θ, Φ_1, and Φ_2 be defined as in the previous sub-case. Also, let $\Gamma'_1=\Gamma\setminus\widetilde{w}$. Then, by IH on the first assumption of (A.19) we have:

$$\mathcal{G}(\Gamma'_1),\Phi_1;\emptyset;\mathcal{G}(\Delta_1),\Theta_1\vdash\mathcal{B}^{k+1}_{\tilde{w}}(P')\triangleright\diamond \tag{A.20}$$

Let $\Gamma'_2=\Gamma\setminus\widetilde{y}$. Then, by IH (Part 2) on the second assumption of (A.19) we have:

$$\mathcal{G}(\Gamma'_2),\Phi_2;\mathcal{G}(\Lambda_2);\mathcal{G}(\Delta_2)\vdash\mathcal{V}_{\tilde{y}}(V)\triangleright\mathcal{G}(U) \tag{A.21}$$

By Table 4.1 we have:

$$\mathcal{B}^{k}_{\tilde{x}}(P) = c_k?(\tilde{x}).c^u!\langle N_V\rangle.\mathbf{0} \mid \mathcal{B}^{k+1}_{\tilde{w}}(P')$$
$$\text{where } N_V = \lambda\tilde{z}.\, z_{[S\rangle}!\langle \mathcal{V}_{\tilde{y}}(V)\rangle.(\overline{c_{k+1}}!\langle\tilde{w}\rangle \mid c^u?(x).x\,\tilde{z})$$

We notice that $u_i \in \mathtt{rn}(V), \mathtt{rn}(P)$ since u_i has tail-recursive type S. Hence, by (A.3) we know $(\Phi_1, \Phi_2)(c^u) = \langle \mathcal{R}^\star(S) \multimap \diamond\rangle$. Further, we know that $S = !\langle U\rangle;S'$ and by Definition 4.3.3, $\mathcal{R}^\star(S) = \mathcal{R}^\star(S')$. So we define $\Phi = \Phi_1, \Phi_2$. Let $\Gamma_1 = \Gamma \setminus \tilde{x}$ where $\tilde{x} = \mathtt{fv}(P)$. We shall prove the following judgment:

$$\mathcal{G}(\Gamma_1), \Phi; \emptyset; \mathcal{G}(\Delta_1, \Delta_2), \Theta \vdash \mathcal{B}^{k}_{\tilde{x}}(u_i!\langle V\rangle.P') \triangleright \diamond$$

We use auxiliary derivations:

$$\text{PolyApp}\ \frac{\text{LVar}\ \dfrac{}{\begin{array}{c}\mathcal{G}(\Gamma_1), \Phi; x : \mathcal{R}^\star(S) \multimap \diamond; \emptyset \vdash \\ x \triangleright \mathcal{R}^\star(S) \multimap \diamond\end{array}} \qquad \text{PolySess}\ \dfrac{}{\begin{array}{c}\mathcal{G}(\Gamma_1), \Phi; \emptyset; \tilde{z} : \mathcal{R}^\star(S) \vdash \\ \tilde{z} \triangleright \mathcal{R}^\star(S)\end{array}}}{\mathcal{G}(\Gamma_1), \Phi; x : \mathcal{R}^\star(S) \multimap \diamond; \tilde{z} : \mathcal{R}^\star(S) \vdash x\,\tilde{z} \triangleright \diamond} \tag{A.22}$$

$$\text{Acc}\ \frac{(\text{A.22}) \qquad \text{Sh}\ \dfrac{}{\begin{array}{c}\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash \\ c^u \triangleright \langle \mathcal{R}^\star(S) \multimap \diamond\rangle\end{array}} \qquad \text{LVar}\ \dfrac{}{\begin{array}{c}\mathcal{G}(\Gamma), \Phi; x : \mathcal{R}^\star(S) \multimap \diamond; \emptyset \vdash \\ x \triangleright \mathcal{R}^\star(S) \multimap \diamond\end{array}}}{\mathcal{G}(\Gamma), \Phi; \emptyset; \Theta', \tilde{z} : \mathcal{R}^\star(S) \vdash c^u?(x).x\,\tilde{z} \triangleright \diamond} \tag{A.23}$$

$$\text{End}\ \frac{\text{Nil}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond} \qquad c_{k+1} \notin \mathtt{dom}(\Gamma, \Phi)}{\mathcal{G}(\Gamma), \Phi; \emptyset; c_{k+1} : \mathsf{end} \vdash \mathbf{0} \triangleright \diamond} \tag{A.24}$$

$$\text{PolySend}\ \frac{c_{k+1} :!\langle \widetilde{M_2}\rangle;\mathsf{end} \in \Theta' \qquad (\text{A.24}) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_2); \emptyset \vdash \tilde{w} \triangleright \widetilde{M_2}}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1); \overline{c_{k+1}} :!\langle \widetilde{M_2}\rangle \vdash \overline{c_{k+1}}!\langle\tilde{w}\rangle \triangleright \diamond} \tag{A.25}$$

$$\text{Par}\ \frac{(\text{A.25}) \qquad (\text{A.23})}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1); \overline{c_{k+1}} :!\langle \widetilde{M_2}\rangle, \tilde{z} : \mathcal{R}^\star(S) \vdash \overline{c_{k+1}}!\langle\tilde{w}\rangle \mid c^u?(x).x\,\tilde{z} \triangleright \diamond} \tag{A.26}$$

$$\text{Send}\ \frac{(\text{A.26}) \qquad (\text{Lemma 2.2.2) with } \tilde{z}\ \dfrac{(\text{Lemma 2.2.2) with } \Phi_1\ \dfrac{(\text{A.20})}{\mathcal{G}(\Gamma_2'), \Phi; \mathcal{G}(\Lambda_2); \emptyset \vdash \mathcal{V}_{\tilde{y}}(V) \triangleright \mathcal{G}(U)}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_2); \emptyset \vdash \mathcal{V}_{\tilde{y}}(V) \triangleright \mathcal{G}(U)}}{\begin{array}{c}\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1, \Lambda_2); \mathcal{G}(\Delta_2), \tilde{z} : \mathcal{R}^\star(S), \overline{c_{k+1}} :!\langle \widetilde{M_2}\rangle \vdash \\ z_{[S\rangle}!\langle \mathcal{V}_{\tilde{y}}(V)\rangle.(\overline{c_{k+1}}!\langle\tilde{w}\rangle \mid c^u?(x).x\,\tilde{z})\end{array}} \tag{A.27}$$

By Lemma A.1.1 we know that if $\tilde{z} : \mathcal{R}^\star(S)$ then $z_{[S\rangle} : \mu\mathsf{t}.!\langle \mathcal{G}(U)\rangle;\mathsf{t}$.

$$\text{PolyAbs}\;\dfrac{(A.27) \qquad \text{PolySess}\;\dfrac{}{\mathcal{G}(\Gamma),\Phi;\emptyset;\widetilde{z}:\mathcal{R}^\star(S)\vdash \widetilde{z}\triangleright \mathcal{R}^\star(S)}}{\mathcal{G}(\Gamma),\Phi;\mathcal{G}(\Lambda_1,\Lambda_2);\mathcal{G}(\Delta_2),\overline{c_{k+1}}:!\langle\widetilde{M_2}\rangle\vdash N_V\triangleright\mathcal{R}^\star(S)\multimap\diamond} \tag{A.28}$$

$$\text{Req}\;\dfrac{\text{LVar}\;\dfrac{}{\mathcal{G}(\Gamma),\Phi;\emptyset;\emptyset\vdash c^u\triangleright\langle\mathcal{R}^\star(S)\multimap\diamond\rangle)} \qquad \text{Nil}\;\dfrac{}{\mathcal{G}(\Gamma),\Phi;\emptyset;\emptyset\vdash \mathbf{0}\triangleright\diamond} \qquad (A.28)}{\mathcal{G}(\Gamma),\Phi;\mathcal{G}(\Lambda_1,\Lambda_2);\mathcal{G}(\Delta_2),\overline{c_{k+1}}:!\langle\widetilde{M_2}\rangle\vdash c^u!\langle N_V\rangle.\mathbf{0}\triangleright\diamond} \tag{A.29}$$

$$\text{PolyRcv}\;\dfrac{(A.29) \qquad \text{PolyVar}\;\dfrac{}{\mathcal{G}(\Gamma),\Phi;\mathcal{G}(\Lambda_1,\Lambda_2);\emptyset\vdash\widetilde{x}\triangleright\widetilde{M}}}{\mathcal{G}(\Gamma_1),\Phi;\emptyset;\mathcal{G}(\Delta_2),\Theta'\vdash c_k?(\widetilde{x}).c^u!\langle N_V\rangle.\mathbf{0}\triangleright\diamond} \tag{A.30}$$

The following tree proves this case:

$$\text{Par}\;\dfrac{(A.30) \qquad \text{(Lemma 2.2.3) with } \tilde{y}\;\dfrac{\text{(Lemma 2.2.2) with } \Phi_2\;\dfrac{(A.20)}{\mathcal{G}(\Gamma_1'),\Phi;\emptyset;\mathcal{G}(\Delta_1),\Theta_1\vdash\mathcal{B}^{k+1}_{\tilde{w}}(P')\triangleright\diamond}}{\mathcal{G}(\Gamma_1),\Phi;\emptyset;\mathcal{G}(\Delta_1),\Theta_1\vdash\mathcal{B}^{k+1}_{\tilde{w}}(P')\triangleright\diamond}}{\mathcal{G}(\Gamma_1),\Phi;\emptyset;\mathcal{G}(\Delta_1,\Delta_2),\Theta\vdash\mathcal{B}^{k}_{\tilde{x}}(r!\langle V\rangle.P')\triangleright\diamond} \tag{A.31}$$

This concludes the analysis for the output case $P = u_i!\langle V\rangle.P'$. We remark that the proof for the case when $V = y$ is specialization of above the proof where $\tilde{y} = \mathtt{fv}(y) = y$, $\mathcal{V}_{\tilde{y}}(y) = y$ and it holds that $y\sigma = y$.

c) Case $P = u_i?(y).P'$. We distinguish two sub-cases: (i) $u_i \in \mathtt{dom}(\Delta)$, (ii) $u_i \in \mathtt{dom}(\Gamma)$, and (iii) $u_i \in \mathtt{dom}(\Delta_\mu)$. We consider sub-cases (i) and (ii); we omit sub-case (iii) as it follows the same reasoning as the corresponding sub-case of the previous (Send) case.

We consider sub-case (i) first. For this case Rule Rcv can be applied:

$$\text{Rcv}\;\dfrac{\Gamma;\Lambda_1;\Delta',u_i:S,\Delta_\mu\vdash P'\triangleright\diamond \qquad \Gamma;\Lambda_2;\emptyset\vdash y\triangleright U}{\Gamma\setminus y;\Lambda_1\setminus\Lambda_2;\Delta',u_i:?(U);S,\Delta_\mu\vdash u_i?(y).P'\triangleright\diamond} \tag{A.32}$$

Let $\widetilde{x} = \mathtt{fv}(P)$ and $\widetilde{w} = \mathtt{fv}(P')$. Also, let $\Gamma_1' = \Gamma\setminus\widetilde{w}$ and Θ_1 be a balanced environment such that

$$\mathtt{dom}(\Theta_1) = \{c_{k+1},\dots,c_{k+\wr P'\wr}\}\cup\{\overline{c_{k+2}},\dots,\overline{c_{k+\wr P'\wr}}\}$$

and $\Theta_1(c_{k+1}) = ?(\widetilde{M}')$ where $\widetilde{M}' = (\mathcal{G}(\Gamma),\mathcal{G}(\Lambda_1))(\widetilde{w})$. We define:

$$\Phi = \prod_{r\in\mathtt{dom}(\Delta_\mu)} c^r:\langle\mathcal{R}^\star(\Delta_{\mu_i}(r))\multimap\diamond\rangle \tag{A.33}$$

Then, by IH on the first assumption of (A.32) we know:

$$\mathcal{G}(\Gamma_1'),\Phi;\emptyset;\mathcal{G}(\Delta',u_i:S),\Theta_1\vdash\mathcal{B}^{k+1}_{\tilde{w}}(P')\triangleright\diamond \tag{A.34}$$

By Definition 4.3.3 and Definition 4.3.10 and the second assumption of (A.32) we have:

$$\mathcal{G}(\Gamma);\mathcal{G}(\Lambda_2);\emptyset\vdash y\triangleright\mathcal{G}(U) \tag{A.35}$$

We define $\Theta = \Theta_1, \Theta'$, where

$$\Theta' = c_k :?(\widetilde{M}), \overline{c_{k+1}} :!\langle \widetilde{M'} \rangle$$

with $\widetilde{M} = (\mathcal{G}(\Gamma), \mathcal{G}(\Lambda_1 \setminus \Lambda_2))(\widetilde{x})$. By Definition 4.3.6, $\wr P \int = \wr P' \int + 1$ so

$$\mathtt{dom}(\Theta) = \{c_k, \ldots, c_{k+\wr P \int -1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\wr P \int -1}}\}$$

and Θ is balanced since $\Theta(c_{k+1})$ $\mathtt{dual}$ $\Theta(\overline{c_{k+1}})$ and Θ_1 is balanced. By Table 4.1:

$$\mathcal{B}^{k}_{\tilde{x}}(u_i?(y).P') = c_k?(\widetilde{x}).u_i?(y).\overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \mid \mathcal{B}^{k+1}_{\tilde{w}}(P'\{u_{i+1}/u_i\})$$

Let $\Gamma_1 = \Gamma \setminus \widetilde{x}$. We shall prove the following judgment:

$$\mathcal{G}(\Gamma_1 \setminus y), \Phi; \emptyset; \mathcal{G}(\Delta', u_i :?(U);S), \Theta \vdash \mathcal{B}^{k}_{\tilde{x}}(u_i?(y).P')$$

The left-hand side component of $\mathcal{B}^{k}_{\tilde{x}}(u_i?(y).P')$ is typed using some auxiliary derivations:

$$\textsc{End}\ \dfrac{\textsc{PolySend}\ \dfrac{\textsc{End}\ \dfrac{\textsc{Nil}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond}}{\mathcal{G}(\Gamma), \Phi; \emptyset; \overline{c_{k+1}} : \mathsf{end} \vdash \mathbf{0} \triangleright \diamond} \qquad \textsc{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1); \emptyset \vdash \widetilde{w} \triangleright \widetilde{M'}}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1), \mathcal{G}(\Lambda_2); \overline{c_{k+1}} :!\langle \widetilde{M'} \rangle;\mathsf{end} \vdash \overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \triangleright \diamond}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1), \mathcal{G}(\Lambda_2); \overline{c_{k+1}} :!\langle \widetilde{M'} \rangle;\mathsf{end}, u_i : \mathsf{end} \vdash \overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \triangleright \diamond} \tag{A.36}$$

$$\textsc{End}\ \dfrac{\textsc{Rcv}\ \dfrac{(A.36) \qquad (A.35)}{\begin{array}{c}\mathcal{G}(\Gamma \setminus y), \Phi; \mathcal{G}(\Lambda_1 \setminus \Lambda_2); u_i :?(\mathcal{G}(U));\mathsf{end}, \overline{c_{k+1}} :!\langle \widetilde{M'} \rangle;\mathsf{end} \vdash \\ u_i?(y).\overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \triangleright \diamond\end{array}}}{\begin{array}{c}\mathcal{G}(\Gamma \setminus y), \Phi; \mathcal{G}(\Lambda_1 \setminus \Lambda_2); u_i :?(\mathcal{G}(U));\mathsf{end}, \overline{c_{k+1}} :!\langle \widetilde{M'} \rangle;\mathsf{end}, c_k : \mathsf{end} \vdash \\ u_i?(y).\overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \triangleright \diamond\end{array}} \tag{A.37}$$

$$\textsc{PolyRcv}\ \dfrac{(A.37) \qquad \textsc{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma \setminus y), \Phi; \mathcal{G}(\Lambda_1); \emptyset \vdash \widetilde{x} \triangleright \widetilde{M}}}{\mathcal{G}(\Gamma_1 \setminus y), \Phi; \emptyset; u_i :?(\mathcal{G}(U));\mathsf{end}, \Theta' \vdash c_k?(\widetilde{x}).u_i?(y).\overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \triangleright \diamond} \tag{A.38}$$

The following tree proves this case:

$$\textsc{Par}\ \dfrac{(A.38) \qquad \text{(Lemma 2.2.1) with } \{\tilde{n}/\tilde{u}\}\ \dfrac{(A.34)}{\begin{array}{c}\mathcal{G}(\Gamma_1 \setminus y), \Phi; \emptyset; \mathcal{G}(\Delta', u_{i+1} : S), \Theta_1 \vdash \\ \mathcal{B}^{k+1}_{\tilde{w}}(P'\{u_{i+1}/u_i\}) \triangleright \diamond\end{array}}}{\begin{array}{c}\mathcal{G}(\Gamma_1 \setminus y), \Phi; \emptyset; \mathcal{G}(\Delta', u_i :?(U);S), \Theta \vdash \\ c_k?(\widetilde{x}).u_i?(y).\overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \mid \mathcal{B}^{k+1}_{\tilde{w}}(P'\{u_{i+1}/u_i\}) \triangleright \diamond\end{array}} \tag{A.39}$$

where $\widetilde{n} = (u_{i+1}, \ldots, u_{i+|\mathcal{G}(S)|})$ and $\widetilde{u} = (u_i, \ldots, u_{i+|\mathcal{G}(S)|-1})$. We may notice that if $y \in \mathtt{fv}(P')$ then $\Gamma_1' = \Gamma_1 \setminus y$. On the other hand, when $y \notin \mathtt{fv}(P')$ then $\Gamma_1' = \Gamma_1$

so we need to apply Lemma 2.2.3 with y after Lemma 2.2.1 to (A.34) in (A.39). Note that we have used the following for the right assumption of (A.39):

$$\begin{aligned} \mathcal{G}(\Delta', u_i : S)\{\tilde{n}/\tilde{u}\} &= \mathcal{G}(\Delta', u_{i+1} : S) \\ \mathcal{B}^{k+1}_{\tilde{w}}(P')\{\tilde{n}/\tilde{u}\} &= \mathcal{B}^{k+1}_{\tilde{w}}(P'\{u_{i+1}/u_i\}) \end{aligned}$$

This concludes sub-case (i). We now consider sub-case (ii), i.e., $u_i \in \mathtt{dom}(\Gamma)$. Here Rule Acc can be applied:

$$\text{Acc}\ \frac{\Gamma; \emptyset; \emptyset \vdash u_i \triangleright \langle U \rangle \qquad \Gamma; \Lambda_1; \Delta, \Delta_\mu \vdash P' \triangleright \diamond \qquad \Gamma; \Lambda_2; \emptyset \vdash y \triangleright U}{\Gamma \setminus y; \Lambda_1 \setminus \Lambda_2; \Delta, \Delta_\mu \vdash u_i?(y).P' \triangleright \diamond} \tag{A.40}$$

Let $\widetilde{x} = \mathtt{fv}(P)$ and $\widetilde{w} = \mathtt{fv}(P')$. Furthermore, let Θ_1, Θ, Γ_1, Γ'_1, and Φ be defined as in sub-case (i). By IH on the second assumption of (A.40) we have:

$$\mathcal{G}(\Gamma'_1), \Phi; \emptyset; \mathcal{G}(\Delta), \Theta_1 \vdash \mathcal{B}^{k+1}_{\tilde{w}}(P') \triangleright \diamond \tag{A.41}$$

By Definition 4.3.3 and Definition 4.3.10 and the first assumption of (A.40) we have:

$$\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash u_i \triangleright \langle \mathcal{G}(U) \rangle \tag{A.42}$$

By Definition 4.3.3, Definition 4.3.10, and the third assumption of (A.40) we have:

$$\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_2); \emptyset \vdash y \triangleright \mathcal{G}(U) \tag{A.43}$$

By Table 4.1, we have:

$$\mathcal{B}^{k}_{\tilde{x}}(u_i?(y).P') = c_k?(\widetilde{x}).u_i?(y).\overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \mid \mathcal{B}^{k+1}_{\tilde{w}}(P'\{u_{i+1}/u_i\}) \tag{A.44}$$

We shall prove the following judgment:

$$\mathcal{G}(\Gamma_1 \setminus y), \Phi; \emptyset; \mathcal{G}(\Delta), \Theta \vdash \mathcal{B}^{k}_{\tilde{x}}(u_i?(y).P') \triangleright \diamond \tag{A.45}$$

To this end, we use some auxiliary derivations:

$$\text{PolySend}\ \frac{\text{End}\ \dfrac{\text{Nil}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond}}{\mathcal{G}(\Gamma), \Phi; \emptyset; \overline{c_{k+1}} : \mathsf{end} \vdash \mathbf{0} \triangleright \diamond} \qquad \text{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1); \emptyset \vdash \widetilde{w} \triangleright \widetilde{M}'}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1); \overline{c_{k+1}} :!\langle \widetilde{M}' \rangle;\mathsf{end} \vdash \overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \triangleright \diamond} \tag{A.46}$$

$$\text{End}\ \frac{\text{Acc}\ \dfrac{(\text{A.42}) \qquad (\text{A.46}) \qquad (\text{A.43})}{\mathcal{G}(\Gamma \setminus y), \Phi; \mathcal{G}(\Lambda_1); \overline{c_{k+1}} :!\langle \widetilde{M}' \rangle;\mathsf{end} \vdash u_i?(y).\overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \triangleright \diamond}}{\mathcal{G}(\Gamma \setminus y), \Phi; \mathcal{G}(\Lambda_1 \setminus \Lambda_2); \overline{c_{k+1}} :!\langle \widetilde{M}' \rangle;\mathsf{end}, c_k : \mathsf{end} \vdash u_i?(y).\overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \triangleright \diamond} \tag{A.47}$$

$$\text{PolyRcv}\ \frac{(\text{A.47}) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma \setminus y), \Phi; \mathcal{G}(\Lambda_1 \setminus \Lambda_2); \emptyset \vdash \widetilde{x} \triangleright \widetilde{M}}}{\mathcal{G}(\Gamma_1 \setminus y), \Phi; \emptyset; \Theta' \vdash c_k?(\widetilde{x}).u_i?(y).\overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \triangleright \diamond} \tag{A.48}$$

The following tree proves this sub-case:

$$\text{Par}\ \dfrac{(\text{A.48})\qquad (\text{A.41})}{\mathcal{G}(\Gamma_1 \setminus y), \Phi; \emptyset; \mathcal{G}(\Delta), \Theta \vdash c_k?(\widetilde{x}).u_i?(y).\overline{c_{k+1}}!\langle \widetilde{w} \rangle.\mathbf{0} \mid \mathcal{B}^{k+1}_{\widetilde{w}}(P') \triangleright \diamond} \tag{A.49}$$

As in sub-case (i), we may notice that if $y \in \mathtt{fv}(P')$ then $\Gamma_1' = \Gamma_1 \setminus y$. On the other hand, if $y \notin \mathtt{fv}(P')$ then $\Gamma_1' = \Gamma_1$ so we need to apply Lemma 2.2.3 with y to (A.41) in (A.49). This concludes the analysis for the input case $P = u_i?(y).P'$. This concludes sub-case (ii).

Now, we consider sub-case (iii). Here we know $P = u_i?(V).P'$ and $u_i : S \in \Delta_\mu$.

$$\text{Rcv}\ \dfrac{\Gamma; \Lambda_1; \Delta', \Delta_\mu, u_i : S' \vdash P' \triangleright \diamond \qquad \Gamma; \Lambda_2; \emptyset \vdash y \triangleright U}{\Gamma \setminus y; \Lambda_1 \setminus \Lambda_2; \Delta', \Delta_\mu, u_i :?(U);S' \vdash u_i?(y).P' \triangleright \diamond} \tag{A.50}$$

Let $\widetilde{w} = \mathtt{fv}(P')$. Let Θ_1,Θ_2, Θ', and Φ be defined as in the sub-case (i). Also, let $\Gamma_1' = \Gamma \setminus \widetilde{w}$. Then, by IH on the first assumption of (A.50) we have:

$$\mathcal{G}(\Gamma_1'), \Phi; \emptyset; \mathcal{G}(\Delta'), \Theta_1 \vdash \mathcal{B}^{k+1}_{\widetilde{w}}(P') \triangleright \diamond \tag{A.51}$$

Further, by IH on the second assumption of (A.50) we have:

$$\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_2); \emptyset \vdash y \triangleright \mathcal{G}(U) \tag{A.52}$$

By Table 4.1 we have:

$$\begin{aligned}\mathcal{B}^{k}_{\widetilde{x}}(P) &= c_k?(\widetilde{x}).c^u!\langle N_y \rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(P') \\ &\qquad\qquad \text{where } N_y = \lambda \widetilde{z}.\, z_{\lfloor S \rangle}?(y).(\overline{c_{k+1}}!\langle \widetilde{w} \rangle \mid c^u?(x).x\, \widetilde{z})\end{aligned}$$

Notice that $u_i \in \mathtt{rn}(P)$ as $\mathtt{tr}(u_i)$. Hence, by (A.33) we know $\Phi(c^u) = \langle \mathcal{R}^\star(S) \multimap \diamond \rangle$. Further, we know that $S =?(U);S'$ and by Definition 4.3.3, $\mathcal{R}^\star(S) = \mathcal{R}^\star(S')$. Let $\Gamma_1 = \Gamma \setminus \widetilde{x}$ where $\widetilde{x} = \mathtt{fv}(P)$. Thus, we shall prove the following judgment:

$$\mathcal{G}(\Gamma_1 \setminus y), \Phi; \emptyset; \mathcal{G}(\Delta'), \Theta \vdash \mathcal{B}^{k}_{\widetilde{x}}(u_i?(y)P') \triangleright \diamond$$

We use auxiliary derivations:

$$\text{PolyApp}\ \dfrac{\text{LVar}\ \dfrac{}{\mathcal{G}(\Gamma_1), \Phi; x : \mathcal{R}^\star(S) \multimap \diamond; \emptyset \vdash x \triangleright \mathcal{R}^\star(S) \multimap \diamond} \qquad \text{PolySess}\ \dfrac{}{\mathcal{G}(\Gamma_1), \Phi; \emptyset; \widetilde{z} : \mathcal{R}^\star(S) \vdash \widetilde{z} \triangleright \mathcal{R}^\star(S)}}{\mathcal{G}(\Gamma_1), \Phi; x : \mathcal{R}^\star(S) \multimap \diamond; \widetilde{z} : \mathcal{R}^\star(S) \vdash x\, \widetilde{z} \triangleright \diamond} \tag{A.53}$$

$$\text{Acc}\ \dfrac{(\text{A.53}) \qquad \text{Sh}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash c^u \triangleright \langle \mathcal{R}^\star(S) \multimap \diamond \rangle} \qquad \text{LVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; x : \mathcal{R}^\star(S) \multimap \diamond; \emptyset \vdash x \triangleright \mathcal{R}^\star(S) \multimap \diamond}}{\mathcal{G}(\Gamma), \Phi; \emptyset; \widetilde{z} : \mathcal{R}^\star(S) \vdash c^u?(x).x\, \widetilde{z} \triangleright \diamond} \tag{A.54}$$

$$\text{End}\ \dfrac{\text{Nil}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond} \qquad c_{k+1} \notin \mathtt{dom}(\Gamma, \Phi)}{\mathcal{G}(\Gamma), \Phi; \emptyset; c_{k+1} : \mathtt{end} \vdash \mathbf{0} \triangleright \diamond} \tag{A.55}$$

$$\textsc{PolySend}\ \dfrac{c_{k+1} :!\langle\widetilde{M'}\rangle;\mathtt{end} \in \Theta' \quad (\text{A.55}) \qquad \textsc{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1); \emptyset \vdash \widetilde{w} \triangleright \widetilde{M'}}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1); \overline{c_{k+1}} :!\langle\widetilde{M'}\rangle \vdash \overline{c_{k+1}}!\langle\widetilde{w}\rangle \triangleright \diamond} \tag{A.56}$$

$$\textsc{Par}\ \dfrac{(\text{A.56}) \qquad (\text{A.54})}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda_1); \Theta', \widetilde{z} : \mathcal{R}^\star(S) \vdash \overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid c^u?(x).x\,\widetilde{z} \triangleright \diamond} \tag{A.57}$$

$$\textsc{Rcv}\ \dfrac{(\text{A.57}) \qquad (\text{A.52})}{\mathcal{G}(\Gamma \setminus y), \Phi; \mathcal{G}(\Lambda_1 \setminus \Lambda_2); \overline{c_{k+1}} :!\langle\widetilde{M'}\rangle, \widetilde{z} : \mathcal{R}^\star(S) \vdash z_{[S\rangle}?(y).(\overline{c_{k+1}}!\langle\widetilde{w}\rangle \mid c^u?(x).x\,\widetilde{z})} \tag{A.58}$$

By Lemma A.1.1 we know that if $\widetilde{z} : \mathcal{R}^\star(S)$ then $z_{[S\rangle} : \mu\mathsf{t}.?(\mathcal{G}(U));\mathsf{t}$.

$$\textsc{PolyAbs}\ \dfrac{(\text{A.27}) \qquad \textsc{PolySess}\ \dfrac{}{\mathcal{G}(\Gamma \setminus y), \Phi; \emptyset; \widetilde{z} : \mathcal{R}^\star(S) \vdash \widetilde{z} \triangleright \mathcal{R}^\star(S)}}{\mathcal{G}(\Gamma \setminus y), \Phi; \mathcal{G}(\Lambda_1 \setminus \Lambda_2); \overline{c_{k+1}} :!\langle\widetilde{M'}\rangle \vdash N_y \triangleright \mathcal{R}^\star(S) \multimap \diamond} \tag{A.59}$$

$$\textsc{Req}\ \dfrac{\textsc{LVar}\ \dfrac{}{\mathcal{G}(\Gamma \setminus y), \Phi; \emptyset; \emptyset \vdash c^u \triangleright \langle \mathcal{R}^\star(S) \multimap \diamond\rangle} \qquad \textsc{Nil}\ \dfrac{}{\mathcal{G}(\Gamma \setminus y), \Phi; \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond} \qquad (\text{A.59})}{\mathcal{G}(\Gamma \setminus y), \Phi; \mathcal{G}(\Lambda_1 \setminus \Lambda_2); \overline{c_{k+1}} :!\langle\widetilde{M'}\rangle \vdash c^u!\langle N_y\rangle.\mathbf{0} \triangleright \diamond} \tag{A.60}$$

$$\textsc{PolyRcv}\ \dfrac{(\text{A.60}) \qquad \textsc{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma \setminus y), \Phi; \mathcal{G}(\Lambda); \emptyset \vdash \widetilde{x} \triangleright \widetilde{M}}}{\mathcal{G}(\Gamma_1 \setminus y), \Phi; \emptyset; \Theta' \vdash c_k?(\widetilde{x}).c^u!\langle N_y\rangle.\mathbf{0} \triangleright \diamond} \tag{A.61}$$

The following tree proves this case:

$$\textsc{Par}\ \dfrac{(\text{A.61}) \qquad (\text{Lemma 2.2.3) with } \tilde{y}\ \dfrac{(\text{A.51})}{\mathcal{G}(\Gamma_1 \setminus y), \Phi; \emptyset; \mathcal{G}(\Delta'), \Theta_1 \vdash \mathcal{B}^{k+1}_{\widetilde{w}}(P') \triangleright \diamond}}{\mathcal{G}(\Gamma_1 \setminus y), \Phi; \emptyset; \mathcal{G}(\Delta'), \Theta \vdash \mathcal{B}^{k}_{\widetilde{x}}(u_i?(y)P') \triangleright \diamond} \tag{A.62}$$

This concludes sub-case (iii).

d) Case $P = V\,(\widetilde{r}, u_i)$. We assume a certain order in the tuple $(\widetilde{r}, u_i)$: names in $\widetilde{r}$ have recursive session types $\widetilde{r} = (r_1, \ldots, r_n) : (S_1, \ldots, S_n)$, and u_i has non-recursive session type $u_i : C$. We distinguish two sub-cases: (i) $V : \widetilde{SC} \multimap \diamond$ and (ii) $V : \widetilde{SC} \rightarrow \diamond$. We will consider only sub-case (i) since the other is similar. For this case Rule PolyApp can be applied:

$$\textsc{PolyApp}\ \dfrac{\Gamma; \Lambda; \Delta_1, \Delta_{\mu_1} \vdash V \triangleright \widetilde{SC} \multimap \diamond \qquad \Gamma; \emptyset; \Delta_2, \Delta_{\mu_2} \vdash (\widetilde{r}, u_i) \triangleright \widetilde{SC}}{\Gamma; \Lambda; \Delta_1, \Delta_2, \Delta_{\mu_1}, \Delta_{\mu_2} \vdash V\,(\widetilde{r}, u_i)} \tag{A.63}$$

Let $\widetilde{x} = \mathtt{fv}(V)$ and $\Gamma_1 \setminus \widetilde{x}$. Let $\widetilde{x} = \mathtt{fv}(V)$ and let Θ_1 be a balanced environment such that

$$\mathtt{dom}(\Theta_1) = \{c_{k+1}, \ldots, c_{k+\{\!|V|\!\}}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\{\!|V|\!\}}}\}$$

and $\Theta_1(c_{k+1}) = ?(\widetilde{M})$ and $\Theta_1(\overline{c_{k+1}}) = !\langle\widetilde{M}\rangle$ where $\widetilde{M} = (\mathcal{G}(\Gamma), \mathcal{G}(\Lambda))(\tilde{x})$.

We define:

$$\Phi_1 = \prod_{r \in \mathtt{dom}(\Delta_{\mu_1})} c^r : \langle \mathcal{R}^\star(\Delta_{\mu_1}(r)) \multimap \diamond \rangle \tag{A.64}$$

Then, by IH (Part 2) on the first assumption of (A.63) we have:

$$\mathcal{G}(\Gamma_1), \Phi_1; \emptyset; \mathcal{G}(\Delta_1), \Theta_1 \vdash \mathcal{V}_{\tilde{x}}(V) \triangleright \mathcal{G}(\widetilde{S}C) \multimap \diamond \tag{A.65}$$

By Definition 4.3.10 and Definition 4.3.3 and the second assumption of (A.63) we have:

$$\mathcal{G}(\Gamma); \emptyset; \mathcal{G}(\Delta_2), \mathcal{G}(\Delta_{\mu_2}) \vdash (\widetilde{r}_1, \ldots, \widetilde{r}_n, \widetilde{m}) : \mathcal{G}(\widetilde{S}C) \tag{A.66}$$

where $\widetilde{r}_i = (r^i_i, \ldots, r^i_{i+|\mathcal{G}(S_i)|-1})$ for $i \in \{1, \ldots, n\}$ and $\widetilde{m} = (u_i, \ldots, u_{i+|G(C)|-1})$.

We define $\Phi = \Phi_1, \Phi_2$ where:

$$\Phi_2 = \prod_{r \in \mathtt{dom}(\Delta_{\mu_2})} c^r : \langle \mathcal{R}^\star(\Delta_{\mu_2}(r)) \multimap \diamond \rangle$$

We define $\Theta = \Theta_1, c_k : ?(\widetilde{M})$.

We will first consider the case where $n = 3$; the proof is then generalized for any $n \geq 1$:

- If $n = 3$ then $P = V\,(r_1, r_2, r_3, u_i)$. By Table 4.1 we have:

  $$\mathcal{B}^k_{\tilde{x}}(V\,(r_1, r_2, r_3, u_i)) = c_k?(\tilde{x}).c^{r_1}!\langle\lambda\widetilde{z}_1.\, c^{r_2}!\langle\lambda\widetilde{z}_2.\, c^{r_3}!\langle\lambda\widetilde{z}_3.\, Q\rangle.\mathbf{0}\rangle.\mathbf{0}\rangle.\mathbf{0}$$

  where $Q = \mathcal{V}_{\tilde{x}}(V)\,(\widetilde{z}_1, \ldots, \widetilde{z}_n, \widetilde{m})$; $\widetilde{z}_i = (z^i_1, \ldots, z^i_{|\mathcal{G}(S_i)|})$ for $i = \{1, 2, 3\}$; $\widetilde{m} = (u_i, \ldots, u_{i+|\mathcal{G}(C)|-1})$.

  We shall prove the following judgment:

  $$\mathcal{G}(\Gamma), \Phi; \emptyset; \mathcal{G}(\Delta_1\Delta_2), \Theta \vdash \mathcal{B}^k_{\tilde{x}}(V\,(\widetilde{r}, u_i)) \triangleright \diamond \tag{A.67}$$

  We use auxiliary derivations:

$$\frac{(A.65)}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1), \Theta_1 \vdash \mathcal{V}_{\tilde{x}}(V) \triangleright \mathcal{G}(\widetilde{S}C) \multimap \diamond}\ \text{(Lemma 2.2.2) with } \Phi_2 \tag{A.68}$$

$$\frac{(A.66)}{\mathcal{G}(\Gamma), \Phi; \emptyset; \mathcal{G}(\Delta_2), \mathcal{G}(\Delta_{\mu_2}) \vdash (\widetilde{z}_1, \widetilde{z}_2, \widetilde{z}_3, \widetilde{m}) \triangleright \mathcal{G}(\widetilde{S}C)}\ \text{(Lemma 2.2.1) with } \sigma \tag{A.69}$$

$$\frac{(A.68) \qquad (A.69)}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1, \Delta_2), \Theta_1, \mathcal{G}(\Delta_{\mu_2}) \vdash \mathcal{V}_{\tilde{x}}(V)\,(\widetilde{z}_1, \widetilde{z}_2, \widetilde{z}_3, \widetilde{m})}\ \text{PolyApp} \tag{A.70}$$

where $\sigma = \{\widetilde{n_1}/\widetilde{z}_1\} \cdot \{\widetilde{n_2}/\widetilde{z}_2\} \cdot \{\widetilde{n_3}/\widetilde{z}_3\}$ with $\widetilde{n}_i = (r_i^i, \ldots, r^i_{i+|\mathcal{G}(S_i)|-1})$ for $i = \{1,2,3\}$.

$$\text{PolyAbs}\ \dfrac{(\text{A.70}) \qquad \text{PolySess}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \widetilde{z}_3 : \mathcal{G}(S_3) \vdash \widetilde{z}_3 \triangleright \mathcal{G}(S_3)}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1\Delta_2), \Theta_1 \vdash \lambda \widetilde{z}_3.\, Q \triangleright \mathcal{G}(S_3) \multimap \diamond} \tag{A.71}$$

$$\text{Req}\ \dfrac{(\text{A.71}) \qquad \text{Nil}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash \mathbf{0}} \qquad \text{LVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash c^{r_3} \triangleright \langle \mathcal{G}(S_3) \multimap \diamond \rangle}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1\Delta_2), \Theta_1, \widetilde{z}_1 : \mathcal{G}(S_1), \widetilde{z}_2 : \mathcal{G}(S_2) \vdash c^{r_3}!\langle \lambda \widetilde{z}_3.\, Q \rangle.\mathbf{0} \triangleright \diamond} \tag{A.72}$$

$$\text{PolyAbs}\ \dfrac{(\text{A.72}) \qquad \text{PolySess}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \widetilde{z}_2 : \mathcal{G}(S_2) \vdash \widetilde{z}_2 \triangleright \mathcal{G}(S_2)}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1\Delta_2), \Theta_1, \widetilde{z}_1 : \mathcal{G}(S_1) \vdash \lambda \widetilde{z}_2.\, c^{r_3}!\langle \lambda \widetilde{z}_3.\, Q \rangle.\mathbf{0} \triangleright \mathcal{G}(S_2) \multimap \diamond} \tag{A.73}$$

$$\text{Req}\ \dfrac{(\text{A.73}) \qquad \text{Nil}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash \mathbf{0}} \qquad \text{LVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash c^{r_2} \triangleright \langle \mathcal{G}(S_2) \multimap \diamond \rangle}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1\Delta_2), \Theta_1, \widetilde{z}_1 : \mathcal{G}(S_1) \vdash c^{r_2}!\langle \lambda \widetilde{z}_2.\, c^{r_3}!\langle \lambda \widetilde{z}_3.\, Q \rangle.\mathbf{0} \rangle.\mathbf{0} \triangleright \diamond} \tag{A.74}$$

$$\text{PolyAbs}\ \dfrac{(\text{A.74}) \qquad \text{PolySess}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \widetilde{z}_1 : \mathcal{G}(S_1) \vdash \widetilde{z}_1 \triangleright \mathcal{G}(S_1)}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1\Delta_2), \Theta_1 \vdash \lambda \widetilde{z}_1.\, c^{r_2}!\langle \lambda \widetilde{z}_2.\, c^{r_3}!\langle \lambda \widetilde{z}_3.\, Q \rangle.\mathbf{0} \rangle.\mathbf{0} \triangleright \mathcal{G}(S_1) \multimap \diamond} \tag{A.75}$$

$$\text{Req}\ \dfrac{(\text{A.75}) \qquad \text{Nil}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash \mathbf{0}} \qquad \text{LVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash c^{r_1} \triangleright \langle \mathcal{G}(S_1) \multimap \diamond \rangle}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1\Delta_2), \Theta_1 \vdash c^{r_1}!\langle \lambda \widetilde{z}_1.\, c^{r_2}!\langle \lambda \widetilde{z}_2.\, c^{r_3}!\langle \lambda \widetilde{z}_3.\, Q \rangle.\mathbf{0} \rangle.\mathbf{0} \rangle.\mathbf{0}} \tag{A.76}$$

The following tree proves this case:

$$\text{PolyRcv}\ \dfrac{(\text{A.76}) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma_1), \Phi; \mathcal{G}(\Lambda); \emptyset \vdash \widetilde{x} \triangleright \widetilde{M}}}{\mathcal{G}(\Gamma_1), \Phi; \emptyset; \mathcal{G}(\Delta_1, \Delta_2), \Theta \vdash \mathcal{B}^k_{\widetilde{x}}(V\,(\widetilde{r}, u_i)) \triangleright \diamond}$$

- Now we consider the general case for any $n \geq 1$:

  By Table 4.1 we have:

$$\mathcal{B}^k_{\widetilde{x}}(V\,\widetilde{r}) = c_k?(\widetilde{x}).\, \overbrace{c^{r_1}!\langle \lambda \widetilde{z}_1. \ldots c^{r_n}!\langle \lambda \widetilde{z}_n.}^{n=|\widetilde{r}|}\, Q \rangle \ldots \rangle$$

where $Q = \mathcal{V}_{\widetilde{x}}(V)\,(\widetilde{r}_1, \ldots, \widetilde{r}_n, \widetilde{m})$ with: $\widetilde{z}_i = (z_1^i, \ldots, z^i_{|\mathcal{G}(S_i)|})$ for $i = \{1, \ldots, n\}$; and $\widetilde{m} = (u_i, \ldots, u_{i+|\mathcal{G}(C)|-1})$.

We shall prove the following judgment:

$$\mathcal{G}(\Gamma), \Phi; \emptyset; \mathcal{G}(\Delta_1\Delta_2), \Theta \vdash \mathcal{B}^k_{\widetilde{x}}(V\,(\widetilde{r}, u_i)) \triangleright \diamond \tag{A.77}$$

We construct auxiliary derivations parametrized by k and denoted by $d(k)$. If $k = n$, derivation $d(n)$ is defined as:

$$\text{(Lemma 2.2.2) with } \Phi_2\ \frac{\text{(A.65)}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1), \Theta_1 \vdash \mathcal{V}_{\tilde{x}}(V) \triangleright \mathcal{G}(\widetilde{S}C) \rightarrow \diamond} \quad \text{(A.78)}$$

$$\text{PolyAbs}\ \frac{\text{(A.78)} \qquad \text{(Lemma 2.2.1) with } \sigma\ \dfrac{\text{(A.66)}}{\mathcal{G}(\Gamma), \Phi; \emptyset; \mathcal{G}(\Delta_{\mu_2}) \vdash (\widetilde{r}_1, \dots, \widetilde{r}_n, \widetilde{m}) \triangleright \mathcal{G}(\widetilde{S}C)}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1, \Delta_2),\ \Theta_1, \mathcal{G}(\Delta_{\mu_2}) \vdash \lambda \widetilde{z}_n.\, Q \triangleright \mathcal{G}(S_n) \multimap \diamond} \quad \text{(A.79)}$$

where $\sigma = \prod_{i \in \{1,\dots,n\}} \{\widetilde{n}_i / \widetilde{z}_i\}$ with $\widetilde{n}_i = (r^i_i, \dots, r^i_{i+|\mathcal{G}(S_i)|-1})$ for $i = \{1, \dots, n\}$.

$$\text{Req}\ \frac{\text{(A.79)} \qquad \text{Nil}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash \mathbf{0}} \qquad \text{LVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash c^{r_n} \triangleright \langle \mathcal{G}(S_n) \multimap \diamond \rangle}}{\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1, \Delta_2),\ \Theta_1, \mathcal{G}(\Delta_{\mu_2}) \vdash c^{r_n}!\langle \lambda \widetilde{z}_n.\, Q \rangle.\mathbf{0}} \quad \text{(A.80)}$$

Otherwise, if $k \in \{1, ..., n-1\}$, derivation $d(k)$ is as follows:

$$\text{PolyAbs}\ \frac{d(k+1) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi; \emptyset; \widetilde{z}_k : \mathcal{G}(S_k) \vdash \widetilde{z}_k \triangleright \mathcal{G}(S_k)}}{\begin{array}{c}\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1, \Delta_2), \Theta_1, (\widetilde{z}_1, \dots, \widetilde{z}_{k-1}) : (\mathcal{G}(S_1), \dots, \mathcal{G}(S_{k-1})) \vdash \\ \lambda \widetilde{z}_k.\ \overbrace{c^{r_{k+1}}!\langle \lambda \widetilde{z}_{k+1}. \dots c^{r_n}!\langle \lambda \widetilde{z}_n.\, Q}^{n-k} \overbrace{\rangle.\mathbf{0} \dots \rangle.\mathbf{0}}^{n-k} \triangleright \mathcal{G}(S_1) \multimap \diamond\end{array}} \quad \text{(A.81)}$$

$$\text{Acc}\ \frac{\text{(A.81)} \qquad \mathcal{G}(\Gamma), \Phi; \emptyset; \emptyset \vdash c^{r_1} \triangleright \langle \mathcal{G}(S_1) \multimap \diamond \rangle}{\begin{array}{c}\mathcal{G}(\Gamma), \Phi; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1, \Delta_2), \Theta_1, (\widetilde{z}_1, \dots, \widetilde{z}_{k-1}) : (\mathcal{G}(S_1), \dots, \mathcal{G}(S_{k-1})) \vdash \\ \overbrace{c^{r_k}!\langle \lambda \widetilde{z}_k. \dots c^{r_n}!\langle \lambda \widetilde{z}_r.\, Q}^{n-k+1} \overbrace{\rangle.\mathbf{0} \dots \rangle.\mathbf{0}}^{n-k+1}\end{array}} \quad \text{(A.82)}$$

The following tree proves this case:

$$\text{PolyRcv}\ \frac{d(1) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma_1), \Phi; \mathcal{G}(\Lambda); \emptyset \vdash \widetilde{x} \triangleright \widetilde{M}}}{\mathcal{G}(\Gamma_1), \Phi; \emptyset; \mathcal{G}(\Delta_1, \Delta_2), \Theta \vdash \mathcal{B}^k_{\tilde{x}}(V\,(\widetilde{r}, u_i)) \triangleright \diamond}$$

2. This part concerns values. Without a los of generality we assume $\widetilde{T} = \widetilde{S}, C$ with $\widetilde{S} = (S_1, \dots, S_n)$ such that $\mathsf{tr}(S_i)$ for $i \in \{1, \dots, n\}$. We can distinguish two sub-cases: (i) $V = y$ and (ii) $V = \lambda \widetilde{y}, z.\, P$.

   We first consider sub-case (i). By assumption $\Gamma; \Lambda; \Delta \vdash y \triangleright C \rightsquigarrow \diamond$.

   Further, we can distinguish two sub-sub-cases (a) $\rightsquigarrow = \multimap$ and (b) $\rightsquigarrow = \rightarrow$. In sub-sub-case (a), when $\rightsquigarrow = \multimap$, only Rule LVar can be applied and by inversion $\Lambda = \{y : \widetilde{T} \multimap \diamond\}$ and $\Delta = \emptyset$. By Table 4.1 we have $\mathcal{V}_{\tilde{x}}(y) = y$ and by Definition 4.3.3 and Definition 4.3.10

we have $\mathcal{G}(\Delta) = \{\mathcal{G}(\widetilde{T} \multimap \diamond)\}$. Hence, we prove the following judgment by applying Rule LVar:

$$\mathcal{G}(\Gamma); \mathcal{G}(\Delta); \emptyset \vdash \mathcal{V}_{\tilde{x}}(y) \triangleright \mathcal{G}(\widetilde{T} \multimap \diamond)$$

In sub-sub-case (b) only Rule Sh can be applied and by inversion we have $\Gamma = \{y : \widetilde{T} \to \diamond\}$, $\Lambda = \emptyset$, and $\Delta = \emptyset$. Similarly to (a), by Definition 4.3.3 and Definition 4.3.10 we have $\mathcal{G}(\Gamma) = \{\mathcal{G}(\widetilde{T} \to \diamond)\}$. Hence, we prove the following judgment by applying Rule SH:

$$\mathcal{G}(\Gamma); \emptyset; \emptyset \vdash \mathcal{V}_{\tilde{x}}(y) \triangleright \mathcal{G}(\widetilde{T} \to \diamond)$$

This concludes sub-case (i). Now, we consider sub-case (ii).

This is the second sub-case concerning values when $V = \lambda \widetilde{y}, z.\, P$ where $\widetilde{y} = y_1, \ldots, y_n$. By assumption we have $\Gamma; \Lambda; \Delta, \Delta_\mu \vdash V \triangleright \widetilde{S}, C \leadsto \diamond$. Here we distinguish two sub-sub-cases (a) $\leadsto = \multimap$ and (b) $\leadsto = \to$:

- $\leadsto = \multimap$. By assumption, $\Gamma; \Lambda; \Delta, \Delta_\mu \vdash V \triangleright \widetilde{S}, C \multimap \diamond$. In this case Rule Abs can be applied. Firstly, we α-convert value V as follows:

$$V \equiv_\alpha \lambda \widetilde{y}, z_1.\, P\{z_1/z\} \tag{A.83}$$

For this case only Rule Abs can be applied:

$$\text{Abs}\ \frac{\Gamma; \Lambda; \Delta_1, \Delta_{\mu_1} \vdash P\{z_1/z\} \triangleright \diamond \qquad \Gamma; \emptyset; \Delta_2, \Delta_{\mu_2} \vdash \widetilde{y}, z_1 \triangleright \widetilde{S}, C}{\Gamma \setminus z_1; \Lambda; \Delta_1 \setminus \Delta_2, \Delta_{\mu_1} \setminus \Delta_{\mu_2} \vdash \lambda \widetilde{y}, z_1.\, P\{z_1/z\} \triangleright \widetilde{S}, C \multimap \diamond} \tag{A.84}$$

Let $\widetilde{x} = \mathtt{fv}(P)$ and $\Gamma_1 = \Gamma \setminus \widetilde{x}$. Also, let Θ_1 be a balanced environment such that

$$\mathtt{dom}(\Theta_1) = \{c_1, \ldots, c_{\wr P \wr}\} \cup \{\overline{c_2}, \ldots, \overline{c_{\wr P \wr}}\}$$

and $\Theta_1(c_1) = ?(\widetilde{M});\mathtt{end}$ with $\widetilde{M} = (\mathcal{G}(\Gamma \setminus y_1), \mathcal{G}(\Lambda))(\widetilde{x})$. We define:

$$\Phi_i = \prod_{r \in \mathtt{dom}(\Delta_{\mu_i})} c^r : \langle \mathcal{R}^\star(\Delta_{\mu_i}(r)) \multimap \diamond \rangle \text{ for } i \in \{1, 2\}$$

Then, by IH (Part 1) on the first assumption of (A.84) we have:

$$\mathcal{G}(\Gamma_1), \Phi_1; \emptyset; \mathcal{G}(\Delta_1), \Theta_1 \vdash \mathcal{B}^1_{\tilde{x}}(P\{y_1/y\}) \triangleright \diamond \tag{A.85}$$

Let $\widetilde{T} = \mathcal{G}(S_1), \ldots, \mathcal{G}(S_n), \mathcal{G}(C)$. By Definition 4.3.3 and Definition 4.3.10 and the second assumption of (A.84) we have:

$$\mathcal{G}(\Gamma); \emptyset; \mathcal{G}(\Delta_2) \vdash \widetilde{y}^1, \ldots, \widetilde{y}^n, \widetilde{z} \triangleright \widetilde{T} \tag{A.86}$$

where $\widetilde{z} = (z_1, \ldots, z_{|\mathcal{G}(C)|})$ and $\widetilde{y^i} = (y^i_1, \ldots, y^i_{|\mathcal{G}(S_i)|}))$ for $i \in \{1, \ldots, n\}$.

We define $\Theta = \Theta_1, \overline{c_k} :! \langle \widetilde{M} \rangle$. By Table 4.1, we have:

$$\mathcal{V}_{\tilde{x}}(\lambda y_1, \ldots, y_n, z.\, P) = \lambda(\widetilde{y^1}, \ldots, \widetilde{y^n}, \widetilde{z}) : (\widetilde{T})^{\leadsto}.\, N$$

where

$$N = (\nu\, \widetilde{c})\, (\nu\, \widetilde{c}_r)\ \prod_{i \in |\widetilde{y}|} (c^{y_i}?(x).x\, \widetilde{y}^i) \mid \overline{c_1}!\langle \widetilde{x} \rangle \mid \mathcal{B}^1_{\tilde{x}}(P\{z_1/z\})$$

with $\widetilde{c} = (c_1, \dots, c_{\wr P \int})$ and $\widetilde{c_r} = \bigcup_{r \in \tilde{y}} c^r$.

We use an auxiliary derivation:

$$\text{Send}\ \dfrac{\text{End}\ \dfrac{\text{Nil}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi_1; \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond}}{\mathcal{G}(\Gamma), \Phi_1; \emptyset; c_k : \mathsf{end} \vdash \mathbf{0} \triangleright \diamond} \qquad \text{PolyVar}\ \dfrac{}{\mathcal{G}(\Gamma), \Phi_1; \mathcal{G}(\Lambda); \emptyset \vdash \widetilde{x} \triangleright \widetilde{M}}}{\mathcal{G}(\Gamma), \Phi_1; \mathcal{G}(\Lambda); \overline{c_1} :!\langle \widetilde{M} \rangle;\mathsf{end} \vdash \overline{c_1}!\langle \widetilde{x} \rangle.\mathbf{0} \triangleright \diamond} \tag{A.87}$$

$$\text{LVar}\ \dfrac{}{\begin{array}{c}\mathcal{G}(\Gamma), \Phi_1; x : \mathcal{R}^\star(\Delta_{\mu_2}(y^i)) \multimap \diamond; \emptyset \vdash \\ x \triangleright \mathcal{R}^\star(\Delta_{\mu_2}(y^i)) \multimap \diamond\end{array}} \tag{A.88}$$

$$\text{PolyApp}\ \dfrac{(A.88) \qquad \text{PolySess}\ \dfrac{}{\begin{array}{c}\mathcal{G}(\Gamma), \Phi_1; \emptyset; \widetilde{y}^i : \mathcal{R}^\star(\Delta_{\mu_2}(y^i)) \vdash \\ \widetilde{y}^i \triangleright \mathcal{R}^\star(\Delta_{\mu_2}(y^i))\end{array}}}{\mathcal{G}(\Gamma), \Phi_1; x : \mathcal{R}^\star(\Delta_{\mu_2}(y^i)) \multimap \diamond; \widetilde{y}^i : \mathcal{R}^\star(\Delta_{\mu_2}(y^i)) \vdash (b\, \widetilde{y}^i)} \tag{A.89}$$

$$\text{Acc}\ \dfrac{(A.89) \qquad \text{Sh}\ \dfrac{}{\begin{array}{c}\mathcal{G}(\Gamma), \Phi_1; \emptyset; \emptyset \vdash c^{y^i} \triangleright \\ \langle \mathcal{R}^\star(\Delta_\mu(y^i)) \multimap \diamond \rangle\end{array}} \qquad \text{LVar}\ \dfrac{}{\begin{array}{c}\mathcal{G}(\Gamma),\ \Phi_1; x : \mathcal{R}^\star(\Delta_\mu(y^i)) \multimap \diamond; \emptyset \\ \vdash x \triangleright \mathcal{R}^\star(\Delta_\mu(y^i)) \multimap \diamond\end{array}}}{\mathcal{G}(\Gamma\sigma), \Phi_1; \emptyset; \widetilde{y}^i : \mathcal{R}^\star(\Delta_{\mu_1}(y^i)) \vdash c^{y_i}?(x).x\, \widetilde{y}^i} \tag{A.90}$$

$$\text{Par } (|\widetilde{y}| - 1 \text{ times})\ \dfrac{\text{for } y^i \in \widetilde{y} \qquad (A.90)}{\mathcal{G}(\Gamma), \Phi_1; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1), \mathcal{G}(\Delta_{\mu_2}) \vdash \prod_{i \in |\widetilde{y}|} (c^{y_i}?(x).x\, \widetilde{y}^i) \triangleright \diamond} \tag{A.91}$$

$$\text{PolyResS}\ \dfrac{\text{PolyRes}\ \dfrac{(A.91) \qquad \text{Par}\ \dfrac{(A.87) \qquad (\text{Lemma 2.2.2) with } \tilde{x}\ \dfrac{(A.85)}{\mathcal{G}(\Gamma), \Phi_1; \emptyset; \mathcal{G}(\Delta_1), \Theta_1 \vdash \mathcal{B}^1_{\tilde{x}}(P\{z_1/z\}) \triangleright \diamond}}{\mathcal{G}(\Gamma), \Phi_1; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1), \Theta \vdash \overline{c_1}!\langle \widetilde{x} \rangle \mid \mathcal{B}^1_{\tilde{x}}(P\{z_1/z\}) \triangleright \diamond}}{\begin{array}{c}\mathcal{G}(\Gamma), \Phi_1; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1), \mathcal{G}(\Delta_{\mu_2}), \Theta \vdash \\ \prod_{i \in |\widetilde{y}|} (c^{y_i}?(x).x\, \widetilde{y}^i) \mid \overline{c_1}!\langle \widetilde{x} \rangle \mid \mathcal{B}^1_{\tilde{x}}(P\{z_1/z\}) \triangleright \diamond\end{array}}}{\begin{array}{c}\mathcal{G}(\Gamma), \Phi_1 \setminus \Phi_2; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1), \mathcal{G}(\Delta_{\mu_2}), \Theta \vdash \\ (\nu\, \widetilde{c}_r) \prod_{i \in |\widetilde{y}|} (c^{y_i}?(x).x\, \widetilde{y}^i) \mid \overline{c_1}!\langle \widetilde{x} \rangle \mid \mathcal{B}^1_{\tilde{x}}(P\{z_1/z\}) \triangleright \diamond\end{array}}$$

$$\mathcal{G}(\Gamma), \Phi_1 \setminus \Phi_2; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1), \mathcal{G}(\Delta_{\mu_2}) \vdash N \triangleright \diamond \tag{A.92}$$

The following tree proves this part:

$$\text{Abs}\ \dfrac{(A.92) \qquad (A.86)}{\mathcal{G}(\Gamma \setminus z_1), \Phi_1 \setminus \Phi_2; \mathcal{G}(\Lambda); \mathcal{G}(\Delta_1 \setminus \Delta_2) \vdash \lambda(\widetilde{y^1}, \dots, \widetilde{y^n}, \widetilde{z}) : (\widetilde{T})^{\multimap}.\, N \triangleright \diamond} \tag{A.93}$$

- $\rightsquigarrow = \rightarrow$. By assumption, $\Gamma; \emptyset; \emptyset \vdash V \triangleright C \rightarrow \diamond$. In this case Rule Prom can be applied:

$$\text{Prom}\ \dfrac{\text{Abs}\ \dfrac{\Gamma; \emptyset; \Delta \vdash P\{y_1/y\} \triangleright \diamond \qquad \Gamma; \emptyset; \Delta \vdash y_1 \triangleright C}{\Gamma \setminus z_1; \emptyset; \emptyset \vdash \lambda \widetilde{y}, z_1.\, P\{z_1/z\} \triangleright C \multimap \diamond}}{\Gamma \setminus z_1; \emptyset; \emptyset \vdash \lambda \widetilde{y}, z_1.\, P\{z_1/z\} \triangleright C \rightarrow \diamond} \tag{A.94}$$

Now, we can see that we can specialize previous sub-case by taking $\Delta_1 \backslash \Delta_2 = \emptyset$ (resp. $\Delta_{\mu 1} \setminus \Delta_{\mu 2} = \emptyset$), that is $\Delta_1 = \Delta_2$ (resp. $\Delta_{\mu 1} = \Delta_{\mu 2}$) and $\Lambda = \emptyset$. Subsequently, we have $\Phi_1 \setminus \Phi_2 = \emptyset$, $\mathcal{G}(\Lambda) = \emptyset$, and $\mathcal{G}(\Delta_1 \setminus \Delta_2) = \emptyset$. Thus, we can apply Rule Prom to (A.93) to prove this sub-case as follows:

$$\textsc{Prom}\ \dfrac{(\text{A.93})}{\mathcal{G}(\Gamma \setminus z_1); \emptyset; \emptyset \vdash \lambda(\widetilde{y^1}, \ldots, \widetilde{y^n}, \widetilde{z}) : (\widetilde{T})^{\rightarrow}.\, N \triangleright \diamond}$$

This concludes this part (and the proof).

$\square$

A.1.2 Proof of Theorem 4.3.1

Theorem 4.3.1 (Static Correctness)**.** *Let P be a closed* HO *process (i.e.* $\mathtt{fv}(P) = \emptyset$*) with $\widetilde{u} = \mathtt{fn}(P)$. If $\Gamma; \emptyset; \Delta \circ \Delta_\mu \vdash P \triangleright \diamond$, then $\mathcal{G}(\Gamma\sigma); \emptyset; \mathcal{G}(\Delta\sigma), \mathcal{G}(\Delta_\mu\sigma) \vdash \mathcal{D}(P) \triangleright \diamond$, where $\sigma = \{\mathsf{init}(\widetilde{u})/\widetilde{u}\}$.*

Proof. By assumption $\Gamma; \emptyset; \Delta \circ \Delta_\mu \vdash P \triangleright \diamond$. Then, by applying Lemma 2.2.1 we have:

$$\Gamma\sigma; \emptyset; \Delta\sigma \circ \Delta_\mu\sigma \vdash P\sigma \triangleright \diamond \tag{A.95}$$

By Lemma 4.3.1 on (A.95) we have:

$$\mathcal{G}(\Gamma_1\sigma), \Phi; \emptyset; \mathcal{G}(\Delta\sigma), \Theta \vdash \mathcal{B}^k_\epsilon(P\sigma) \triangleright \diamond \tag{A.96}$$

where Θ is balanced with $\mathtt{dom}(\Theta) = \{c_k, c_{k+1}, \ldots, c_{k+\llparenthesis P \rrparenthesis-1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\llparenthesis P \rrparenthesis-1}}\}$, and $\Theta(c_k) = ?(\cdot)$, and $\Phi = \prod_{r \in \mathtt{dom}(\Delta_\mu)} c^r : \langle \mathcal{R}^\star(\Delta_\mu(r)) {\multimap} \diamond \rangle$. By assumption, $\mathtt{fv}(P) = \emptyset$.
By Definition 4.3.9, we shall prove the following judgment:

$$\mathcal{G}(\Gamma\sigma); \emptyset; \mathcal{G}(\Delta\sigma) \circ \mathcal{G}(\Delta_\mu\sigma) \vdash (\nu\, \widetilde{c})\, (\nu\, \widetilde{c}_r) \Big(\prod_{r \in \widetilde{v}} c^r?(x).x\, \widetilde{r} \mid \overline{c_k}!\langle\rangle.\mathbf{0} \mid \mathcal{B}^k_\epsilon(P\sigma) \Big)$$

where: $k > 0$; $\widetilde{v} = \mathtt{rn}(P)$; $\widetilde{r} = (r_1, \ldots, r_{|\mathcal{G}(S)|})$ for each $r \in \widetilde{v}$.
We know $\mathtt{dom}(\Delta_\mu) = \widetilde{v}$. We assume that recursive session types are unfolded. By Definition 4.3.10 and Definition 4.3.3, for $r \in \mathtt{dom}(\Delta_\mu)$ we have:

$$\mathcal{G}(\Delta_\mu)(r) = \mathcal{R}(\Delta_\mu(r)) = \mathcal{R}^\star(\Delta_\mu(r))$$

We use a family of auxiliary derivations parametrized by $r \in \widetilde{v}$.

$$\textsc{PolyApp}\ \dfrac{\textsc{LVar}\ \dfrac{}{\begin{array}{c}\mathcal{G}(\Gamma\sigma), \Phi; x : \mathcal{R}^\star(\Delta_\mu(r)) {\multimap} \diamond; \emptyset \vdash \\ x \triangleright \mathcal{R}^\star(\Delta_\mu(r)) {\multimap} \diamond\end{array}} \qquad \textsc{PolySess}\ \dfrac{}{\begin{array}{c}\mathcal{G}(\Gamma\sigma), \Phi; \emptyset; \widetilde{r} : \mathcal{R}^\star(\Delta_\mu(r)) \vdash \\ \widetilde{r} \triangleright \mathcal{R}^\star(\Delta_\mu(r))\end{array}}}{\mathcal{G}(\Gamma\sigma), \Phi; x : \mathcal{R}^\star(\Delta_\mu(r)) {\multimap} \diamond; \widetilde{r} : \mathcal{R}^\star(\Delta_\mu(r)) \vdash (x\, \widetilde{r})} \tag{A.97}$$

$$\textsc{Acc}\ \dfrac{(\text{A.97}) \qquad \textsc{Sh}\ \dfrac{}{\begin{array}{c}\mathcal{G}(\Gamma\sigma), \Phi; \emptyset; \emptyset \vdash c^r \triangleright \\ \langle \mathcal{R}^\star(\Delta_\mu(r)) {\multimap} \diamond \rangle\end{array}} \qquad \textsc{LVar}\ \dfrac{}{\begin{array}{c}\mathcal{G}(\Gamma\sigma), \Phi; x : \mathcal{R}^\star(\Delta_\mu(r)) {\multimap} \diamond; \emptyset \\ \vdash x \triangleright \mathcal{R}^\star(\Delta_\mu(r)) {\multimap} \diamond\end{array}}}{\mathcal{G}(\Gamma\sigma), \Phi; \emptyset; \widetilde{r} : \mathcal{R}^\star(\Delta_\mu(r)) \vdash c^r?(x).x\, \widetilde{r}} \tag{A.98}$$

We will then use:

$$\text{Par } (|\widetilde{v}|-1 \text{ times})\ \dfrac{\text{for } r \in \widetilde{v} \qquad (\text{A.98})}{\mathcal{G}(\Gamma\sigma), \Phi; \emptyset; \mathcal{G}(\Delta_\mu\sigma) \vdash \prod_{r\in\widetilde{v}} c^r?(x).x\,\widetilde{r}} \tag{A.99}$$

where we apply Rule Par $|\widetilde{v}|-1$ times and for every $r \in \widetilde{v}$ we apply derivation (A.98). Notice that by Definition 4.3.10 and Definition 4.3.3 we have $\mathcal{G}(\Delta_\mu\sigma) = \prod_{r\in\widetilde{v}} \widetilde{r} : \mathcal{R}^\star(\Delta_\mu(r))$.

$$\text{Par}\ \dfrac{\text{Sess}\ \dfrac{}{\mathcal{G}(\Gamma\sigma), \Phi; \emptyset; \overline{c_k} :!\langle\cdot\rangle;\mathtt{end} \vdash \overline{c_k}!\langle\rangle.\mathbf{0}} \qquad (\text{A.96})}{\mathcal{G}(\Gamma\sigma), \Phi; \emptyset; \mathcal{G}(\Delta\sigma), \Theta, \overline{c_k} :!\langle\cdot\rangle;\mathtt{end} \vdash \overline{c_k}!\langle\rangle.\mathbf{0} \mid \mathcal{B}^k_\epsilon(P\sigma)} \tag{A.100}$$

The following tree proves this case:

$$\text{PolyResS}\ \dfrac{\text{PolyRes}\ \dfrac{\text{Par}\ \dfrac{(\text{A.99}) \qquad (\text{A.100})}{\begin{array}{c}\mathcal{G}(\Gamma\sigma), \Phi; \emptyset; \mathcal{G}(\Delta\sigma), \Theta, \overline{c_k} :!\langle\cdot\rangle;\mathtt{end} \circ \mathcal{G}(\Delta_\mu\sigma) \vdash \\ \prod_{r\in\widetilde{v}}(c^r?(x).x\,\widetilde{r}) \mid \overline{c_k}!\langle\rangle.\mathbf{0} \mid \mathcal{B}^k_\epsilon(P\sigma)\end{array}}}{\begin{array}{c}\mathcal{G}(\Gamma\sigma); \emptyset; \mathcal{G}(\Delta\sigma), \Theta, \overline{c_k} :!\langle\cdot\rangle;\mathtt{end} \circ \mathcal{G}(\Delta_\mu\sigma) \vdash \\ (\nu\,\widetilde{c}_r)\,(\prod_{r\in\widetilde{v}}(c^r?(x).x\,\widetilde{z}) \mid \overline{c_k}!\langle\rangle.\mathbf{0} \mid \mathcal{B}^k_\epsilon(P\sigma))\end{array}}}{\mathcal{G}(\Gamma\sigma); \emptyset; \mathcal{G}(\Delta\sigma) \circ \mathcal{G}(\Delta_\mu\sigma) \vdash (\nu\,\widetilde{c})\,(\nu\,\widetilde{c}_r)\,(\prod_{r\in\widetilde{v}}(c^r?(x).x\,\widetilde{r}) \mid \overline{c_k}!\langle\rangle.\mathbf{0} \mid \mathcal{B}^k_\epsilon(P\sigma))} \tag{A.101}$$

$\square$

A.2 Appendix to Section 4.4

A.2.1 Proof of Lemma 4.4.3

Lemma 4.4.3. *Given an indexed process $P_1\{\tilde{W}/\tilde{x}\}$, the set $\mathcal{C}^{\tilde{W}}_{\tilde{x}}(P_1)$ is closed under τ-transitions on non-essential prefixes. That is, if $R_1 \in \mathcal{C}^{\tilde{W}}_{\tilde{x}}(P_1)$ and $R_1 \xrightarrow{\tau} R_2$ is inferred from the actions on non-essential prefixes, then $R_2 \in \mathcal{C}^{\tilde{W}}_{\tilde{x}}(P_1)$.*

Proof. By the induction on the structure of P_1. We consider two base cases:

- Case $P_1 = \mathbf{0}$. Let $\widetilde{B}$ such that $\widetilde{W} \bowtie \widetilde{B}$. Then, the elements of $\mathcal{C}^{\tilde{W}}_{\tilde{x}}(\mathbf{0})$ are $R_1 = (\nu\, c_k)\, \overline{c_k}!\langle \widetilde{B}\rangle \mid c_k?(\widetilde{x}).\mathbf{0}$ and $R_2 = \mathbf{0}$. Clearly, $R_1 \xrightarrow{\tau} R_2$.

- Case $P_1 = V_1\,(\widetilde{r}, u)$. Let $n = |\widetilde{r}|$. Then, we have

$$\mathcal{C}^{\tilde{W}}_{\tilde{x}}(P_1) = N_1 \cup N_4 \cup \big(\cup_{1 \le l \le n} N_2^l\big) \cup \big(\cup_{1 \le l \le n-1} N_3^l\big)$$

  where

$$N_1 = \{\mathcal{R}_{\tilde{v},\tilde{r}} \mid \overline{c_k}!\langle \widetilde{B}\rangle \mid \mathcal{B}^k_{\tilde{x}}(P_1) : \widetilde{W} \bowtie \widetilde{B}\}$$

$$N_2^l = \{\mathcal{R}_{\tilde{v},r_l,\ldots,r_n} \mid \overbrace{c^{r_l}!\langle \lambda \widetilde{z}_l.c^{r_{l+1}}!\langle \lambda\widetilde{z}_{l+1}.\cdots .c^{r_n}!\langle \lambda \widetilde{z}_n.}^{|\tilde{r}|-l+1} Q_l^{V_2}\rangle\,\rangle\rangle : V_1\{\tilde{W}/\tilde{x}\} \bowtie V_2\}$$

$$N_3^l = \{\mathcal{R}_{\tilde{v},r_{l+1},\ldots,r_n} \mid \lambda \widetilde{z}_l.\, \overbrace{c^{r_{l+1}}!\langle \lambda\widetilde{z}_{l+1}.\cdots .c^{r_n}!\langle \lambda \widetilde{z}_n.}^{|\tilde{r}|-l} Q_l^{V_2}\rangle\,\rangle\; \widetilde{r}_l : V_1\{\tilde{W}/\tilde{x}\} \bowtie V_2\}$$

$$N_4 = \{\mathcal{R}_{\tilde{v}} \mid V_2\,(\widetilde{r}_1, \ldots, \widetilde{r}_n, \widetilde{m}) : V_1\{\tilde{W}/\tilde{x}\} \bowtie V_2\}$$

  with

$$Q_l^{V_2} = V_2\,(\widetilde{r}_1, \ldots, \widetilde{r}_{l-1}, \widetilde{z}_l, \ldots, \widetilde{z}_n, \widetilde{m})$$

  We can see that for $R_1 \in N_1$ there exist $R_2^0 \in N_2^0$ such that $R_1 \xrightarrow{\tau} R_2^0$. Further, we could see that for $R_2^l \in N_2^l$ there is $R_3^l \in N_3^l$ such that $R_2^l \xrightarrow{\tau} R_3^l$. Now, we can see that for $R_3^l \in N_3^l$ there is $R_2^{l+1} \in N_2^{l+1}$ such that $R_3^l \xrightarrow{\tau} R_2^{l+1}$. Finally, we have for

$$R_3^n \in N_3^n = \{\mathcal{R}_{\tilde{v}} \mid \lambda \widetilde{z}_n.\, Q_n^{V_2}\, \widetilde{r}_n : V_1\{\tilde{W}/\tilde{x}\} \bowtie V_2\}$$

  there is $R_4 \in N_4$ such that $R_3^n \xrightarrow{\tau} R_4$.

We consider two inductive cases as remaining cases are similar:

- Case $P_1 = u_i!\langle V_1\rangle.P_2$. We distinguish two sub-case: (i) $\neg\mathtt{tr}(u_i)$ and (ii) $\mathtt{tr}(u_i)$. In both sub-cases, we distinguish two kinds of an object value V_1: (a) $V_1 \equiv x$, such that $\{V_x/x\} \in \{\tilde{W}/\tilde{x}\}$ and (b) $V_1 = \lambda y : C.\, P'$, that is V_1 is a pure abstraction.

  First, we consider sub-case (i). Let $\widetilde{y} = \mathtt{fv}(V_1)$, $\widetilde{w} = \mathtt{fv}(P_2)$, $\widetilde{W_1}$, and $\widetilde{W_2}$ such that $\{\widetilde{W}/\widetilde{x}\} = \{\widetilde{W_1}/\widetilde{y}\} \cdot \{\widetilde{W_2}/\widetilde{w}\}$. Further, let $\widetilde{v} = \mathtt{rn}(P_1\{\widetilde{W}/\widetilde{x}\})$ and $\sigma = \{u_{i+1}/u_i\}$. Then, by the definition of $\mathcal{C}^{-}_{-}(\,-\,)$ (Table 4.3), we have that $\mathcal{C}^{\tilde{W}}_{\tilde{x}}(u_i!\langle V_1\rangle.P_2) = N_1 \cup N_2$ where:

$$N_1 = \{\mathcal{R}_{\tilde{v}} \mid \overline{c_k}!\langle \widetilde{B}\rangle \mid \mathcal{B}^k_{\tilde{x}}(P_1) : \widetilde{W} \bowtie \widetilde{B}\}$$

$$N_2 = \{\mathcal{R}_{\tilde{v}} \mid u_i!\langle V_2\rangle.\overline{c_k}!\langle \widetilde{B}_2\rangle \mid \mathcal{B}^{k_2}_{\tilde{w}}(P_2\sigma) : V_1\sigma\{\widetilde{W_1}/\widetilde{y}\} \bowtie V_2,\ \widetilde{W_2} \bowtie \widetilde{B}_2\}$$

  We can see that in both cases of V_1, a variable or a pure abstraction, we have that for $R_1 \in N_1$ there is $R_2 \in N_2$ such that $R_1 \xrightarrow{\tau} R_2$. In the sub-case (a) by $\{V_x/x\} \in \{\tilde{W}/\tilde{x}\}$

we have $V_2 \in \widetilde{B}$, such that $V_x \bowtie V_2$, that by τ-move substitutes x in $\mathcal{B}^k_{\tilde{x}}(P_1)$. Thus, by $x\sigma\{\tilde{W}_1/\tilde{y}\} = V_x$ we have $V_1\sigma\{\tilde{W}_1/\tilde{y}\} \bowtie V_2$. In the sub-case (b), by definition of $\mathcal{B}^-_-(-)$ we have that by τ-move $\widetilde{B}_1 \subseteq \widetilde{B}$ substitute $\tilde{y}$ in $\mathcal{V}_{\tilde{y}}(V_1\sigma)$ so we have $R_1 \xrightarrow{\tau} R_2$ where

$$V_2 = \mathcal{V}_{\tilde{y}}(V_1\sigma)\{\tilde{B}_1/\tilde{y}\}$$

Now, we may notice that by Table 4.3 we have

$$V_2 \in \mathcal{C}^{\tilde{W}_1}_{\tilde{y}}(V_1\sigma)$$

Hence, by Definition 4.4.11 and Definition 4.4.13 we have $V_1\sigma\{\tilde{W}_1/\tilde{y}\} \bowtie V_2$. Further, we may notice that there is no τ-transition involving non-essential prefixes in R_2. This concludes this sub-case.

Now, we consider sub-case (ii). Let $\tilde{y} = \mathtt{fv}(V_1)$, $\tilde{w} = \mathtt{fv}(P_2)$, $\widetilde{W}_1$, and $\widetilde{W}_2$ such that $\{\widetilde{W}/\tilde{x}\} = \{\widetilde{W}_1/\tilde{y}\} \cdot \{\widetilde{W}_2/\tilde{w}\}$. Then, we have $\mathcal{C}^{\widetilde{W}}_{\tilde{x}}(P_1) = N_1 \cup N_2 \cup N_3 \cup N_4$ where

$$N_1 = \{\mathcal{R}_{\tilde{v}} \mid \overline{c_k}!\langle\widetilde{B}\rangle \mid \mathcal{B}^k_{\tilde{x}}(P_1) : \widetilde{W} \bowtie \widetilde{B}\}$$
$$N_2 = \{\mathcal{R}_{\tilde{v}} \mid c^u!\langle M^{\tilde{B}_2}_{V_2}\rangle \mid \mathcal{B}^k_{\tilde{w}}(P_2) : V_1\{\tilde{W}_1/\tilde{y}\} \bowtie V_2,\ \widetilde{W}_2 \bowtie \widetilde{B}_2\}$$
$$N_3 = \{\mathcal{R}_{\tilde{v}\setminus u} \mid M^{\tilde{B}_2}_{V_2}\,\tilde{u} \mid \mathcal{B}^k_{\tilde{w}}(Q) : V_1\{\tilde{W}_1/\tilde{y}\} \bowtie V_2,\ \widetilde{W}_2 \bowtie \widetilde{B}_2\}$$
$$N_4 = \{\mathcal{R}_{\tilde{v}\setminus u} \mid u_{[S\rangle}!\langle V_2\rangle.\overline{c_k}!\langle\widetilde{B}_2\rangle.c^u?(b).(b\,\tilde{u}) \mid \mathcal{B}^k_{\tilde{w}}(P_2) : V_1\{\tilde{W}_1/\tilde{y}\} \bowtie V_2,\ \widetilde{W}_2 \bowtie \widetilde{B}_2\}$$

where

$$M^{\tilde{B}_2}_{V_2} = \lambda\tilde{z}.\, z_{[S\rangle}!\langle V_2\rangle.\overline{c_k}!\langle\widetilde{B}_2\rangle \mid c^u?(b).(b\,\tilde{z})$$

As in the previous sub-case, we can see that in both cases of V_1, a variable or a pure abstraction, we have that for $R_1 \in N_1$ there is $R_2 \in N_2$ such that $R_1 \xrightarrow{\tau} R_2$, for appropriate choice of $\widetilde{B}$, $\widetilde{B}_2$, and k. Similarly, by the communication on shared name c^u for $R_2 \in N_2$ there is $R_3 \in N_3$ such that $R_2 \xrightarrow{\tau} R_3$. Finally, for $R_3 \in N_3$ there is $R_4 \in N_4$ such that $R_3 \xrightarrow{\tau} R_4$ by the application. This concludes output case.

- Case $P_1 = Q_1 \mid Q_2$. Let $\tilde{y} = \mathtt{fv}(Q_1)$, $\tilde{w} = \mathtt{fv}(Q_2)$, $\widetilde{W}_1$, and $\widetilde{W}_2$ such that $\{\widetilde{W}/\tilde{x}\} = \{\tilde{W}_1/\tilde{y}\} \cdot \{\tilde{W}_2/\tilde{w}\}$. Then, by the definition of $\mathcal{C}^-_-(\,-\,)$ (Table 4.3), we have that $\mathcal{C}^{\widetilde{W}}_{\tilde{x}}(Q_1 \mid Q_2) = N_1 \cup N_2 \cup N_3$ where:

$$N_1 = \{\overline{c_k}!\langle\widetilde{B}\rangle \mid \mathcal{B}^k_{\tilde{x}}(Q_1 \mid Q_2) : \widetilde{W} \bowtie \widetilde{B}\}$$
$$N_2 = \{\overline{c_k}!\langle\widetilde{B}_1\rangle.\overline{c_{k+l}}!\langle\widetilde{B}_2\rangle \mid \mathcal{B}^k_{\tilde{y}}(Q_1) \mid \mathcal{B}^{k+l}_{\tilde{w}}(Q_2) : \widetilde{W}_1 \bowtie \widetilde{B}_1,\ \widetilde{W}_2 \bowtie \widetilde{B}_2\}$$
$$N_3 = \{R_1 \mid R_2 : R_1 \in \mathcal{C}^{\tilde{W}_1}_{\tilde{y}}(Q_1),\ R_2 \in \mathcal{C}^{\tilde{W}_2}_{\tilde{w}}(Q_2)\}$$

with $l = \wr Q_1 \int$.

We show that the thesis holds for processes in each of these three sets:

1. Clearly, by picking appropriate $\widetilde{B}$, $\widetilde{B}_1$, and $\widetilde{B}_2$, for any $R^1_1 \in N_1$ there is R^1_2 such that $R^1_1 \xrightarrow{\tau} R^1_2$ and $R^1_2 \in N_2$.
2. Now, we consider set N_3. Let us pick $R^3 = R_1 \mid R_2 \in N_3$, for some $R_1 \in \mathcal{C}^{\tilde{W}_1}_{\tilde{y}}(Q_1)$ and $R_2 \in \mathcal{C}^{\tilde{W}_2}_{\tilde{w}}(Q_2)$. By the definition of $\mathcal{C}^-_-(-)$ (Table 4.3) we know all propagator names are restricted element-wise in R^3, and so there is no communication between R_1 and R_2 on propagator prefixes. This ensures that any τ-actions emanating from R^3 arise from R_1 or R_2 separately, not from their interaction. The thesis then follows by IH, for we know that if $R_1 \xrightarrow{\tau} R'_1$ then $R'_1 \in \mathcal{C}^{\tilde{W}_1}_{\tilde{y}}(Q_1)$; similarly, if $R_2 \xrightarrow{\tau} R'_2$ then $R'_2 \in \mathcal{C}^{\tilde{W}_2}_{\tilde{w}}(Q_2)$. Thus, by the definition $R'_1 \mid R'_2 \in N_3$.

3. Finally, we show that for any $R_2 \in N_2$ if $R_2 \xrightarrow{\tau} R_2'$ then $R_2' \in \mathcal{C}_{\tilde{x}}^{\widetilde{W}}(Q_1 \mid Q_2)$. We know that $R_2 = \overline{c_k}!\langle \widetilde{B}_1 \rangle.\overline{c_{k+l}}!\langle \widetilde{B}_2 \rangle \mid \mathcal{B}_{\tilde{y}}^{k}(Q_1) \mid \mathcal{B}_{\tilde{w}}^{k+l}(Q_2)$, where $\widetilde{B}_i$ are such that $\widetilde{W}_i \bowtie \widetilde{B}_i$ for $i \in \{1,2\}$. Clearly, by a synchronization on c_k, we have

$$R_2 \xrightarrow{\tau} \overline{c_{k+l}}!\langle \widetilde{B}_2 \rangle \mid B_{Q_1} \mid \mathcal{B}_{\tilde{w}}^{k+l}(Q_2) = R_2'$$

where B_{Q_1} stands for the derivative of $\mathcal{B}_{\tilde{y}}^{k}(Q_1)$ after the synchronization (and substitution of $\widetilde{B}_1$).

To show that R_2' is already in $\mathcal{C}_{\tilde{x}}^{\widetilde{W}}(Q_1 \mid Q_2)$, we consider an $R^3 \in N_3$ such that

$$R^3 = \overline{c_k}!\langle \widetilde{B}_1 \rangle \mid \mathcal{B}_{\tilde{y}}^{k}(Q_1) \mid \overline{c_{k+l}}!\langle \widetilde{B}_2 \rangle \mid \mathcal{B}_{\tilde{w}}^{k+l}(Q_2)$$

Note that there is a τ-transition on c_k such that $R^3 \xrightarrow{\tau} R_2'$. Because processes in N_3 satisfy the thesis (cf. the previous sub-sub-case), we have that $R_2' \in N_3$. Therefore, $R_2' \in \mathcal{C}_{\tilde{x}}^{\widetilde{W}}(Q_1 \mid Q_2)$, as desired. This concludes parallel composition case.

This concludes the proof. $\square$

A.2.2 Proof of Lemma 4.4.4

For the proof we will use the following syntactic sugar.

Definition A.2.1 (Function $\widehat{\mathcal{C}}_{-}^{-}(-)$). Let P be a HO process, ρ be a values substitution, and σ be an indexed name substitution. We define $\widehat{\mathcal{C}}_{\sigma}^{\rho}(P)$ as follows:

$$\widehat{\mathcal{C}}_{\sigma}^{\rho}(P_1) = \mathcal{C}_{\tilde{x}}^{\widetilde{W}\sigma}(P_1\sigma) \qquad \text{with } \rho = \{\widetilde{W}/\tilde{x}\}$$

Lemma 4.4.4. *Assume $P_1\{\widetilde{W}/\tilde{x}\}$ is a process such that $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\widetilde{W}/\tilde{x}\} \triangleright \diamond$ with $\mathsf{balanced}(\Delta_1)$ and $P_1\{\widetilde{W}/\tilde{x}\}\,\mathcal{S}\,Q_1$.*

1. *Whenever $P_1\{\widetilde{W}/\tilde{x}\} \xrightarrow{(\nu\,\widetilde{m}_1)\,n!\langle V_1\rangle} P_2$, such that $\overline{n} \notin \mathtt{fn}(P_1\{\widetilde{W}/\tilde{x}\})$, then there exist Q_2 and V_2 such that $Q_1 \xRightarrow{(\nu\,\widetilde{m}_2)\,\breve{n}!\langle V_2\rangle} Q_2$ and, for a fresh t,*

$$(\nu\,\widetilde{m}_1)(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{\widetilde{W}/\tilde{x}\}\,\mathcal{S}\,(\nu\,\widetilde{m}_2)(Q_2 \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2)$$

2. *Whenever $P_1\{\widetilde{W}/\tilde{x}\} \xrightarrow{n?(V_1)} P_2$, such that $\overline{n} \notin \mathtt{fn}(P_1\{\widetilde{W}/\tilde{x}\})$, then there exist Q_2, V_2, and σ such that $Q_1 \xRightarrow{\breve{n}?(V_2)} Q_2$ where $V_1\sigma \bowtie V_2$ and $P_2\,\mathcal{S}\,Q_2$,*

3. *Whenever $P_1 \xrightarrow{\tau} P_2$ then there exists Q_2 such that $Q_1 \xRightarrow{\tau} Q_2$ and $P_2\,\mathcal{S}\,Q_2$.*

Proof. By transition induction. Let $\rho_1 = \{\widetilde{W}/\tilde{x}\}$. By inversion of $P_1\rho_1\,\mathcal{S}\,Q_1$ we know there is $\sigma_1 \in \mathsf{index}(\mathtt{fn}(P\rho_1))$ such that $Q_1 \in \widehat{\mathcal{C}}_{\sigma_1}^{\rho_1}(P_1)$. Then, we need the following assertion on the index substitution. If $P_1\rho_1 \xrightarrow{\ell} P_2\rho_2$ and $\mathtt{subj}(\ell) = n$ then there exists Q_2 such that $Q_1 \xrightarrow{\breve{\ell}} Q_2$ with $\mathtt{subj}(\breve{\ell}) = n_i$ and $Q_2 \in \widehat{\mathcal{C}}_{\sigma_2}^{\rho_2}(P_2)$ such that $\sigma_2 \in \mathsf{index}(P_2\rho_2)$, $\mathsf{next}(n_i) \in \sigma_2$, and $\sigma_1 \cdot (\sigma_2 \setminus \mathsf{next}(n_i)) = (\sigma_2 \setminus \mathsf{next}(n_i)) \cdot \sigma_1$.

First, we consider three base cases: Rules $\mathtt{Snd}$, $\mathtt{Rv}$, and $\mathtt{App}$. Then, we distinguish five inductive cases and analyze three cases (as cases $\mathtt{Par}_R$ and $\mathtt{Par}_L$, and $\mathtt{New}$ and $\mathtt{Res}$ are similar). Thus, in total we consider six cases:

1. Case $\langle \mathtt{Snd} \rangle$. Then $P_1 = n!\langle V_1 \rangle.P_2$. We first consider the case when P_1 is not a trigger collection, and then briefly discuss the case when it is a trigger collection.

   We distinguish two sub-cases: (i) $\neg\mathsf{tr}(n)$ and (ii) $\mathsf{tr}(n)$. In both sub-cases, we distinguish two kinds of an object value V_1: (a) $V_1 \equiv x$, such that $\{{}^{V_x}/x\} \in \{{}^{\tilde{W}}/\tilde{x}\}$ and (b) $V_1 = \lambda y : C.\, P'$, that is V_1 is a pure abstraction. Next, we consider two sub-cases:

   i) Sub-case $\neg\mathsf{tr}(n)$. Let $\widetilde{W_1}$, $\widetilde{W_2}$, $\tilde{y}$, and $\tilde{w}$ such that

$$P_1\{{}^{\tilde{W}}/\tilde{x}\} = n!\langle V_1\{{}^{\tilde{W}_1}/\tilde{y}\}\rangle.P_2\{{}^{\tilde{W}_2}/\tilde{w}\}$$

We have the following transition:

$$\langle \mathtt{Snd} \rangle\ \frac{}{P_1\{{}^{\tilde{W}}/\tilde{x}\} \xrightarrow{n!\langle V_1\{{}^{\tilde{W}_1}/\tilde{y}\}\rangle} P_2\{{}^{\tilde{W}_2}/\tilde{w}\}}$$

Let $\sigma_1 \in \mathsf{index}(\tilde{u})$ where $\tilde{u} = \mathtt{fn}(P_1\{{}^{\tilde{W}}/\tilde{x}\})$ such that $\{{}^{n_i}/n\} \in \sigma_1$. Also, let $\sigma_2 = \sigma_1 \cdot \mathsf{next}(n_i)$. Further, let $\tilde{v} = \mathtt{rn}(P_1\{{}^{\tilde{W}}/\tilde{x}\})$, $\tilde{c}_r = \cup_{r \in \tilde{v}} c^r$, $\tilde{c}_k = (c_k, \ldots, c_{k+\lbag P_1 \rbag -1})$, and $\tilde{c}_{k+1} = (c_{k+1}, \ldots, c_{k+\lbag P_1 \rbag -1})$. When P_1 is not a trigger, by the definition of $\mathcal{S}$ (Table 4.3), for both sub-cases, we have $Q_1 \in N_1 \cup N_2$ where:

$$\begin{aligned}
N_1 &= \{(\nu\, \tilde{c}_r)\,(\nu\, \tilde{c}_k)\, \mathcal{R}_{\tilde{v}} \mid \overline{c_k}!\langle \tilde{B} \rangle \mid \mathcal{B}^{k}_{\tilde{x}}(P_1\sigma_1) : \widetilde{W} \bowtie \tilde{B}\} \\
N_2 &= \{(\nu\, \tilde{c}_r)\,(\nu\, \tilde{c}_{k+1})\, \mathcal{R}_{\tilde{v}} \mid n_i!\langle V_2 \rangle.\overline{c_{k+1}}!\langle \tilde{B}_2 \rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(P_2\sigma_2) : \\
&\qquad\qquad V_1\sigma\{{}^{\widetilde{W}_1}/\tilde{y}\} \bowtie V_2,\ \widetilde{W}_2 \bowtie \tilde{B}_2\}
\end{aligned}$$

If $Q_1 \in N_1$, then there is some $Q_2 \in N_2$ such that Q_1 reduces to Q_2 through communication on non-essential prefixes. By Lemma 4.4.3 it is then sufficient to consider the situation when $Q_2 \in N_2$. Let $\tilde{v}_1 = \mathtt{rn}(P_2\{{}^{\tilde{W}_2}/\tilde{z}\})$, $\tilde{v}_2 = \mathtt{rn}(V_2\{{}^{\tilde{W}_1}/\tilde{y}\})$, $\tilde{c}_{r_1} = \cup_{r \in \tilde{v}_1} c^r$, and $\tilde{c}_{r_2} = \cup_{r \in \tilde{v}_2} c^r$. By Definition 4.4.16 and the assumption that $P_1\{{}^{\tilde{W}}/\tilde{x}\}$ is well-typed we have $\tilde{c}_r = \tilde{c}_{r_1} \cdot \tilde{c}_{r_2}$ and $\tilde{c}_{r_1} \cap \tilde{c}_{r_2} = \emptyset$. In that case we have the following transition:

$$\langle \mathtt{New} \rangle\ \dfrac{\langle \mathtt{Par}_R \rangle\ \dfrac{\langle \mathtt{Par}_L \rangle\ \dfrac{\langle \mathtt{Snd} \rangle\ \dfrac{}{n_i!\langle V_2 \rangle.\overline{c_{k+1}}!\langle \tilde{B}_2 \rangle \xrightarrow{n_i!\langle V_2 \rangle} \overline{c_{k+1}}!\langle \tilde{B}_2 \rangle}}{n_i!\langle V_2 \rangle.\overline{c_{k+1}}!\langle \tilde{B}_2 \rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(P_2\sigma_2) \xrightarrow{n_i!\langle V_2 \rangle} R_1}}{\mathcal{R}_{\tilde{v}} \mid n_i!\langle V_2 \rangle.\overline{c_{k+1}}!\langle \tilde{B}_2 \rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(P_2\sigma_2) \xrightarrow{n_i!\langle V_2 \rangle} R_1} \qquad \tilde{c}_r \cdot \tilde{c}_{k+1} \cap \mathtt{fn}(n_i!\langle V_2 \rangle) = \emptyset}{(\nu\, \tilde{c}_r)\,(\nu\, \tilde{c}_{k+1})\,(\mathcal{R}_{\tilde{v}} \mid n_i!\langle V_2 \rangle.\overline{c_{k+1}}!\langle \tilde{B}_2 \rangle \mid \mathcal{B}^{k+1}_{\tilde{z}}(P_2\sigma_2)) \xrightarrow{(\nu\, \tilde{c}_{r_2})\, n_i!\langle V_2 \rangle} Q_2'} \tag{A.102}$$

where $Q_2' = (\nu\, \tilde{c}_{r_1})\,(\nu\, \tilde{c}_{k+1})\, \mathcal{R}_{\tilde{v}} \mid \overline{c_{k+1}}!\langle \tilde{B}_2 \rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(P_2\sigma_2)$ with $V_1\{{}^{\tilde{W}_1}/\tilde{y}\}\sigma_2 \bowtie V_2$ and $\widetilde{W}_2\sigma_1 \bowtie \tilde{B}_2$. Then, we shall show the following:

$$(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{{}^{\tilde{W}}/\tilde{x}\}\ \mathcal{S}\ (\nu\, \tilde{c}_{r_2})\,(Q_2' \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2) \tag{A.103}$$

By assumption that $P_1\{{}^{\tilde{W}}/\tilde{x}\}$ is well-typed, we know $\tilde{v} = \tilde{v}_1 \cdot \tilde{v}_2$ and $\tilde{v}_1 \cap \tilde{v}_2 = \emptyset$. Thus, by Definition 4.4.16 we know $\mathcal{R}_{\tilde{v}} = \mathcal{R}_{\tilde{v}_1} \mid \mathcal{R}_{\tilde{v}_2}$, that is

$$\begin{aligned}
&(\nu\, \tilde{c}_{r_2})\,(Q_2' \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2) \\
&\quad \equiv (\nu\, \tilde{c}_{r_1})\,(\nu\, \tilde{c}_{r_2})\,(\nu\, \tilde{c}_{k+1})(\mathcal{R}_{\tilde{v}_1} \mid \overline{c_{k+1}}!\langle \tilde{B} \rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(P_2\sigma_1) \mid \mathcal{R}_{\tilde{v}_2} \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2) \\
&\quad \equiv (\nu\, \tilde{c}_r)\,(\nu\, \tilde{c}_{k+1})(\mathcal{R}_{\tilde{v}_1} \mid \overline{c_{k+1}}!\langle \tilde{B} \rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(P_2\sigma_1) \mid \mathcal{R}_{\tilde{v}_2} \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2) \\
&\quad = (\nu\, \tilde{c}_r)\,(\nu\, \tilde{c}_{k+1})\, R
\end{aligned}$$

From the definition of $\mathcal{S}$ (Definition 4.4.17) we have that $n \notin \mathtt{fn}(\widetilde{W}_2)$ and thus $\widetilde{W}_2\sigma_1 = \widetilde{W}_2\sigma_2$. Thus, by the definition of $\widetilde{B}_2$ ($\widetilde{W}_2\sigma_2 \bowtie \widetilde{B}_2$) we can see that

$$\mathcal{R}_{\tilde{v}_1} \mid \overline{c_{k+1}}!\langle \widetilde{B} \rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(P_2\sigma_1) \mid \mathcal{R}_{\tilde{v}_2} \in \mathcal{C}^{\widetilde{W}_2\sigma_2}_{\tilde{w}}(P_2\sigma_2) \tag{A.104}$$

Now, we can see that assertion holds as by definition $\sigma_2 = \sigma_1 \cdot \mathsf{next}(n_i)$. Let $\sigma_2' = \sigma_2 \cdot \{{}^{t_1}/t\}$. Then, we have

$$\mathcal{C}^{\tilde{W}_1\sigma_2'}_{\tilde{y}}(t_1 \hookleftarrow_{\mathtt{H}} V_1\sigma_2) = \{P' : (t_1 \hookleftarrow_{\mathtt{H}} V_1)\{{}^{\tilde{W}_1}/\tilde{y}\}\sigma_2 \diamond P'\}$$

As $(t_1 \hookleftarrow_{\mathtt{H}} V_1)\{{}^{\tilde{W}_1}/\tilde{y}\}\sigma_2 = t_1 \hookleftarrow_{\mathtt{H}} V_1\{{}^{\tilde{W}_1}/\tilde{y}\}\sigma_2$ and $V_1\{{}^{\tilde{W}_1}/\tilde{y}\}\sigma_2 \bowtie V_2$ we have

$$(t_1 \hookleftarrow_{\mathtt{H}} V_1)\{{}^{\tilde{W}_1}/\tilde{y}\}\sigma_2 \diamond (t_1 \hookleftarrow_{\mathtt{H}} V_2) \tag{A.105}$$

that is

$$\mathcal{R}_{\tilde{v}_2} \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2 \in \mathcal{C}^{\tilde{W}_1\sigma_2'}_{\tilde{y}}(t_1 \hookleftarrow_{\mathtt{H}} V_1\sigma_2)$$

Finally, by Table 4.3 we have

$$\mathcal{C}^{\tilde{W}\sigma_2'}_{\tilde{x}}(P_2\sigma_2 \parallel t_1 \hookleftarrow_{\mathtt{H}} V_1\sigma_2) = \{R_1 \parallel R_2 : R_1 \in \mathcal{C}^{\tilde{W}_2\sigma_2'}_{\tilde{w}}(P_2\sigma_2),\ R_2 \in \mathcal{C}^{\tilde{W}_1\sigma_2'}_{\tilde{y}}(t_1 \hookleftarrow_{\mathtt{H}} V_1\sigma_2)\}$$

So, by this, (A.104), and (A.105) we have:

$$R \in \mathcal{C}^{\tilde{W}\sigma_2'}_{\tilde{x}}(P_2\sigma_2' \parallel t_1 \hookleftarrow_{\mathtt{H}} V_1\sigma_2)$$

Further, by the assumption $\overline{n} \notin \tilde{u}$ we know $\{{}^{n_i}/\overline{n}\} \notin \sigma_1$. Thus, by $\sigma_2' = \sigma_1 \cdot \mathsf{next}(n_i) \cdot \{{}^{t_j}/t\}$ and Definition 4.4.10, we have

$$\sigma_2' \in \mathsf{index}(\mathtt{fn}((P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{{}^{\tilde{W}}/\tilde{x}\}))$$

Hence, the goal (A.103) follows. This concludes sub-case $\neg\mathsf{tr}(n)$.

ii) Sub-case $\mathsf{tr}(n)$. Let $\widetilde{W}_1$, $\widetilde{W}_2$, $\tilde{y}$, and $\tilde{w}$ be such that

$$P_1\{{}^{\tilde{W}}/\tilde{x}\} = n!\langle V_1\{{}^{\tilde{W}_1}/\tilde{y}\}\rangle.P_2\{{}^{\tilde{W}_2}/\tilde{w}\}$$

The transition inference tree is as follows:

$$\langle\mathtt{Snd}\rangle\ \dfrac{}{P_1\{{}^{\tilde{W}}/\tilde{x}\} \xrightarrow{n!\langle V_1\{{}^{\tilde{W}_1}/\tilde{y}\}\rangle} P_2\{{}^{\tilde{W}_2}/\tilde{w}\}}$$

Let $\sigma_1 \in \mathsf{index}(\tilde{u})$ where $\tilde{u} = \mathtt{fn}(P_1\{{}^{\tilde{W}}/\tilde{x}\})$. Also, let $\tilde{v} = \mathtt{rn}(P_1)$, $\tilde{c}_r = \cup_{r\in\tilde{v}} c^r$, $\tilde{c}_k = (c_k, \ldots, c_{k+\wr P_1 \wr -1})$, $\tilde{c}_{k+1} = (c_{k+1}, \ldots, c_{k+\wr P_1 \wr -1})$, and let S be such that $n : S$. Then, by the definition of $\mathcal{S}$ (Definition 4.4.17) we have $Q_1 \in N_1 \cup N_2 \cup N_3 \cup N_4$ where

$$N_1 = \{(\nu\,\tilde{c}_r)\,(\nu\,\tilde{c}_k)\,\mathcal{R}_{\tilde{v}} \mid \overline{c_k}!\langle \widetilde{B} \rangle \mid \mathcal{B}^{k}_{\tilde{x}}(P_1\sigma_1) : \widetilde{W}\sigma_1 \bowtie \widetilde{B}\}$$
$$N_2 = \{(\nu\,\tilde{c}_r)\,(\nu\,\tilde{c}_{k+1})\,\mathcal{R}_{\tilde{v}} \mid c^n!\langle M^{\tilde{B}_2}_{V_2} \rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(P_2\sigma_1) : \widetilde{W}_2\sigma_1 \bowtie \widetilde{B}_2\}$$
$$N_3 = \{(\nu\,\tilde{c}_r)\,(\nu\,\tilde{c}_{k+1})\,\mathcal{R}_{\tilde{v}} \mid M^{\tilde{B}_2}_{V_2}\,\tilde{n} \mid \mathcal{B}^{k+1}_{\tilde{w}}(P_2\sigma_1) : V_1\{{}^{\tilde{W}_1}/\tilde{y}\}\sigma_1 \bowtie V_2,\ \widetilde{W}_2\sigma_1 \bowtie \widetilde{B}_2\}$$
$$N_4 = \{(\nu\,\tilde{c}_r)\,(\nu\,\tilde{c}_{k+1})\,\mathcal{R}_{\tilde{v}\backslash n} \mid n_{\llbracket S \rangle}!\langle V_2 \rangle.(\overline{c_{k+1}}!\langle \widetilde{B}_2 \rangle \mid c^n?(x).x\,\tilde{z} \mid \mathcal{B}^{k+1}_{\tilde{w}}(P_2\sigma_1)$$
$$: V_1\{{}^{\tilde{W}_1}/\tilde{y}\}\sigma_1 \bowtie V_2,\ \widetilde{W}_2\sigma_1 \bowtie \widetilde{B}_2\}$$

with

$$M_{V_2}^{\tilde{B}_2} = \lambda \widetilde{z}.\, z_{[S\rangle}!\langle V_2\rangle.(\overline{c_{k+1}}!\langle \widetilde{B}_2\rangle \mid c^n?(x).x\, \widetilde{z})$$

If $Q_1 \in N_1 \cup N_2 \cup N_3$, then Q_1 reduces to some $Q_4 \in N_4$ through communication on non-essential prefixes. By Lemma 4.4.3 it suffices to consider the case when $Q_4 \in N_4$. Let $\widetilde{v}_1 = \mathtt{rn}(P_2)$, $\widetilde{v}_2 = \mathtt{rn}(V_2)$, $\widetilde{c}_{r_1} = \cup_{r\in\widetilde{v}_1} c^r$, and $\widetilde{c}_{r_2} = \cup_{r\in\widetilde{v}_2} c^r$. By Definition 4.4.16 and the assumption that $P_1\{\tilde{W}/\tilde{x}\}$ is well-typed we have $\widetilde{c}_r = \widetilde{c}_{r_1} \cdot \widetilde{c}_{r_2}$ and $\widetilde{c}_{r_1} \cap \widetilde{c}_{r_2} = \emptyset$. We then infer the following transition:

$$Q_4 \xrightarrow{(\nu\, \widetilde{c}_{r_2})\, n_{[S\rangle}!\langle V_2\rangle} Q_4'$$

where $Q_4' = (\nu\, \widetilde{c}_{r_1})\, (\nu\, \widetilde{c}_{k+1})\, \mathcal{R}_{\widetilde{v}\setminus n} \mid \overline{c_{k+1}}!\langle \widetilde{B}_2\rangle \mid c^n?(b).(b\, \widetilde{n}) \mid \mathcal{B}^{k+1}_{\widetilde{w}}(P_2\sigma_1)$. Then, we shall show the following

$$(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{\tilde{W}/\tilde{x}\}\ \mathcal{S}\ (\nu\, \widetilde{c}_{r_2})\, (Q_4' \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2) \tag{A.106}$$

By assumption that $P_1\{\tilde{W}/\tilde{x}\}$ is well-typed, we know $\widetilde{v} = \widetilde{v}_1 \cdot \widetilde{v}_2$ and $\widetilde{v}_1 \cap \widetilde{v}_2 = \emptyset$. Hence, by Definition 4.4.16 we know $\mathcal{R}_{\widetilde{v}} = \mathcal{R}_{\widetilde{v}_1} \mid \mathcal{R}_{\widetilde{v}_2}$, that is

$$\begin{aligned}
Q_4' &\equiv (\nu\, \widetilde{c}_{r_1})\, (\nu\, \widetilde{c}_{k+1})\, \mathcal{R}_{\widetilde{v}\setminus n} \mid c^n?(x).x\, \widetilde{n} \mid \overline{c_{k+1}}!\langle \widetilde{B}_2\rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(P_2\sigma_1)\\
&= (\nu\, \widetilde{c}_{r_1})\, (\nu\, \widetilde{c}_{k+1})\, \mathcal{R}_{\widetilde{v}} \mid \overline{c_{k+1}}!\langle \widetilde{B}_2\rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(P_2\sigma_1)\\
&= (\nu\, \widetilde{c}_{r_1})\, (\nu\, \widetilde{c}_{k+1})\, \mathcal{R}_{\widetilde{v}_1} \mid \mathcal{R}_{\widetilde{v}_2} \mid \overline{c_{k+1}}!\langle \widetilde{B}_2\rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(P_2\sigma_1)
\end{aligned}$$

That is, we have

$$\begin{aligned}
&(\nu\, \widetilde{c}_{r_2})\, (Q_4' \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2)\\
&\quad\equiv (\nu\, \widetilde{c}_{r_2})\, ((\nu\, \widetilde{c}_{r_1})\, (\nu\, \widetilde{c}_{k+1})\, \mathcal{R}_{\widetilde{v}_1} \mid \mathcal{R}_{\widetilde{v}_2} \mid \overline{c_{k+1}}!\langle \widetilde{B}_2\rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(P_2\sigma_1) \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2)\\
&\quad\equiv (\nu\, \widetilde{c}_{r_1})\, (\nu\, \widetilde{c}_{r_2})\, (\nu\, \widetilde{c}_{k+1})(\mathcal{R}_{\widetilde{v}_1} \mid \overline{c_{k+1}}!\langle \widetilde{B}\rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(P_2\sigma_1) \mid \mathcal{R}_{\widetilde{v}_2} \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2)\\
&\quad\equiv (\nu\, \widetilde{c}_{r})\, (\nu\, \widetilde{c}_{k+1})(\mathcal{R}_{\widetilde{v}_1} \mid \overline{c_{k+1}}!\langle \widetilde{B}\rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(P_2\sigma_1) \mid \mathcal{R}_{\widetilde{v}_2} \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2) = (\nu\, \widetilde{c}_r)\, (\nu\, \widetilde{c}_{k+1})\, R
\end{aligned}$$

Now, by Definition 4.4.8 we may notice that $\widetilde{v}_2 = \mathtt{rn}(t \hookleftarrow_{\mathtt{H}} V_2)$. Let $\sigma_1' = \sigma_1 \cdot \{t_1/t\}$. So, by Table 4.3 we have

$$\mathcal{R}_{\widetilde{v}_1} \mid \overline{c_{k+1}}!\langle \widetilde{B}\rangle \mid \mathcal{B}^{k+1}_{\widetilde{w}}(P_2\sigma_1) \in \mathcal{C}^{\tilde{W}_2\sigma_1}_{\widetilde{w}}(P_2\sigma_1)$$

and

$$\mathcal{R}_{\widetilde{v}_2} \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2 \in \mathcal{C}^{\tilde{W}_1\sigma_1}_{\widetilde{y}}(t_1 \hookleftarrow_{\mathtt{H}} V_1\sigma_1') \tag{A.107}$$

Thus, by the definition of the parallel composition case of $\mathcal{C}^{-}_{-}(-)$ (Table 4.3) we have

$$R \in \mathcal{C}^{\tilde{W}\sigma_1'}_{\widetilde{x}}((P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\sigma_1')$$

Now, we can notice that $\widetilde{c}_r = \mathsf{cr}(R)$ and $\widetilde{c}_{k+1} = \mathsf{fpn}(R)$. Further, we have

$$\sigma_1' \in \mathsf{index}(\mathtt{fn}((P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{\tilde{W}/\tilde{x}\}))$$

Thus, (A.106) follows. This concludes sub-case $\mathsf{tr}(n)$ of case $\langle\mathtt{Snd}\rangle$.

Finally, we briefly analyze the case when P_1 is a trigger collection. Let σ_1 be defined as above. Let H_1' be such that $P_1\{\tilde{W}/\tilde{x}\}\sigma_1 \diamond H_1'$. Then, by Table 4.3, Q_1 is as follows:

$$Q_1 = \mathcal{R}_{\tilde{v}} \parallel H_1'$$

where $\widetilde{v} = \mathtt{rn}(P_1\{\tilde{W}/\tilde{x}\})$. Further, by Definition 4.4.15 we know $H_1' = n_i!\langle V_2\rangle.H_2'$ such that $V_1\{\tilde{W_1}/\tilde{y}\}\sigma_2 \bowtie V_2$ and $P_2\{\tilde{W_2}/\tilde{w}\}\sigma_2 \diamond H_2'$ with $\sigma_2 = \sigma_1 \cdot \mathsf{next}(n_i)$. We can see that

$$H_1' \xrightarrow{n_i!\langle V_2\rangle} H_2'$$

So, we should show

$$(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{\tilde{W}/\tilde{x}\} \;\; \mathcal{S} \;\; (\mathcal{R}_{\tilde{v}} \parallel H_2' \mid t_1 \hookleftarrow_{\mathtt{H}} V_2) \tag{A.108}$$

Similarly to previous sub-cases, we have

$$\mathcal{R}_{\tilde{v}} \parallel H_2' \mid t_1 \hookleftarrow_{\mathtt{H}} V_2 \equiv \mathcal{R}_{\tilde{v}_1} \parallel H_2' \parallel \mathcal{R}_{\tilde{v}_2} \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2$$

By $P_2\{\tilde{W_2}/\tilde{w}\}\sigma_2 \diamond H_2'$ and $\widetilde{v}_1 = \mathtt{rn}(P_2\{\tilde{W_2}/\tilde{w}\})$ we have

$$\mathcal{R}_{\tilde{v}_1} \parallel H_2' \in \mathcal{C}_{\tilde{w}}^{\tilde{W}_2\sigma_2}(P_2\sigma_2) \tag{A.109}$$

Let $\sigma_2' = \sigma_2 \cdot \{t_1/t\}$. By (A.109), (A.107), and definition of $\mathcal{C}_{-}^{-}(\,-\,)$ (Table 4.3) for the parallel composition case we have

$$\mathcal{R}_{\tilde{v}_1} \parallel H_2' \parallel \mathcal{R}_{\tilde{v}_2} \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2 \in \mathcal{C}_{\tilde{x}}^{\tilde{W}\sigma_2'}(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)$$

Thus, we reach goal (A.108). This concludes case $\langle\mathtt{Snd}\rangle$.

2. Case $\langle\mathtt{Rv}\rangle$. In this case we know $P_1 = n?(y).P_2$. We first consider cases when P_1 is not a trigger collection, and then briefly discuss the case when it is a trigger collection. As in the previous case, we distinguish two sub-cases: (i) $\neg\mathsf{tr}(n)$ and (ii) $\mathsf{tr}(n)$:

   i) Sub-case $\neg\mathsf{tr}(n)$. We have the following transition:

$$\langle\mathtt{Rv}\rangle\;\frac{}{(n_i?(y).P_2)\{\tilde{W}/\tilde{x}\} \xrightarrow{n?(V_1)} P_2\{\tilde{W}/\tilde{x}\}\{V_1/y\}}$$

   Here we assume $y \in \mathtt{fv}(P_2)$. Let $\sigma_1 \in \mathsf{index}(\widetilde{u})$ with $\widetilde{u} = \mathtt{fn}(P_1\{\tilde{W}/\tilde{x}\})$ such that $\{n_i/n\} \in \sigma_1$. Also, let $\sigma_2 = \sigma_1 \cdot \mathsf{next}(n_i)$. Further, let $\widetilde{v} = \mathtt{rn}(P_1\{\tilde{W}/\tilde{x}\})$, $\widetilde{c}_r = \cup_{r\in\tilde{v}} c^r$, $\widetilde{c}_k = (c_k, \ldots, c_{k+\wr P_1 \wr -1})$, and $\widetilde{c}_{k+1} = (c_{k+1}, \ldots, c_{k+\wr P_1 \wr -1})$. By the definition of $\mathcal{S}$ (Table 4.3) we have $Q_1 \in N_1 \cup N_2$ where

$$N_1 = \{(\nu\,\widetilde{c}_r)\,(\nu\,\widetilde{c}_k)\,\mathcal{R}_{\tilde{v}} \mid \overline{c_k}!\langle\widetilde{B}\rangle \mid \mathcal{B}_{\tilde{x}}^{k}(P_1\sigma_1) : \widetilde{W}\sigma_1 \bowtie \widetilde{B}\}$$
$$N_2 = \{(\nu\,\widetilde{c}_r)\,(\nu\,\widetilde{c}_{k+1})\,\mathcal{R}_{\tilde{v}} \mid n_i?(y).\overline{c_{k+1}}!\langle\widetilde{B}y\rangle \mid \mathcal{B}_{\tilde{x}y}^{k+1}(P_2\sigma_2) : \widetilde{W}\sigma_1 \bowtie \widetilde{B}\}$$

   Similar to the other cases, if $Q_1 \in N_1$, then Q_1 reduces to some $Q_1' \in N_2$ through communication on non-essential prefixes.

   Now, we pick V_2 such that $V_1\sigma_v \cdot \sigma_2 \bowtie V_2$ where $\sigma_v \in \mathsf{index}(\mathtt{fn}(V) \setminus \widetilde{u})$ such that

$$\sigma_v \cdot \sigma_2 \in \mathsf{index}(\mathtt{fn}(P_2\{\tilde{W}V_1/\tilde{x}y\})) \tag{A.110}$$

By Lemma 4.4.3 it suffices to consider the case when $Q_1' \in N_2$, under which we have the following transition:

$$\langle\texttt{Rv}\rangle\ \frac{}{n_i?(y).\overline{c_{k+1}}!\langle\widetilde{B}y\rangle \xrightarrow{n_i?(V_2)} \overline{c_{k+1}}!\langle\widetilde{B}V_2\rangle} \tag{A.111}$$

$$\langle\texttt{Par}_L\rangle\ \frac{(\text{A.111}) \qquad \texttt{bn}(n_i?(V_2)) \cap \texttt{fn}(\mathcal{B}^{k+1}_{\tilde{x}y}(P_2\sigma)) = \emptyset}{n_i?(y).\overline{c_{k+1}}!\langle\widetilde{B}y\rangle \mid \mathcal{B}^{k+1}_{\tilde{x}y}(P_2\sigma_2) \xrightarrow{n_i?(V_2)} \overline{c_{k+1}}!\langle\widetilde{B}V_2\rangle \mid \mathcal{B}^{k+1}_{\tilde{x}y}(P_2\sigma_2)} \tag{A.112}$$

$$\langle\texttt{Res}\rangle\ \frac{(\text{A.112}) \qquad \widetilde{c}_r \cdot \widetilde{c}_{k+1} \cap \texttt{fn}(n_i?(V_2)) = \emptyset}{Q_1' \xrightarrow{n_i?(V_2)} R}$$

where $R = (\nu\,\widetilde{c}_r)\,(\nu\,\widetilde{c}_{k+1})\,\overline{c_{k+1}}!\langle\widetilde{B}V_2\rangle \mid \mathcal{B}^{k+1}_{\tilde{x}y}(P_2\sigma_2)$ with $\widetilde{W}\sigma_1 \bowtie \widetilde{B}$.
We should show that

$$P_2\{{}^{\widetilde{W}V_1}/_{\tilde{x}y}\}\ \mathcal{S}\ R \tag{A.113}$$

Now, we discuss the assertion on index substitutions. We pick the following substitution $\sigma_2' = \sigma_2 \cdot \sigma_v$. First, we may notice that $\sigma_2 \in \mathsf{index}(\widetilde{u})$ as by the assumption $\overline{n} \notin \widetilde{u}$ we know $\{{}^{\overline{n}_i}/_{\overline{n}}\} \notin \sigma_1$. Thus, we have $\sigma_2' \in \mathsf{index}(P_2\{{}^{\widetilde{W}V_1}/_{\tilde{x}y}\})$. Next, as σ_v and σ_1 are disjunctive, we have $(\sigma_2' \setminus \mathsf{next}(n_i)) \cdot \sigma_1 = \sigma_1 \cdot (\sigma_2' \setminus \mathsf{next}(n_i))$. We know $n \notin \texttt{fn}(\widetilde{W})$ and $n \notin \texttt{fn}(V_1)$. Thus, $\widetilde{W}V_1\sigma_v \cdot \sigma_1 = \widetilde{W}V_1\sigma_v \cdot \sigma_2$. That is, we may notice that $\widetilde{W}V_1\sigma_2 \cdot \sigma_v \bowtie \widetilde{B}V_2$. Further, by the definition of σ_v, we have $P_2\sigma_2 = P_2\sigma_2 \cdot \sigma_v$. Thus, we have

$$R \in \mathcal{C}^{\widetilde{W}V_1\sigma_2'}_{\tilde{x}y}(P_2\sigma_2')$$

Finally, by this and (A.110) the goal (A.113) follows. This concludes sub-case $\neg\mathsf{tr}(n)$.

ii) Sub-case $\mathsf{tr}(n)$. The transition inference tree is as follows:

$$\langle\texttt{Rv}\rangle\ \frac{}{(n?(y).P_2)\{{}^{\widetilde{W}}/_{\tilde{x}}\} \xrightarrow{n?(V_1)} P_2\{{}^{\widetilde{W}}/_{\tilde{x}}\}\{{}^{V_1}/_{y}\}}$$

Let $\sigma_1 = \mathsf{index}(\widetilde{u})$ where $\widetilde{u} = \texttt{fn}(P_1\{{}^{\widetilde{W}}/_{\tilde{x}}\})$. Also, let $\widetilde{v} = \texttt{rn}(P_1)$, $\widetilde{c}_r = \cup_{r\in\widetilde{v}} c^r$, $\widetilde{c}_k = (c_k, \ldots, c_{k+\lceil P_1\rfloor-1})$, $\widetilde{c}_{k+1} = (c_{k+1}, \ldots, c_{k+\lceil P_1\rfloor-1})$, and let S be such that $n : S$. Then, by the definition of $\mathcal{S}$ (Definition 4.4.17) we have $Q_1 \in N_1 \cup N_2 \cup N_3 \cup N_4$ where

$$N_1 = \{(\nu\,\widetilde{c}_r)\,(\nu\,\widetilde{c}_k)\,\mathcal{R}_{\widetilde{v}} \mid \overline{c_k}!\langle\widetilde{B}\rangle \mid \mathcal{B}^{k}_{\tilde{x}}(P_1\sigma_1) : \widetilde{W}\sigma_1 \bowtie \widetilde{B}\}$$
$$N_2 = \{(\nu\,\widetilde{c}_r)\,(\nu\,\widetilde{c}_k)\,\mathcal{R}_{\widetilde{v}} \mid c^n!\langle M^{\widetilde{B}}_V\rangle \mid \mathcal{B}^{k+1}_{\tilde{x}y}(P_2\sigma_1) : \widetilde{W}\sigma_1 \bowtie \widetilde{B}\}$$
$$N_3 = \{(\nu\,\widetilde{c}_r)\,(\nu\,\widetilde{c}_k)\,\mathcal{R}_{\widetilde{v}\setminus n} \mid M^{\widetilde{B}}_V\,\widetilde{n} \mid \mathcal{B}^{k+1}_{\tilde{x}y}(P_2\sigma_1) : \widetilde{W}\sigma_1 \bowtie \widetilde{B}\}$$
$$N_4 = \{(\nu\,\widetilde{c}_r)\,(\nu\,\widetilde{c}_{k+1})\,\mathcal{R}_{\widetilde{v}\setminus n} \mid n_{[S\rangle}?(y).(\overline{c_{k+1}}!\langle\widetilde{B}y\rangle \mid c^n?(x).x\,\widetilde{n} \mid \mathcal{B}^{k+1}_{\tilde{x}y}(P_2\sigma_1) : \widetilde{W}\sigma_1 \bowtie \widetilde{B}\}$$

with

$$M^{\widetilde{B}}_V = \lambda\widetilde{z}.\,z_{[S\rangle}?(y).(\overline{c_{k+1}}!\langle\widetilde{B}y\rangle \mid c^n?(x).x\,\widetilde{z})$$

Similar to the other cases, if $Q_1 \in N_1 \cup N_2 \cup N_3$, then there exists some $Q_1' \in N_4$ such that Q_1 reduces to Q_1' through communication on non-essential prefixes. By Lemma 4.4.3 it suffice to consider the case when $Q_1 \in N_4$. We infer the following transition:

$$Q_1 \xrightarrow{n_{[S\rangle}?(V_2)} (\nu\, \widetilde{c}_r)\, (\nu\, \widetilde{c}_{k+1})\, R$$

where $R = \mathcal{R}_{\widetilde{v}\setminus n} \mid \overline{c_{k+1}}!\langle \widetilde{B} V_2\rangle \mid c^n?(b).(b\,\widetilde{n}) \mid \mathcal{B}^{k+1}_{\tilde{x}y}(P_2\sigma_1)$ and $V_1\sigma \bowtie V_2$ for some σ. We should show that

$$P_2\{\tilde{W} V_1/\tilde{x}y\}\ \mathcal{S}\ (\nu\, \widetilde{c}_r)\, (\nu\, \widetilde{c}_{k+1})\, R \tag{A.114}$$

We may notice that we have $\widetilde{v} = \mathtt{rn}(P_1) = \mathtt{rn}(P_2)$ and as $\mathtt{tr}(n)$ we have $n \in \widetilde{v}$. Thus, we have the following structural equivalence

$$R \equiv \mathcal{R}_{\widetilde{v}} \mid \overline{c_{k+1}}!\langle \widetilde{B} V_2\rangle \mid \mathcal{B}^{k+1}_{\tilde{x}y}(P_2\sigma_1)$$

Further, we have $V_1\sigma \bowtie V_2$, $\widetilde{W}\sigma_1 \bowtie \widetilde{B}$. Thus by the definition of $\mathcal{C}^{-}_{-}(-)$ (Table 4.3) the goal (A.114) follows. This concludes sub-case $\mathtt{tr}(n)$ of case $\langle\mathtt{Rv}\rangle$.

Now, we briefly consider the case when P_1 is a trigger collection. Let σ_1 be defined as above. Let H_1' be such that $P_1\{\tilde{W}/\tilde{x}\}\sigma_1 \diamond H_1'$. Then, by definition of $\mathcal{S}$, we know Q_1 has the following shape:

$$Q_1 = \mathcal{R}_{\widetilde{v}} \parallel H_1$$

where $\widetilde{v} = \mathtt{rn}(P_1\{\tilde{W}/\tilde{x}\})$. Further, by Definition 4.4.15 we know $H_1' = n_i?(y).H_2'$ such that $P_2\{\tilde{W}/\tilde{x}\}\sigma_2 \diamond H_2'$, where $\sigma_2 = \sigma_1 \cdot \mathsf{next}(n_i)$. Now, let V_2 be such that $V_1\sigma \bowtie V_2$, for some σ. We could see that

$$H_1' \xrightarrow{n_i?(V_2)} H_2'\{V_2/y\}$$

We should show that

$$P_2\{\tilde{W}/\tilde{x}\}\{V_1/y\}\ \mathcal{S}\ \mathcal{R}_{\widetilde{v}} \parallel H_2\{V_2/y\} \tag{A.115}$$

By $P_2\{\tilde{W}/\tilde{x}\}\sigma_2 \diamond H_2'$ and noticing that $\diamond$ is closed under the substitution of $\bowtie$-related values we have

$$P_2\{\tilde{W}/\tilde{x}\}\{V_1/y\}\sigma_2 \diamond H_2'\{V_2/y\}$$

Thus, goal (A.115) follows. This concludes case $\langle\mathtt{Rv}\rangle$.

3. Case $\langle\mathtt{App}\rangle$. Here we know $P_1 = V_1\,(\widetilde{r}, u)$ where $\widetilde{r} = (r_1, \ldots, r_n)$. We distinguish two sub-cases: (i) $V_1 = x$ where $\{V_x/x\} \in \{\tilde{W}/\tilde{x}\}$ and (ii) V_1 is an abstraction. Let $V_1\{\tilde{W}/\tilde{x}\} = \lambda(\widetilde{y}, z).\, P_2$ where $\widetilde{y} = \widetilde{y}^1, \ldots, \widetilde{y}^n$. The inference tree is as follows

$$\langle\mathtt{App}\rangle\ \frac{}{(V_1\,(\widetilde{r}, u))\{\tilde{W}/\tilde{x}\} \xrightarrow{\tau} P_2\{\tilde{r}, u/\tilde{y}, z\}}$$

Let $\sigma_1 = \mathsf{index}(\mathtt{fn}(P_1\rho_1))$ such that $\{u_i/u\} \in \sigma_1$. Further, let $\widetilde{v} = \mathtt{rn}(V_1)$, $\widetilde{c}_{vr} = \bigcup_{r\in\tilde{v},\tilde{r}} c^r$, $\widetilde{c}_k = (c_k, \ldots, c_{k+\wr P_1 \int -1})$, and $\widetilde{c}_{k+1} = (c_{k+1}, \ldots, c_{k+\wr P_1 \int -1})$. Also, let $\widetilde{m} = (u_i, \ldots, u_{i+|\mathcal{G}(C)|-1})$ with $u_i : C$ and $\widetilde{r}_i = (r^i_1, \ldots, r^i_{|\mathcal{R}^\star(S_i)|}))$ with $r_i : S_i$ for $i = \{1, \ldots, n\}$. Then, by the definition of $\mathcal{S}$ we have $Q_1 \in N$ where N is defined as follows:

$$N = N_1 \cup N_4 \cup \big(\cup_{1\le l\le n} N_2^l\big) \cup \big(\cup_{1\le l\le n-1} N_3^l\big)$$

where

$$
\begin{aligned}
N_1 &= \{(\nu\,\widetilde{c}_k)\,(\nu\,\widetilde{c}_{vr})\,\mathcal{R}_{\tilde{v},\tilde{r}} \mid \overline{c_k}!\langle\widetilde{B}\rangle \mid \mathcal{B}^{k}_{\tilde{x}}(P_1\sigma_1) : \widetilde{W}\sigma_1 \bowtie \widetilde{B}\}\\
N_2^l &= \{(\nu\,\widetilde{c}_{k+1})\,(\nu\,\widetilde{c}_{vr})\,\mathcal{R}_{\tilde{v},r_l,\dots,r_n} \mid \overbrace{c^{r_l}!\langle\lambda\widetilde{z}_l.c^{r_{l+1}}!\langle\lambda\widetilde{z}_{l+1}.\cdots\,.c^{r_n}!\langle\lambda\widetilde{z}_n.}^{|\tilde{r}|-l+1}\,Q_l^{V_2}\rangle\,\rangle\rangle\\
&\qquad : V_1\{\tilde{W}/\tilde{x}\}\sigma_1 \bowtie V_2\}\\
N_3^l &= \{(\nu\,\widetilde{c}_{k+1})\,(\nu\,\widetilde{c}_{vr})\,\mathcal{R}_{\tilde{v},r_{l+1},\dots,r_n} \mid \lambda\widetilde{z}_l.\,\overbrace{c^{r_{l+1}}!\langle\lambda\widetilde{z}_{l+1}.\cdots\,.c^{r_n}!\langle\lambda\widetilde{z}_n.}^{|\tilde{r}|-l}\,Q_l^{V_2}\rangle\,\rangle\;\widetilde{r}_l\\
&\qquad : V_1\{\tilde{W}/\tilde{x}\}\sigma_1 \bowtie V_2\}\\
N_4 &= \{(\nu\,\widetilde{c}_{k+1})\,(\nu\,\widetilde{c}_{vr})\,\mathcal{R}_{\tilde{v}} \mid V_2\,(\widetilde{r}_1,\dots,\widetilde{r}_n,\widetilde{m}) : V_1\{\tilde{W}/\tilde{x}\}\sigma_1 \bowtie V_2\}
\end{aligned}
$$

where

$$Q_l^{V_2} = V_2\,(\widetilde{r}_1,\dots,\widetilde{r}_{l-1},\widetilde{z}_l,\dots,\widetilde{z}_n,\widetilde{m})$$

Note that for any

$$Q_1 \in N_1 \cup N_4 \cup \big(\cup_{1\le l\le n} N_2^l\big) \cup \big(\cup_{1\le l\le n-1} N_3^l\big)$$

there exist $Q_1' \in N_5$ such that Q_1 reduces to Q_1' through communication on non-essential prefixes. By Lemma 4.4.3 it then suffices to consider the case $Q_1' \in N_5$.

Let $V_2 = \lambda\widetilde{y}^1,\dots,\widetilde{y}^n,\widetilde{z}.\,Q_2$ where $\widetilde{z} = (z_1,\dots,z_{|\mathcal{G}(C)|})$. Then, we have the following transition:

$$Q_1' \xrightarrow{\tau} (\nu\,\widetilde{c}_{k+1})\,(\nu\,\widetilde{c}_r)\,R$$

where $R = \mathcal{R}_{\tilde{v}} \mid Q_2\{\tilde{r}_1,\dots,\tilde{r}_n,\widetilde{m}/\widetilde{y}^1,\dots,\widetilde{y}^n,\widetilde{z}\}$. We should show that

$$P_2\{\tilde{r},u/\tilde{y},z\}\;\;\mathcal{S}\;\;(\nu\,\widetilde{c}_{k+1})\,(\nu\,\widetilde{c}_{vr})\,R \tag{A.116}$$

By $V_1\rho_1\sigma_1 \bowtie V_2$ (with $\rho_1 = \{\tilde{W}/\tilde{x}\}$) and Definition 4.4.13 either $V_2 \in \mathcal{C}(V_1\rho_1\sigma_1)$ or $V_1\rho_1 \bowtie_{\mathsf{c}} V_2$. In the former case, we know $V_2 \in \widehat{\mathcal{C}}^{\rho_1'}_{\sigma_1'}(V_1')$ where $\rho_1' = \{\tilde{W}'/\tilde{x}'\}$ is such that $V_1'\rho_1'\sigma_1' = V_1\rho_1\sigma_1$.

Let $\widetilde{B}'$ be such that $\widetilde{W}'\sigma_1' \bowtie \widetilde{B}'$. By Table 4.3 we have

$$V_2 = \mathcal{V}_{\tilde{x}'}(\lambda(\widetilde{y},z).\,P_2')\{\tilde{B}'/\tilde{x}'\} = \lambda(\widetilde{y^1},\dots,\widetilde{y^n},\widetilde{z}).\,Q_2\{\tilde{B}'/\tilde{x}'\}$$

where

$$Q_2 = (\nu\,\widetilde{c}_y)\,\prod_{i\in|\widetilde{y}|}(c^{y_i}?(b).(b\,\widetilde{y}^i)) \mid \overline{c_{k+1}}!\langle\widetilde{B}'\rangle \mid \mathcal{B}^{k+1}_{\tilde{x}'}(P_2'\{z_1/z\})$$

with $\widetilde{c}_y = \bigcup_{i\in|\tilde{y}|} c^{y_i}$ and P_2' is such that

$$P_2'\{\tilde{W}'/\tilde{x}'\} = P_2$$

Thus, we know

$$R \equiv \mathcal{R}_{\tilde{v}} \mid (\nu\,\widetilde{c}_y)\,\prod_{i\in|\widetilde{y}|}(c^{y_i}?(b).(b\,\widetilde{r}_i)) \mid \overline{c_{k+1}}!\langle\widetilde{B}'\rangle \mid \mathcal{B}^{k+1}_{\tilde{x}'}(P_2'\{z_1/z\})$$

Now, we know $(\nu\, \widetilde{c}_{vr})\, R \equiv (\nu\, \widetilde{c}_v)\, R$ where $\widetilde{c}_v = \bigcup_{r\in\tilde{v}} c^r$ since $\widetilde{c}_{vr} \setminus \widetilde{c}_v \not\subseteq \mathtt{fn}(R)$. Further, by renaming bound names we have

$$R \equiv \mathcal{R}_{\tilde{v}} \mid (\nu\, \widetilde{c}_r) \prod_{r\in\widetilde{r}} (c^r?(b).(b\, \widetilde{r})) \mid \overline{c_{k+1}}!\langle \widetilde{B}' \rangle \mid \mathcal{B}^{k+1}_{\tilde{x}'}(P'_2\{^{z_1}/_{z}\})\{^{\tilde{c}_r}/_{\tilde{c}_y}\}$$

where $\widetilde{c}_r = \bigcup_{r\in\tilde{r}} c^r$. Now, by the definition of $\mathcal{R}_{\tilde{v}}$ (Definition 4.4.16) we know

$$\mathcal{R}_{\tilde{v},\tilde{r}} = \mathcal{R}_{\tilde{v}} \mid \prod_{r\in\widetilde{r}} (c^n?(x).x\, \widetilde{r})$$

Thus, we have

$$R \equiv (\nu\, \widetilde{c}_r)\, \mathcal{R}_{\tilde{v},\tilde{r}} \mid \overline{c_{k+1}}!\langle \widetilde{B}' \rangle \mid \mathcal{B}^{k+1}_{\tilde{x}'}(P'_2\{^{z_1}/_{z}\})\{^{\tilde{c}_r}/_{\tilde{c}_y}\} = (\nu\, \widetilde{c}_r)\, R'$$

and by the definition we have $R' \in \mathcal{C}^{\widetilde{W}'}_{\tilde{x}'}(P'_2)$. We may notice that $\widetilde{v}, \widetilde{r} = \mathtt{rn}(P'_2)$ and $(\nu\, c_{vr})\, R \equiv (\nu\, c_v)\, (\nu\, c_r)\, R'$.

The later case, when $V_1\rho_1 \bowtie_{\mathsf{c}} V_2$, follows by the fact that bodies of characteristic and triggers values are $\diamond$-related to their minimal counterparts as shown in Lemma 4.4.2 and that relation $\diamond$ is closed under names substitutions.

So, the goal (A.116) follows. This concludes case $\langle \mathtt{App} \rangle$ (and base cases). Next, we consider inductive cases.

4. Case $\langle \mathtt{Par}_L \rangle$. In this case we distinguish two sub-cases: (i) $P_1\rho = P'_1\rho'_1 \mid P''_1\rho''_1$ and (ii) $P_1\rho = P'_1\rho'_1 \parallel P''_1\rho''_1$ where P'_1 is a triggers collection. The final rule in the inference tree is:

$$\langle \mathtt{Par}_L \rangle\ \frac{P'_1\rho'_1 \xrightarrow{\ell} P'_2\rho'_2 \qquad \mathtt{bn}(\ell) \cap \mathtt{fn}(P''_1) = \emptyset}{P'_1\rho'_1 \mid P''_1\rho''_1 \xrightarrow{\ell} P'_2\rho'_2 \mid P''_1\rho''_1}$$

Let $\sigma_1 \in \mathsf{index}(\widetilde{u})$ where $\widetilde{u} = \mathtt{fn}(P_1\{^{\widetilde{W}}/_{\tilde{x}}\})$. Further, we know $\rho'_1 = \{^{\widetilde{W}_1}/_{\tilde{y}}\}$ and $\rho''_1 = \{^{\widetilde{W}_2}/_{\tilde{z}}\}$. Further, let σ'_1 and σ''_1 such that

$$P_1\rho_1\sigma_1 = P'_1\rho'_1\sigma'_1 \mid P''_1\rho''_1\sigma''_1$$

In sub-case (i), by the definition of $\mathcal{S}$ (Table 4.3) we have $Q_1 \in N_1 \cup N_2 \cup N_3$ where

$$\begin{aligned}
N_1 =& \{(\nu\, \widetilde{c}_k)\, (\nu\, \widetilde{c}_r)\, (\overline{c_k}!\langle \widetilde{B} \rangle \mid \mathcal{B}^{k}_{\tilde{x}}(P'_1\sigma'_1 \mid P''_1\sigma''_1)) : \widetilde{W}\sigma_1 \bowtie \widetilde{B}\} \\
N_2 =& \{(\nu\, \widetilde{c}_{k+1})\, (\nu\, \widetilde{c}_r)\, (\overline{c_{k+1}}!\langle \widetilde{B}_1 \rangle.\overline{c_{k+2}}!\langle \widetilde{B}_2 \rangle \mid \mathcal{B}^{k+1}_{\tilde{y}}(P'_1\sigma'_1) \mid \mathcal{B}^{k+2}_{\tilde{z}}(P''_1\sigma''_1)) \\
& \quad : \widetilde{W_i}\sigma_1 \bowtie \widetilde{B}_i, i \in \{1,2\}\} \\
N_3 =& \{(\nu\, \widetilde{c})\, (\nu\, \widetilde{c}_r)\, R'_1 \mid R''_1 : R'_1 \in \mathcal{C}^{\widetilde{W}_1}_{\tilde{y}}(P'_1\sigma'_1),\ R''_1 \in \mathcal{C}^{\widetilde{W}_2}_{\tilde{z}}(P''_1\sigma''_1)\}
\end{aligned}$$

where $\widetilde{c}_k = (c_k, \ldots, c_{k+\wr P_1 \int -1})$, $\widetilde{c}_{k+1} = (c_{k+1}, \ldots, c_{k+\wr P_1 \int -1})$, and $\widetilde{c} = \mathsf{fpn}(R'_1 \mid R''_1)$. Note that for $Q_1 \in N_1 \cup N_2$ there exists some Q'_1 and Q''_1 such that Q_1 reduces to $Q'_1 \mid Q''_1 \in N_3$ through communication on non-essential prefixes, with

$$Q'_1 \in \widehat{\mathcal{C}}^{\rho'_1}_{\sigma'_1}(P'_1) \tag{A.117}$$

$$Q''_1 \in \widehat{\mathcal{C}}^{\rho''_1}_{\sigma''_1}(P''_1) \tag{A.118}$$

Then, by Lemma 4.4.3 it suffices to consider the case of $Q'_1 \mid Q''_1 \in N_3$. By the definition of $\mathcal{S}$ we have

$$P'_1\rho'_1 \,\mathcal{S}\, Q'_1 \tag{A.119}$$

$$P''_1\rho''_1 \,\mathcal{S}\, Q''_1 \tag{A.120}$$

To apply IH we do the case analysis on the action ℓ:

- Sub-case $\ell \not\equiv (\nu\, \widetilde{m}_1)\, n!\langle V_1\rangle$. By (A.119) and IH we know there is Q'_2 such that $Q'_1 \overset{\ell}{\Rightarrow} Q'_2$ and

  $$P'_2\rho'_2 \;\; \mathcal{S} \;\; Q'_2 \tag{A.121}$$

  We should show that

  $$P'_2\rho'_2 \mid P''_1\rho''_1 \;\; \mathcal{S} \;\; Q'_2 \mid Q''_1 \tag{A.122}$$

  We know that there is R' such that

  $$Q'_1 \overset{\tau}{\Rightarrow} R' \xrightarrow{\breve{\ell}} Q'_2 \tag{A.123}$$

  Thus, by Rule $\langle \mathtt{Par}_L \rangle$ we can infer the following:

  $$Q'_1 \mid Q''_1 \overset{\tau}{\Rightarrow} R' \mid Q''_1$$

  Further, we can infer

  $$\langle \mathtt{Par}_L \rangle \; \frac{R' \xrightarrow{\breve{\ell}} Q'_2}{R' \mid Q''_1 \xrightarrow{\breve{\ell}} Q'_2 \mid Q''_1}$$

  Then, by the IH (A.121) and the definition of $\mathcal{S}$ (Definition 4.4.17) we know

  $$Q'_2 \in \widehat{\mathcal{C}}^{\rho'_2}_{\sigma'_1 \cdot \sigma'_2}(P'_2)$$

  Now, when $\ell = n?\langle V_1\rangle$ by the index substitution assertion we have $\sigma'_2 \in \mathsf{index}(\mathtt{fn}(P'_2\rho'_2))$ such that $\mathsf{next}(n_i) \in \sigma'_2$, $\sigma'_1 \cdot (\sigma'_2 \setminus \mathsf{next}(n_i)) = (\sigma'_2 \setminus \mathsf{next}(n_i)) \cdot \sigma'_1$, and Thus, the following holds

  $$\widehat{\mathcal{C}}^{\rho''_1 \cdot \rho'_2}_{\sigma'_2 \cdot \sigma''_1}(P''_1) = \widehat{\mathcal{C}}^{\rho''_1}_{\sigma''_1}(P''_1)$$

  $$\widehat{\mathcal{C}}^{\rho''_1 \cdot \rho'_2}_{\sigma'_2 \cdot \sigma''_1}(P'_2) = \widehat{\mathcal{C}}^{\rho'_2}_{\sigma'_2}(P'_2)$$

  So, by this, (A.118), and definition of $\mathcal{C}^{-}_{-}(\,-\,)$ we have

  $$Q''_1 \mid Q'_2 \in \widehat{\mathcal{C}}^{\rho''_1 \cdot \rho_2;}_{\sigma''_1 \cdot \sigma'_2}(P''_1 \mid P'_2) \tag{A.124}$$

  In sub-case $\ell = n?\langle V_1\rangle$ we may notice that $\sigma''_1 \cdot \sigma'_2 \in \mathsf{index}(\mathtt{fn}(P'_2 \mid P''_1)\rho'_2\rho''_1)$ and $\mathsf{next}(n_i) \in \sigma''_1 \cdot \sigma'_2$ as by the assumption we have $\bar{n} \notin \mathtt{fn}(P'_2\rho'_2 \mid P''_1\rho''_1)$. Thus, our assertion holds.

  Thus, by this and (A.124) the goal (A.122) follows. This concludes this sub-case.

- Sub-case $\ell \equiv (\nu\, \widetilde{m}_1)\, n!\langle V_1 \rangle$. This sub-case follows the essential steps of the previous sub-case. Here we omit details on index subsitutions as ther are similar to the first sub-case. By (A.119) and IH we know there is Q_2' such that $Q_1' \xRightarrow{(\nu\, \tilde{m}_2)\, n_i!\langle V_2 \rangle} Q_2'$ and

$$(\nu\, \widetilde{m}_1)\, (P_2' \parallel t \hookleftarrow_{\mathtt{H}} V_1)\rho_1' \;\mathcal{S}\; (\nu\, \widetilde{m}_2)\, (Q_2' \parallel t \hookleftarrow_{\mathtt{H}} V_2) \tag{A.125}$$

We should show that

$$(\nu\, \widetilde{m}_1)\, (P_2' \mid P_1'' \parallel t \hookleftarrow_{\mathtt{H}} V_1)\rho_1 \;\mathcal{S}\; (\nu\, \widetilde{m}_2)\, (Q_2' \mid Q_1'' \parallel t \hookleftarrow_{\mathtt{H}} V_2) \tag{A.126}$$

We pick R' as in the previous sub-case. So, we can infer the following transition:

$$\langle \mathtt{Par}_L \rangle \; \frac{R' \xrightarrow{(\nu\, \tilde{m}_2)\, n_i!\langle V_2 \rangle} Q_2'}{R' \mid Q_1'' \xrightarrow{(\nu\, \tilde{m}_2)\, n_i!\langle V_2 \rangle} Q_2' \mid Q_1''}$$

Next, by Table 4.3 we can infer the following. First, by (A.125) we know

$$Q_2' \parallel t \hookleftarrow_{\mathtt{H}} V_2 \in \widehat{\mathcal{C}}^{\rho_1'}_{\sigma_2'}(P_2' \parallel t \hookleftarrow_{\mathtt{H}} V_1)$$

By IH and assertion, we know $\mathsf{next}(n_i) \in \sigma_2'$. Similarly to the previous sub-case, we have

$$\widehat{\mathcal{C}}^{\rho_1''}_{\sigma_1'' \cdot \sigma_2'}(P_1'') = \mathcal{C}^{\rho_1''}_{\sigma_1''}(P_1'')$$

Thus, we have

$$(\nu\, \widetilde{m}_2)\, (Q_2' \mid Q_1'' \parallel t \hookleftarrow_{\mathtt{H}} V_2) \in \widehat{\mathcal{C}}^{\rho_1}_{\sigma_1'' \cdot \sigma_2'}((\nu\, \widetilde{m}_1)\, (P_2' \mid P_1'' \parallel t \hookleftarrow_{\mathtt{H}} V_1))$$

Further, by using the index substitution assertion we have

$$\sigma_2' \cdot \sigma_1'' \in \mathsf{index}(\mathtt{fn}(P_2'\rho_2' \mid P_1''\rho_1''))$$

Thus, (A.126) follows.

This concludes case $\langle \mathtt{Par}_L \rangle$.

5. Case $\langle \mathtt{Tau} \rangle$. We distinguish two sub-cases: (i) $P_1\rho_1 = P_1'\rho_1' \mid P_1''\rho_1''$ and (ii) $P_1\rho_1 = P_1'\rho_1' \parallel P_1''\rho_1''$ where one of parallel components is a trigger collection. Without loss of generality, we assume $\ell_1 = (\nu\, \widetilde{m}_1)\, \overline{n}!\langle V_1 \rangle$ and $\ell_2 = n?(V_1)$. The final rule in the inference tree is then as follows:

$$\langle \mathtt{Tau} \rangle \; \frac{P_1'\rho_1' \xrightarrow{\ell_1} P_2'\rho_2' \qquad P_1''\rho_1'' \xrightarrow{\ell_2} P_2''\rho_2'' \qquad \ell_1 \asymp \ell_2}{P_1'\rho_1' \mid P_1''\rho_1'' \xrightarrow{\tau} (\nu\, \widetilde{m}_1)\, (P_2'\rho_2' \mid P_2''\rho_2'')}$$

Let $\sigma_1 = \mathsf{index}(\widetilde{u})$ where $\widetilde{u} = \mathtt{fn}(P_1\{\tilde{W}/\tilde{x}\})$. Further, let σ_1' and σ_1'' such that

$$P_1\rho_1\sigma_1 = P_1'\rho_1'\sigma_1' \mid P_1''\rho_1''\sigma_1''$$

We know $\rho_1' = \{\tilde{W}_1/\tilde{y}\}$ and $\rho_1'' = \{\tilde{W}_2/\tilde{w}\}$. By the definition of $\mathcal{S}$ (Table 4.3) we have $Q_1 \in N_1 \cup N_2 \cup N_3$ where

$$\begin{aligned}
N_1 &= \{(\nu\, \widetilde{c}_k)\,(\overline{c_k}!\langle \widetilde{B}\rangle \mid \mathcal{B}^k_{\tilde{x}}(P_1'\sigma_1' \mid P_1''\sigma_1'')) : \widetilde{W}\sigma_1 \bowtie \widetilde{B}\}\\
N_2 &= \{(\nu\, \widetilde{c}_{k+1})\,(\overline{c_{k+1}}!\langle \widetilde{B}_1\rangle.\overline{c_{k+l+1}}!\langle \widetilde{B}_2\rangle \mid \mathcal{B}^{k+1}_{\tilde{y}}(P_1'\sigma_1') \mid \mathcal{B}^{k+l+1}_{\tilde{w}}(P_1''\sigma_1''))\\
&\qquad : \widetilde{W}_i\sigma_1 \bowtie \widetilde{B}_i, i \in \{1,2\}\}\\
N_3 &= \{R_1' \mid R_1'' : R_1' \in \mathcal{C}^{\tilde{W}_1}_{\tilde{y}}(P_1'\sigma_1'),\ R_1'' \in \mathcal{C}^{\tilde{W}_2}_{\tilde{w}}(P_1''\sigma_1'')\}
\end{aligned}$$

By Lemma 4.4.3, for $Q_1^1 \in N_1$ there exists $Q_1^2 \in N_2$ such that

$$Q_1^1 \overset{\tau}{\Rightarrow} Q_1^2 \overset{\tau}{\Rightarrow} Q_1' \mid Q_1''$$

where $Q_1' \mid Q_1'' \in N_3$, that is

$$Q_1' \in \widehat{\mathcal{C}}^{\rho_1'}_{\sigma_1'}(P_1') \tag{A.127}$$

$$Q_1'' \in \widehat{\mathcal{C}}^{\rho_1''}_{\sigma_1''}(P_1'') \tag{A.128}$$

Thus, in both cases we only consider how $Q_1' \mid Q_1''$ evolves. By the definition of $\mathcal{S}$ we have

$$P_1'\rho_1'\, \mathcal{S}\, Q_1' \tag{A.129}$$

$$P_1''\rho_1''\, \mathcal{S}\, Q_1'' \tag{A.130}$$

We have the following IH:

a) By (A.129) and IH there is Q_2' such that

$$Q_1' \xrightarrow{(\nu\, \widetilde{m}_1)\, \overline{n}_i!\langle V_2\rangle} Q_2' \tag{A.131}$$

and

$$(\nu\, \widetilde{m}_1)\,(P_2' \parallel t \hookleftarrow_{\mathtt{H}} V_1)\rho_1'\ \ \mathcal{S}\ \ (\nu\, \widetilde{m}_2)\,(Q_2' \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2) \tag{A.132}$$

Now, by our assertion, we have $\sigma_2' \in \mathsf{index}(\mathtt{fn}(P_2'\rho_2'))$ such that $\mathsf{next}(\overline{n}_i) \in \sigma_2'$, $\sigma_1' \cdot (\sigma_2' \setminus \mathsf{next}(\overline{n}_i)) = (\sigma_2' \setminus \mathsf{next}(\overline{n}_i)) \cdot \sigma_1'$, and

$$Q_2' \in \widehat{\mathcal{C}}^{\rho_2'}_{\sigma_2'}(P_2') \tag{A.133}$$

More precisely, we know that $\sigma_2' \subseteq \sigma_1' \cdot \mathsf{next}(\overline{n}_i) \subseteq \sigma_1\mathsf{next}(\overline{n}_i)$. So, we have

$$Q_2' \in \widehat{\mathcal{C}}^{\rho_2'}_{\sigma_1 \cdot \mathsf{next}(\overline{n}_i)}(P_2') \tag{A.134}$$

Moreover, let $\sigma_v \in \mathsf{index}(\mathtt{fn}(t \hookleftarrow_{\mathtt{H}} V_1\rho_1'))$ such that

$$t_1 \hookleftarrow_{\mathtt{H}} V_2 \in \widehat{\mathcal{C}}^{\rho_1'}_{\sigma_v}(t_1 \hookleftarrow_{\mathtt{H}} V_1) \tag{A.135}$$

We may notice that we have $\sigma_v \subseteq \sigma_1$.

b) By (A.130) and IH there is Q''_2 such that

$$Q''_1 \xRightarrow{n_j?(V_2)} Q''_2 \tag{A.136}$$

and

$$P''_2 \rho''_2 \ \mathcal{S} \ Q''_2 \tag{A.137}$$

Now, by (A.136) and the assertion on index substitutions, we have $\sigma''_2 \in \mathsf{index}(\mathtt{fn}(P''_2) \cup \rho''_2)$ such that $\mathsf{next}(n_j) \in \sigma''_2$, $\sigma'_1 \cdot (\sigma''_2 \setminus \mathsf{next}(n_j)) = (\sigma''_2 \setminus \mathsf{next}(n_j)) \cdot \sigma'_1$, and

$$Q''_2 \in \mathcal{C}^{\rho''_2}_{\sigma''_2}(P''_2) \tag{A.138}$$

More precisely, we know that $\sigma''_2 = \sigma''_1 \cdot \sigma_v \cdot \mathsf{next}(n_j) \subseteq \sigma_1 \cdot \mathsf{next}(n_j)$. So, we have

$$Q''_2 \in \widehat{\mathcal{C}}^{\rho''_2}_{\sigma_1 \cdot \mathsf{next}(n_j)}(P''_2) \tag{A.139}$$

Now, by (A.131) we know there is R' such that

$$Q'_1 \overset{\tau}{\Rightarrow} R' \xrightarrow{\breve{\ell}_1} Q'_2$$

where $\breve{\ell}_1 = (\nu\, \widetilde{m}_2)\, \overline{n}_i!\langle V_2 \rangle$. Similarly, by (A.136) there is R'' such that

$$Q''_1 \overset{\tau}{\Rightarrow} R'' \xrightarrow{\breve{\ell}_2} Q''_2$$

where $\breve{\ell}_2 = n_j?(V_2)$. By Rule $\textsc{Par}_\mathrm{L}$ and Rule $\textsc{Par}_\mathrm{R}$ we can infer the following:

$$Q'_1 \mid Q''_1 \overset{\tau}{\Rightarrow} R' \mid R''$$

Now, to proceed we must show $\breve{\ell}_1 \asymp \breve{\ell}_2$, which boils down to showing that indices of $\overline{n}_i$ and n_j match. For this, we distinguish two sub-cases: (i) $\neg\mathsf{tr}(\overline{n}_i)$ and $\neg\mathsf{tr}(n_j)$ and (ii) $\mathsf{tr}(\overline{n}_i)$ and $\mathsf{tr}(n_j)$. In the former sub-case, we have $\{^{\overline{n}_i}/_n\} \in \sigma_1$ and $\{^{n_j}/_n\} \in \sigma_1$, where $\sigma_1 = \mathsf{index}(\widetilde{u})$. Further, by this and and Definition 4.4.10 we know that $i = j$. Now, we consider the later case. By assumption that $P_1\{^{\widetilde{W}}/_{\tilde{x}}\}$ is well-typed, we know there Γ_1, Λ_1, and Δ_1 such that $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{^{\widetilde{W}}/_{\tilde{x}}\} \triangleright \diamond$ with $\mathsf{balanced}(\Delta_1)$, Thus, we have $n : S \in \Delta_1$ and $\overline{n} : T \in \Delta_1$ such that S **dual** T. Hence, by the definition of $[-\rangle$ (Definition 4.3.5) we have $i = [S\rangle = [T\rangle = j$. Hence, we can infer the following transition:

$$\langle \mathtt{Tau} \rangle \ \frac{R' \xrightarrow{\breve{\ell}_1} Q'_2 \qquad R'' \xrightarrow{\breve{\ell}_2} Q''_2 \qquad \breve{\ell}_1 \asymp \breve{\ell}_2}{(R' \mid R'') \xrightarrow{\tau} (\nu\, \widetilde{m}_2)\, (Q'_2 \mid Q''_2)}$$

Now, we should show that

$$(\nu\, \widetilde{m}_1)\, (P'_2\rho'_2 \mid P''_2\rho''_2) \ \mathcal{S} \ (\nu\, \widetilde{m}_2)\, (Q'_2 \mid Q''_2) \tag{A.140}$$

Now, by (A.134), (A.139), and $(P'_2 \mid P''_2)\rho'_2 \cdot \rho''_2 = P'_2\rho'_2 \mid P'_2\rho''_2$ we have

$$Q'_2 \mid Q''_2 \in \widehat{\mathcal{C}}^{\rho'_2 \cdot \rho''_2}_{\sigma_1 \cdot \mathsf{next}(n_i) \cdot \mathsf{next}(\overline{n}_i)}(P'_2 \mid P''_2)$$

Further, we have $\sigma_1 \cdot \mathsf{next}(n_i) \cdot \mathsf{next}(\overline{n}_i) \in \mathsf{index}(\mathtt{fn}((P'_2 \mid P''_2)\rho'_2 \cdot \rho''_2))$ This follow directly by the definition of σ_1, that is $\sigma_1 \in \mathsf{index}(\mathtt{fn}(P_1\rho_1))$, and Definition 4.4.10.

6. Case $\langle\texttt{New}\rangle$. In this case we know $P_1 = (\nu\, m : C)\, P_1'$. The final rule in the transition inference tree is as follows:

$$\langle\texttt{New}\rangle\ \frac{P_1'\rho_1 \xrightarrow{(\nu\,\widetilde{n}_1)\,u!\langle V_1\rangle} P_2\rho_2 \qquad m \in \texttt{fn}(V_1)}{(\nu\, m)\, P_1'\rho_1 \xrightarrow{(\nu\, m\cdot\widetilde{n}_1)\,u!\langle V_1\rangle} P_2\rho_1} \tag{A.141}$$

Let $\sigma_1 = \mathsf{index}(\texttt{fn}(P_1\rho_1))$. By the definition of $\mathcal{S}$ (Table 4.3) we have $Q_1 \in N_1$ where

$$N_1 = \{(\nu\,\widetilde{c}_r)\,(\nu\,\widetilde{c})\,(\nu\,\widetilde{m}_2)\,(\nu\,\widetilde{c}^m)\,R : R \in \mathcal{C}_{\tilde{x}}^{\widetilde{W}\sigma_1}(P_1'\sigma_1 \cdot \{^{m_1\overline{m}_1}/_{m\overline{m}}\})\}$$

where $\widetilde{c}_r = \mathsf{cr}(R)$, $\widetilde{m}_2 = (m_1, \ldots, m_{|\mathcal{G}(C)|})$, and $\widetilde{c}^m = c^m \cdot c^{\overline{m}}$ if $\texttt{tr}(C)$, otherwise $\widetilde{c}^m = \epsilon$. By IH, if $P_1'\rho_1\,\mathcal{S}\,Q_1'$ there are Q_2 and V_2 such that

$$Q_1' \xRightarrow{(\nu\,\widetilde{n}_2)\,u_i!\langle V_2\rangle} Q_2$$

and

$$(\nu\,\widetilde{n}_1)\,(P_2 \parallel t \hookleftarrow_{\texttt{H}} V_1)\rho_1\ \ \mathcal{S}\ \ (\nu\,\widetilde{n}_2)\,(Q_2 \parallel t \hookleftarrow_{\texttt{H}} V_2) \tag{A.142}$$

For $Q_1 \in N_1$ we should show that

$$Q_1 \xRightarrow{(\nu\,\widetilde{m}_2\cdot\widetilde{n}_2)\,u_i!\langle V_2\rangle} Q_2 \tag{A.143}$$

such that

$$(\nu\, m\cdot\widetilde{n}_1)\,(P_2 \parallel t \hookleftarrow_{\texttt{H}} V_1)\rho_1\ \ \mathcal{S}\ \ (\nu\,\widetilde{c}_r'\cdot\widetilde{m}_2\cdot\widetilde{n}_2)\,(Q_2 \parallel t \hookleftarrow_{\texttt{H}} V_2) \tag{A.144}$$

where $\widetilde{c}_r' = \mathsf{cr}(V_2)$. Note that by the definition we have $\mathsf{fpn}(V_2) = \emptyset$. By Lemma 4.4.3, we know there is R such that $P_1'\rho_1\,\mathcal{S}\,R$ and

$$Q_1' \overset{\tau}{\Rightarrow} R \xrightarrow{(\nu\,\tilde{n}_2)\,u_i!\langle V_2\rangle} Q_2 \tag{A.145}$$

Now, by rule $\langle\texttt{New}\rangle$ we have

$$Q_1 \overset{\tau}{\Rightarrow} (\nu\,\widetilde{c}_r')\,(\nu\,\widetilde{m}_2)\,R$$

Now, we need to apply Rule $\langle\texttt{New}\rangle$ $|\widetilde{m}_2|$ times to (A.145) to infer the following:

$$\langle\texttt{New}\rangle\ \frac{R' \xrightarrow{(\nu\,\widetilde{n}_2)\,u_i!\langle V_2\rangle} Q_2 \qquad \widetilde{m}_2 \subseteq \texttt{fn}(V_2)}{(\nu\,\widetilde{m}_2)\,R \xrightarrow{(\nu\,\widetilde{m}_2\cdot\widetilde{n}_2)\,u_i!\langle V_2\rangle} Q_2}$$

Therefore, the sub-goal (A.143) follows. Now, by (A.142) and by Definition 4.4.17 we can infer the following:

$$(\nu\,\widetilde{n}_2)\,(Q_2 \parallel t \hookleftarrow_{\texttt{H}} V_2) \equiv (\nu\,\widetilde{c}_r')\,(\nu\,\widetilde{n})\,(\mathcal{R}_{\tilde{v}} \mid R_2 \mid \mathcal{R}_{\tilde{w}} \parallel t \hookleftarrow_{\texttt{H}} V_2)$$

where $\widetilde{v} = \texttt{rn}(R_2)$, $\widetilde{w} = \texttt{rn}(V_2)$, $\widetilde{c}_r' = \mathsf{cr}(V_2)$, and

$$(\nu\,\widetilde{n})\,(\mathcal{R}_{\tilde{v}} \mid R_2 \mid \mathcal{R}_{\tilde{w}} \parallel t \hookleftarrow_{\texttt{H}} V_2) \in \widehat{\mathcal{C}}_{\sigma_1}^{\rho_1}((\nu\,\widetilde{n}_1)\,(P_2 \parallel t \hookleftarrow_{\texttt{H}} V_1))$$

By this and the definition of $\mathcal{C}_{-}^{-}(\,-\,)$ (Table 4.3) we have

$$(\nu\,\widetilde{m})\,(\nu\,\widetilde{c}^m)\,(\nu\,\widetilde{n})\,(\mathcal{R}_{\tilde{v}} \mid R_2 \mid \mathcal{R}_{\tilde{w}} \parallel t \hookleftarrow_{\texttt{H}} V_2) \in \widehat{\mathcal{C}}_{\sigma_1}^{\rho_1}((\nu\, m)\,(\nu\,\widetilde{n}_1)\,(P_2 \parallel t \hookleftarrow_{\texttt{H}} V_1))$$

Further, we may notice

$$(\nu\,\widetilde{m}_2\cdot\widetilde{n}_2)\,(Q_2 \parallel t \hookleftarrow_{\texttt{H}} V_2) \equiv (\nu\,\widetilde{c}_r)\,(\nu\,\widetilde{m}_2)\,(\nu\,\widetilde{n})\,(\mathcal{R}_{\tilde{v}} \mid R_2 \mid \mathcal{R}_{\tilde{w}} \parallel t \hookleftarrow_{\texttt{H}} V_2)$$

Therefore, the sub-goal (A.144) follows. This concludes case $\langle\texttt{New}\rangle$ and the proof.

$\square$

A.2.3 Proof of Lemma 4.4.7

Lemma 4.4.7. *Assume $P_1\{\tilde{W}/\tilde{x}\}$ is a process and $P_1\{\tilde{W}/\tilde{x}\}\,\mathcal{S}\,Q_1$.*

1. *Whenever $Q_1 \xrightarrow{(\nu\,\widetilde{m_2})\,n_i!\langle V_2\rangle} Q_2$, such that $\overline{n}_i \notin \mathtt{fn}(Q_1)$, then there exist P_2 and V_2 such that $P_1\{\tilde{W}/\tilde{x}\} \xrightarrow{(\nu\,\widetilde{m_2})\,n!\langle V_2\rangle} P_2$ and, for a fresh t,*
$$(\nu\,\widetilde{m_1})(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\{\tilde{W}/\tilde{x}\}\,\mathcal{S}\,(\nu\,\widetilde{m_2})(Q_2 \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2).$$
2. *Whenever $Q_1 \xrightarrow{n_i?(V_2)} Q_2$, such that $\overline{n}_i \notin \mathtt{fn}(Q_1)$, there exist P_2, V_2, and σ such that $P_1\{\tilde{W}/\tilde{x}\} \xrightarrow{n?(V_1)} P_2$ where $V_1\sigma \bowtie V_2$ and $P_2\,\mathcal{S}\,Q_2$.*
3. *Whenever $Q_1 \xrightarrow{\tau} Q_2$ either (i) $P_1\{\tilde{W}/\tilde{x}\}\,\mathcal{S}\,Q_2$ or (ii) there exists P_2 such that $P_1 \xrightarrow{\tau} P_2$ and $P_2\,\mathcal{S}\,Q_2$.*

Proof (Sketch). By transition induction. First, we analyze the case of non-essential prefixes, which induce τ-actions that do not correspond to actions in P_1. This concerns the sub-case (i) of Part 3. This directly follows by Lemma 4.4.3, that is by the fact that $\mathcal{C}^{\tilde{W}}_{\tilde{x}}(P_1)$ is closed under transitions on non-essential prefixes.
Now, assume $Q_1 \xrightarrow{\ell} Q_2$ when ℓ is an essential prefix. This is mainly the converse of the proof of Lemma 4.4.4 noting that there are no essential actions in $\mathcal{C}^{\tilde{W}}_{\tilde{x}}(P_1)$ not matched in P_1. We consider only one case:

- Case $\langle\mathtt{Snd}\rangle$. In this case we know $P_1 = n!\langle V_1\rangle.P_2$. We distinguish two sub-cases: (i) $\neg\mathsf{tr}(u_i)$ and (ii) $\mathsf{tr}(u_i)$. In both sub-cases, we distinguish two kinds of an object value V_1: (a) $V_1 \equiv x$, such that $\{V_x/x\} \in \{\tilde{W}/\tilde{x}\}$ and (b) $V_1 = \lambda y : C.\, P'$, that is V_1 is a pure abstraction. We only consider sub-case (a).
  Let $\widetilde{W_1}$, $\widetilde{W_2}$, $\tilde{y}$, and $\tilde{w}$ such that
$$P_1\{\tilde{W}/\tilde{x}\} = n!\langle V_1\{\tilde{W_1}/\tilde{y}\}\rangle.P_2\{\tilde{W_2}/\tilde{w}\}$$
  Let $\sigma_1 \in \mathsf{index}(\tilde{u})$ where $\tilde{u} = \mathtt{fn}(P_1\{\tilde{W}/\tilde{x}\})$ such that $\{n_i/n\} \in \sigma_1$. Also, let $\sigma_2 = \sigma_1 \cdot \mathsf{next}(n_i)$. When P_1 is not a trigger, by the definition of $\mathcal{S}$ (Table 4.3), for both sub-cases, we have $Q_1 \in N_1$ where:
$$N_1 = \{(\nu\,\tilde{c}_r)\,(\nu\,\tilde{c}_{k+1})\,\mathcal{R}_{\tilde{v}} \mid n_i!\langle V_2\rangle.\overline{c_{k+1}}!\langle\widetilde{B}_2\rangle \mid \mathcal{B}^{k+1}_{\tilde{w}}(P_2\sigma_2) : V_1\sigma\{\widetilde{W_1}/\tilde{y}\} \bowtie V_2,\ \widetilde{W}_2 \bowtie \widetilde{B}_2\}$$
  For $Q_1 \in N_1$ we have the following transition inference tree:
$$\langle\mathtt{Snd}\rangle\ \frac{}{Q_1 \xrightarrow{n_i!\langle V_1\rangle} Q_2}$$
  where
$$Q_2 = (\nu\,\tilde{c}_r)\,(\nu\,\tilde{c}_{k+1})\,\overline{c_{k+1}}!\langle\widetilde{B}_2\rangle \mid \mathcal{B}^{k}_{\tilde{w}}(P_2\sigma_2)$$
  We have
$$\langle\mathtt{Snd}\rangle\ \frac{}{P_1\{\tilde{W}/\tilde{x}\} \xrightarrow{n!\langle V_1\{\widetilde{W_1}/\tilde{y}\}\rangle} P_2\{\tilde{W_2}/\tilde{w}\}}$$
  We should show that
$$(\nu\,\widetilde{m_1})(P_2 \parallel t \hookleftarrow_{\mathtt{H}} V_1)\ \mathcal{S}\ (\nu\,\widetilde{m_2})(Q_2 \parallel t_1 \hookleftarrow_{\mathtt{H}} V_2)$$
  This immediately follows by the definition of $\mathcal{S}$ and Table 4.3. This concludes $\mathtt{Snd}$ case.

As can be seen the proof of this part is essentially the inverse of the proof of Lemma 4.4.4. We just need to show that $\mathcal{C}_{\tilde{x}}^{\tilde{W}\sigma}(P_1\sigma)$ does not introduce extra actions on essential prefixes not present in P_1. This is evident by the inspection of the definition of $\mathcal{C}_{-}^{-}(\,-\,)$. Briefly, only in the case of the input and the output prefix $\mathcal{J}_{-}^{-}(\,-\,)$ introduce actions that mimic those prefixes. Remaining cases only introduce actions on non-essential prefixes (τ-actions on propagator names). $\square$

B Appendix to Chapter 5

B.1 Additional Examples

B.1.1 First Decomposition: Core Fragment

Example B.1.1 (Core Fragment)**.** Let P be a π process which incorporates name-passing and implements channels u and $\overline{w}$ with types $S = !\langle \overline{T} \rangle;\mathtt{end}$ and $T = ?(\mathsf{Int});!\langle \mathsf{Bool} \rangle;\mathtt{end}$, respectively:

$$P = (\nu\, u : S)\, (u!\langle w \rangle.\overline{w}?(t).\overline{w}!\langle \mathsf{odd}(t) \rangle.\mathbf{0} \mid \overline{u}?(x).x!\langle 5 \rangle.x?(b).\mathbf{0}) = (\nu\, u)\, (A \mid B)$$

The degree of P is $\lfloor P \rceil = 25$. Then, the decomposition of P into a collection of first-order processes typed with minimal session types is:

$$\begin{aligned}
\mathcal{F}(P) &= (\nu\, c_1, \ldots, c_{25})\, (\overline{c_1}!\langle\rangle.\mathbf{0} \mid \mathcal{A}^1_\epsilon(P\sigma)) \\
&= (\nu\, c_1, \ldots, c_{25})\, (\overline{c_1}!\langle\rangle.\mathbf{0} \mid \mathcal{A}^1_\epsilon((\nu\, u)\, (A \mid B)\sigma)) \\
&= (\nu\, c_1, \ldots, c_{25})\, (\overline{c_1}!\langle\rangle.\mathbf{0} \mid (\nu\, u_1)\, \mathcal{A}^1_\epsilon((A \mid B)\sigma')), \\
&\quad \text{where } \sigma = \mathsf{init}(\mathtt{fn}(P)),\ \sigma' = \sigma \cdot \{{}^{u_1\overline{u_1}}/_{u\overline{u}}\}
\end{aligned}$$

We have:

$$\mathcal{A}^1_\epsilon((A \mid B)\sigma')) = c_1?().\overline{c_2}!\langle\rangle.\overline{c_{13}}!\langle\rangle \mid \mathcal{A}^2_\epsilon(A\sigma') \mid \mathcal{A}^{13}_\epsilon(B\sigma')$$

We use some abbreviations for subprocesses of A and B :

$$\begin{aligned}
A' &= \overline{w_1}?(t).A'' & A'' &= \overline{w}_1!\langle \mathsf{odd}(t) \rangle.\mathbf{0} \\
B' &= x_1!\langle 5 \rangle.B'' & B'' &= x_1?(b).\mathbf{0}
\end{aligned}$$

The breakdown of A is:

$$\begin{aligned}
\mathcal{A}^2_\epsilon(A) &= c_2?().(\nu\, a_1)\, (u_1!\langle a_1 \rangle.(\overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid a_1?(y_1).y_1?(z_1).\overline{c_3}!\langle z_1 \rangle \mid \\
&\qquad c_3?(z_1).z_1?(x).\overline{c_4}!\langle x \rangle \mid c_4?(x).(\nu\, s)\, (x!\langle s \rangle.\overline{s}!\langle w_1, w_2 \rangle))) \\
\mathcal{A}^5_\epsilon(A') &= c_5?().\overline{w_1}?(y_2).\overline{c_6}!\langle y_2 \rangle \mid (\nu\, s_1)\, (c_6?(y_2).\overline{c_7}!\langle y_2 \rangle.\overline{c_8}!\langle\rangle \mid \\
&\qquad c_7?(y_2).(\nu\, s')\, (y_2!\langle s' \rangle.\overline{s'}!\langle s_1 \rangle.\mathbf{0})) \mid c_8?().(\nu\, a_2)\, (\overline{s_1}!\langle a_2 \rangle.(\overline{c_{10}}!\langle\rangle \mid c_{10}?().\mathbf{0} \mid \\
&\qquad a_2?(y_3).y_3?(t_1).(\overline{c_9}!\langle\rangle \mid \mathcal{A}^9_\epsilon(A'')))) \\
\mathcal{A}^9_\epsilon(A'') &= c_9?().(\nu\, a)\, (\overline{w}_2!\langle a \rangle.(\overline{c_{12}}!\langle\rangle \mid c_{12}?().\mathbf{0} \mid \\
&\qquad\quad a?(y).y?(z_1).\overline{c_{11}}!\langle z_1 \rangle \mid c_{10}?(z_1).z_1?(x).\overline{c_{11}}!\langle x \rangle \mid c_{11}?(x).(\nu\, s)\, (x!\langle s \rangle.\overline{s}!\langle \mathsf{odd}(t) \rangle)))
\end{aligned}$$

The breakdown of B is:

$$\begin{aligned}
\mathcal{A}^{13}_\epsilon(B) &= c_{13}?().\overline{u}_1?(y_4).\overline{c_{14}}!\langle y_4 \rangle \mid (\nu\, s_1)\, (c_{14}?(y).\overline{c_{15}}!\langle y \rangle.\overline{c_{16}}!\langle\rangle \mid \\
&\qquad c_{15}?(y_4).(\nu\, s'')\, (y_4!\langle s'' \rangle.\overline{s''}!\langle s_1 \rangle.\mathbf{0}) \mid c_{16}?().(\nu\, a_3)\, (s_1!\langle a_3 \rangle.(\overline{c_{21}}!\langle\rangle \mid \\
&\qquad c_{21}?().\mathbf{0} \mid a_3?(y_5).y_5?(x_1, x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B'))))) \\
\mathcal{A}^{17}_\epsilon(B') &= c_{17}?().(\nu\, a_4)\, (x_1!\langle a_4 \rangle.(\overline{c_{20}}!\langle\rangle \mid \mathcal{A}^{20}_\epsilon(B'') \mid a_4?(y_6).y_6?(z_1).\overline{c_{18}}!\langle z_1 \rangle \mid \\
&\qquad c_{18}?(z_1).z_1?(x).\overline{c_{19}}!\langle x \rangle \mid c_{19}?(x).(\nu\, s''')\, (x!\langle s''' \rangle.\overline{s'''}!\langle 5 \rangle))) \\
\mathcal{A}^{20}_\epsilon(B'') &= c_{20}?().x_2?(y).\overline{c_{21}}!\langle y \rangle \mid \\
&\qquad\quad (\nu\, s_1)\, (c_{21}?(y).\overline{c_{22}}!\langle y \rangle.\overline{c_{23}}!\langle\rangle \mid c_{22}?(y).(\nu\, s)\, (y!\langle s \rangle.\overline{s}!\langle s_1 \rangle) \mid \\
&\qquad\qquad c_{23}?(\widetilde{x}).(\nu\, a)\, (\overline{s_1}!\langle a \rangle.(\overline{c_{25}}!\langle\rangle \mid c_{25}?().\mathbf{0} \mid \\
&\qquad\qquad a?(y').y'?(b_1).(\overline{c_{24}}!\langle\rangle \mid c_{24}?().\mathbf{0}))))
\end{aligned}$$

Names $\overline{w}_1$ and $\overline{w}_2$ are typed, respectively, with

$$M_1 = ?(\langle ?(?(\langle ?(\mathsf{Int});\mathtt{end}\rangle);\mathtt{end});\mathtt{end}\rangle);\mathtt{end}$$
$$M_2 = !\langle\langle ?(?(\langle ?(\mathsf{Bool});\mathtt{end}\rangle);\mathtt{end});\mathtt{end}\rangle\rangle;\mathtt{end}$$

Name u_1 is typed with $!\langle\langle ?(?(\langle ?(\overline{M}_1, \overline{M}_2);\mathtt{end}\rangle);\mathtt{end});\mathtt{end}\rangle\rangle;\mathtt{end}$. Now let us observe the reduction chain. The collection of processes synchronizes on c_1, c_2, c_{13} after three reductions.

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^3 &(\nu\, \widetilde{c})\,(\,(\nu\, a_1)\,(\,u_1!\langle a_1\rangle.\,(\overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid a_1?(y_1).y_1?(z_1).\overline{c_3}!\langle z_1\rangle \mid c_3?(z_1).z_1?(x).\overline{c_4}!\langle x\rangle \mid \\ &c_4?(x).(\nu\, s)\,(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle)))) \mid \overline{u_1}?(y_4).\,\overline{c_{14}}!\langle y_4\rangle \mid (\nu\, s_1)\,(c_{14}?(y).\overline{c_{15}}!\langle y\rangle.\overline{c_{16}}!\langle\rangle \mid \\ &c_{15}?(y_4).(\nu\, s'')\,(y_4!\langle s''\rangle.\overline{s''}!\langle s_1\rangle.\mathbf{0}) \mid c_{16}?().(\nu\, a_3)\,(s_1!\langle a_3\rangle.(\overline{c_{21}}!\langle\rangle \mid \\ &c_{21}?().\mathbf{0} \mid a_3?(y_5).y_5?(x_1, x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B'))))) \\ &\text{where } \widetilde{c} = (c_3, \dots, c_{12}, c_{14}, \dots, c_{25})\end{aligned}$$

Then, the broken down process A communicates with process B through channel u_1 by passing name a_1 (highlighted above). Here we notice that the original transmission of value w is not immediately mimicked on channel u, but it is delegated to some other channel through series of channel redirections starting with the transmission of name a_1. Further, received name a_1 is locally propagated by c_{14} and c_{15}. This represents redundant communications on propagators induced by breaking down sequential prefixes produced by two encodings $[\![\cdot]\!]^1_g$ and $[\![\cdot]\!]^2$ (not present in the original process). Another synchronization occurs on c_{16}.

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^7 &(\nu\, \widetilde{c}_*)\,(\nu\, a_1)\,(\,\overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid a_1?(y_1).\,y_1?(z_1).\overline{c_3}!\langle z_1\rangle \mid c_3?(z_1).z_1?(x).\overline{c_4}!\langle x\rangle \mid \\ &c_4?(x).(\nu\, s)\,(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle) \mid (\nu\, s_1)\,((\nu\, s'')\,(\,a_1!\langle s''\rangle.\,\overline{s''}!\langle s_1\rangle.\mathbf{0}) \mid (\nu\, a_3)\,(s_1!\langle a_3\rangle.(\overline{c_{21}}!\langle\rangle \mid \\ &c_{21}?().\mathbf{0} \mid a_3?(y_5).y_5?(x_1, x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B')))))) \\ &\text{where } \widetilde{c}_* = (c_3, \dots, c_{12}, c_{17}, \dots, c_{25})\end{aligned}$$

The next step involves communication on a_1: session name s'' is passed and substitutes variable y_1.

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^8 &(\nu\, \widetilde{c}_*)\,(\nu\, s'')\,(\,\overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid s''?(z_1).\,\overline{c_3}!\langle z_1\rangle \mid c_3?(z_1).z_1?(x).\overline{c_4}!\langle x\rangle \mid \\ &c_4?(x).(\nu\, s)\,(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle) \mid (\nu\, s_1)\,(\,\overline{s''}!\langle s_1\rangle.\,\mathbf{0} \mid (\nu\, a_3)\,(s_1!\langle a_3\rangle.(\overline{c_{21}}!\langle\rangle \mid c_{21}?().\mathbf{0} \mid \\ &a_3?(y_5).y_5?(x_1, x_2).(\,\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B'))))))\end{aligned}$$

The process then synchronizes on channel s''. After z_1 is replaced by s_1, it is further sent to the next parallel process through the propagator c_3.

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^{10} &(\nu\, \widetilde{c}_{**})\,(\nu\, s_1)\,(\,\overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid s_1?(x).\,\overline{c_4}!\langle x\rangle \mid c_4?(x).(\nu\, s)\,(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle) \mid \\ &(\nu\, a_3)\,(\,s_1!\langle a_3\rangle.\,(\overline{c_{21}}!\langle\rangle \mid c_{21}?().\mathbf{0} \mid \\ &a_3?(y_5).y_5?(x_1, x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B'))))) \\ &\text{where } \widetilde{c}_{**} = (c_4, \dots, c_{12}, c_{17}, \dots, c_{25})\end{aligned}$$

Communication on s_1 leads to variable x being substituted by name a_3, which is then passed on c_4 to the next process. In addition, inaction is simulated by synchronization on c_{21}.

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^{13} &(\nu\, \widetilde{c}_\bullet)\,(\nu\, a_3)\,(\,\overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid (\nu\, s)\,(\,a_3!\langle s\rangle.\,\overline{s}!\langle w_1, w_2\rangle) \mid \\ &a_3?(y_5).\,y_5?(x_1, x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B'))\,) \\ &\text{where } \widetilde{c}_\bullet = (c_5, \dots, c_{12}, c_{17}, \dots, c_{25})\end{aligned}$$

Now, the distribution of the decomposition of w from one process to another can finally be simulated by two reductions: first, a synchronization on a_3 sends the endpoint of session s, which replaces variable y_5; afterwards, the dual endpoint is used to send the names w_1, w_2, substituting the variables x_1, x_2.

$$\begin{aligned}
\mathcal{F}(P) \longrightarrow^{14} & (\nu\, \widetilde{c}_{\bullet\bullet})\,(\nu\, s)\,(\,\overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid \\
& \boxed{\overline{s}!\langle w_1, w_2\rangle} \mid \boxed{s?(x_1, x_2).}\,(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B')))\\
\mathcal{F}(P) \longrightarrow^{15} & (\nu\, \widetilde{c}_{\bullet\bullet})\,(\,\overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid \\
& \overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B')\{w_1w_2/x_1x_2\}) = Q\\
& \text{where } \widetilde{c}_{\bullet\bullet} = (c_5, \ldots, c_{12}, c_{17}, \ldots, c_{25})
\end{aligned}$$

Here, we remark that as prefix $s?(x_1, x_2)$ bounds variables x_1, x_2 in the breakdown of the continuation $(\mathcal{A}^{17}_\epsilon(B'))$. Thus, there is no need for propagators to pass contexts: propagators here only serve to enforce the ordering of actions. On the other hand, this rely on arbitrary process nesting which is induced by the final application of encoding $[\![\cdot]\!]^2$ in the composition. Thus the trio structure is lost.

Undoubtedly, the first action of the original first-order process has been simulated. We may notice that in Q names w_1, w_2 substitute x_1, x_2 and subsequent τ-action on w can be simulated on name w_1. The following reductions follow the same pattern. Thus, the outcome of our decomposition function is a functionally equivalent process that is typed with minimal session types.

B.1.2 First Decomposition: Labelled Choice

We now illustrate how the approach in Section 5.2 can be extended to account for labelled (deterministic) choice (selection and branching constructs). In HOπ, these constructs are denoted

$$u \triangleleft l.P \qquad u \triangleright \{l_i : P_i\}_{i\in I}$$

with reduction rule

$$n \triangleleft l_j.Q \mid \overline{n} \triangleright \{l_i : P_i\}_{i\in I} \longrightarrow Q \mid P_j \quad (j \in I)$$

Example B.1.2 (First Decomposition: Labelled Choice)**.** Let P and Q be π processes which incorporate branching and selection:

$$\begin{aligned}
P &= u \triangleright \{l_1 : (\nu\, h)\,(h!\langle m\rangle.h?(t).\mathbf{0} \mid \overline{h}?(x).\overline{h}!\langle x\rangle.\mathbf{0}), l_2 : s!\langle m\rangle.\overline{s}?(x).\mathbf{0}\}\\
Q &= \overline{u} \triangleleft l_1.\mathbf{0}
\end{aligned}$$

Let R denote the restricted parallel composition of P and Q: $R = (\nu\, u)\,(P \mid Q)$. The respective process degrees have the same values as before. The decomposition of R is:

$$\begin{aligned}
\mathcal{F}(R) &= (\nu\, c_1, \ldots, c_5)\,(\overline{c_1}!\langle\rangle.\mathbf{0} \mid \mathcal{A}^1_\epsilon((\nu\, u)\,(P \mid Q)\sigma)_g), \text{ where } \sigma = \{u_1\overline{u_1}/u\overline{u}\}\\
&= (\nu\, c_1, \ldots, c_5)\,(\overline{c_1}!\langle\rangle.\mathbf{0} \mid (\nu\, u_1)\,(c_1?().\overline{c_2}!\langle\rangle.\overline{c_3}!\langle\rangle \mid \mathcal{A}^2_\epsilon(P)_g \mid \mathcal{A}^3_\epsilon(Q)_g))
\end{aligned}$$

We consider P and Q separately. We begin with the breakdown of Q.

$$\begin{aligned}
\mathcal{A}^3_\epsilon(Q)_g = {} & c_3?().(\nu\, a_1)\,(\overline{c_4}!\langle a_1\rangle.((\nu\, u_2)\,(c_4?(y).(\nu\, s)\,(y!\langle s\rangle.\overline{s}!\langle\overline{u}_2\rangle)) \mid c_5?().\mathbf{0} \mid \\
& a_1?(y').y'?(y_1).u_1 \triangleleft l_1.u_1?(z).\overline{c_5}!\langle\rangle.(\nu\, s')\,(z!\langle s'\rangle.\overline{s}'!\langle y_1\rangle)))
\end{aligned}$$

The breakdown of process P is:

$$\begin{aligned}\mathcal{A}^2_\epsilon(P)_g = c_2?().u_1 \triangleright \{&l_1 : (\nu\, a_2)\, u_1!\langle a_2\rangle.a_2?(y'').y''?(y_1).(\nu\, c_3,\dots,c_{21})\, (\overline{c_3}!\langle\rangle \mid \mathcal{A}^3_\epsilon(P_1\{y_1/u_1\})_g),\\ &l_2 : (\nu\, a_2)\, u_1!\langle a_2\rangle.a_2?(y'').y''?(y_1).(\nu\, c_3,\dots,c_{11})\, (\overline{c_3}!\langle\rangle \mid \mathcal{A}^3_\epsilon(P_2\{y_1/u_1\})_g)\}\end{aligned}$$

As in the previous approach of this example, we are interested in the breakdown of P_1, because it belongs to the selected branch.

$$\begin{aligned}\mathcal{A}^3_\epsilon(P_1\{y_1/u_1\})_g &= (\nu\, h_1, h_2)\, \mathcal{A}^3_\epsilon(h_1!\langle m_1\rangle.h_1?(t).\mathbf{0} \mid \overline{h_1}?(x).\overline{h_1}!\langle x\rangle.\mathbf{0})_g\\ &= (\nu\, h_1, h_2)\, (c_3?().\overline{c_4}!\langle\rangle.\overline{c_{13}}!\langle\rangle \mid \mathcal{A}^4_\epsilon(h_1!\langle m_1\rangle.h_1?(t).\mathbf{0})_g \mid \mathcal{A}^{13}_\epsilon(\overline{h_1}?(x).\overline{h_1}!\langle x\rangle.\mathbf{0})_g)\end{aligned}$$

$$\begin{aligned}\mathcal{A}^4_\epsilon(h_1!\langle m_1\rangle._g h_1?(t).\mathbf{0}) &= c_4?().(\nu\, a_3)\, (h_1!\langle a_3\rangle.(\overline{c_7}!\langle\rangle \mid \mathcal{A}^7_\epsilon(h_2?(t).\mathbf{0})_g \mid a_3?(y_2).y_2?(z_1).\overline{c_5}!\langle z_1\rangle \mid\\ &\qquad c_5?(z_1).z_1?(x).\overline{c_6}!\langle x\rangle \mid c_6?(x).(\nu\, s_1)\, (x!\langle s_1\rangle.\overline{s_1}!\langle m_1\rangle.\mathbf{0})))\\ \mathcal{A}^7_\epsilon(h_2?(t).\mathbf{0})_g &= c_7?().h_2?(y_3).\overline{c_8}!\langle y_3\rangle \mid (\nu\, s_2)(c_8?(y_3).\overline{c_9}!\langle y_3\rangle.\overline{c_{10}}!\langle\rangle \mid\\ &\qquad c_9?(y_3).(\nu\, s_3)\, (y_3!\langle s_3\rangle.\overline{s_3}!\langle s_2\rangle.\mathbf{0}) \mid c_{10}?().(\nu\, a_4)\, (\overline{s_2}!\langle a_4\rangle.(\overline{c_{12}}!\langle\rangle \mid\\ &\qquad c_{12}?().\mathbf{0} \mid a_4?(y_4).y_4?(t_1).\overline{c_{11}}!\langle\rangle \mid c_{11}?().\mathbf{0}\ ))\\ \mathcal{A}^{13}_\epsilon(\overline{h_1}?(x).\overline{h_1}!\langle x\rangle.\mathbf{0})_g &= c_{13}?().\overline{h_1}?(y_5).\overline{c_{14}}!\langle y_5\rangle \mid (\nu\, s_4)(c_{14}?(y_5).\overline{c_{15}}!\langle y_5\rangle.\overline{c_{16}}!\langle\rangle \mid\\ &\qquad c_{15}?(y_5).(\nu\, s_5)\, (y_5!\langle s_5\rangle.\overline{s_5}!\langle s_4\rangle.\mathbf{0}) \mid c_{16}?().(\nu\, a_5)\, (\overline{s_4}!\langle a_5\rangle.(\overline{c_{21}}!\langle\rangle \mid\\ &\qquad c_{21}?().\mathbf{0} \mid a_5?(y_6).y_6?(x_1).\overline{c_{17}}!\langle x_1\rangle \mid \mathcal{A}^{17}_{x_1}(\overline{h_2}!\langle x_1\rangle.\mathbf{0})_g\ )))\\ \mathcal{A}^{17}_{x_1}(\overline{h_2}!\langle x_1\rangle.\mathbf{0})_g &= c_{17}?(x_1).(\nu\, a_6)\, (\overline{h_2}!\langle a_6\rangle.(\overline{c_{20}}!\langle x_1\rangle \mid c_{20}?().\mathbf{0} \mid a_6?(y_7).y_7?(z_2).\overline{c_{18}}!\langle z_2\rangle \mid\\ &\qquad c_{18}?(z_2).z_2?(x').\overline{c_{19}}!\langle x'\rangle \mid c_{19}?(x').(\nu\, s_6)\, (x'!\langle s_6\rangle.\overline{s_6}!\langle x_1\rangle.\mathbf{0})))\end{aligned}$$

In order to verify that the decomposition of R simulates the behaviour of the initial process, we refer to its reduction chain. The first transitions involve synchronization on c_1, c_2, c_3 and c_4. The trigger name a_2 is passed through propagator c_4, such that it replaces variable y.

$$\begin{aligned}\mathcal{F}(R) \longrightarrow^4\ &(\nu\, c_5)\, (\nu\, u_1)\, (u_1 \triangleright \{l_1 : (\nu\, a_2)\ \ u_1!\langle a_2\rangle.a_2?(y'').y''?(y_1).(\nu\, c_3,\dots,c_{21})\, (\overline{c_3}!\langle\rangle \mid\\ &\mathcal{A}^3_\epsilon(P_1\{y_1/u_1\})_g),\ l_2 : (\nu\, a_2)\ \ u_1!\langle a_2\rangle.a_2?(y'').y''?(y_1).(\nu\, c_3,\dots,c_{11})\, (\overline{c_3}!\langle\rangle \mid\\ &\mathcal{A}^3_\epsilon(P_2\{y_1/u_1\})_g)\} \mid (\nu\, a_1)\, ((\nu\, u_2)\, ((\nu\, s)\, (a_1!\langle s\rangle.\overline{s}!\langle\overline{u_2}\rangle)) \mid c_5?().\mathbf{0} \mid\\ &a_1?(y').y'?(y_1).u_1 \triangleleft l_1.u_1?(z).\overline{c_5}!\langle\rangle.(\nu\, s')\, (z!\langle s'\rangle.\overline{s'}!\langle y_1\rangle)))\end{aligned}$$

The next step involves synchronization on a_1, which leads to the variable y' being substituted by session name s.

$$\begin{aligned}\mathcal{F}(R) \longrightarrow^5\ &(\nu\, c_5)\, (\nu\, u_1)\, (u_1 \triangleright \{l_1 : (\nu\, a_2)\ \ u_1!\langle a_2\rangle.a_2?(y'').y''?(y_1).(\nu\, c_3,\dots,c_{21})\, (\overline{c_3}!\langle\rangle \mid\\ &\mathcal{A}^3_\epsilon(P_1\{y_1/u_1\})_g),\ l_2 : (\nu\, a_2)\ \ u_1!\langle a_2\rangle.a_2?(y'').y''?(y_1).(\nu\, c_3,\dots,c_{11})\, (\overline{c_3}!\langle\rangle \mid\\ &\mathcal{A}^3_\epsilon(P_2\{y_1/u_1\})_g)\} \mid (\nu\, s)\, ((\nu\, u_2)\, (\overline{s}!\langle\overline{u_2}\rangle) \mid c_5?().\mathbf{0} \mid\\ &s?(y_1).u_1 \triangleleft l_1.u_1?(z).\overline{c_5}!\langle\rangle.(\nu\, s')\, (z!\langle s'\rangle.\overline{s'}!\langle y_1\rangle)))\end{aligned}$$

Now, session name s and its dual endpoint can communicate - channel name $\overline{u_2}$ is sent, replacing y_1.

$$\begin{aligned}\mathcal{F}(R) \longrightarrow^6\ &(\nu\, c_5)\, (\nu\, u_1)\, (u_1 \triangleright \{l_1 : (\nu\, a_2)\ \ u_1!\langle a_2\rangle.a_2?(y'').y''?(y_1).(\nu\, c_3,\dots,c_{21})\, (\overline{c_3}!\langle\rangle \mid\\ &\mathcal{A}^3_\epsilon(P_1\{y_1/u_1\})_g),\ l_2 : (\nu\, a_2)\ \ u_1!\langle a_2\rangle.a_2?(y'').y''?(y_1).(\nu\, c_3,\dots,c_{11})\, (\overline{c_3}!\langle\rangle \mid\\ &\mathcal{A}^3_\epsilon(P_2\{y_1/u_1\})_g)\} \mid (\nu\, u_2)\, (c_5?().\mathbf{0} \mid u_1 \triangleleft l_1.u_1?(z).\overline{c_5}!\langle\rangle.(\nu\, s')\, (z!\langle s'\rangle.\overline{s'}!\langle\overline{u_2}\rangle)))\end{aligned}$$

At this point, the action of choosing the first branch can be simulated by selecting label l_1 on channel name u_1.

$$\begin{aligned}\mathcal{F}(R) \longrightarrow^7\ &(\nu\, c_5)\, (\nu\, u_1)\, ((\nu\, a_2)\ \ u_1!\langle a_2\rangle.a_2?(y'').y''?(y_1).(\nu\, c_3,\dots,c_{21})\, (\overline{c_3}!\langle\rangle \mid\\ &\mathcal{A}^3_\epsilon(P_1\{y_1/u_1\})_g) \mid (\nu\, u_2)\, (c_5?().\mathbf{0} \mid u_1?(z).\overline{c_5}!\langle\rangle.(\nu\, s')\, (z!\langle s'\rangle.\overline{s'}!\langle\overline{u_2}\rangle)))\end{aligned}$$

The following two reductions consist of synchronizations on u_1 and c_5. As a result, variable z is replaced by the trigger name a_2.

$$\begin{aligned}\mathcal{F}(R) \longrightarrow^9 \; &(\nu\, a_2)\,(a_2?(y'').y''?(y_1).(\nu\, c_3,\dots,c_{21})\,(\overline{c_3}!\langle\rangle \mid \\ &\mathcal{A}^3_\epsilon(P_1\{y_1/u_1\})_g) \mid (\nu\, u_2)\,((\nu\, s')\,(a_2!\langle s'\rangle.\overline{s}'!\langle\overline{u_2}\rangle)))\end{aligned}$$

Communication on a_2 allows for synchronization on s'. Channel a_2 receives the session name s', which further receives the dual endpoint $\overline{u_2}$.

$$\begin{aligned}\mathcal{F}(R) \longrightarrow^{11} \; &(\nu\, c_3,\dots,c_{21})\,(\nu\, u_2)\,(\overline{c_3}!\langle\rangle \mid \mathcal{A}^3_\epsilon(P_1\{\overline{u_2}/u_1\})_g) \\ = \; &(\nu\, c_3,\dots,c_{21})\,(\nu\, u_2)\,(\overline{c_3}!\langle\rangle \mid (\nu\, h_1,h_2)\,(c_3?().\overline{c_4}!\langle\rangle.\overline{c_{13}}!\langle\rangle \mid \\ &\mathcal{A}^4_\epsilon(h_1!\langle m_1\rangle.h_1?(t).\mathbf{0})_g \mid \mathcal{A}^{13}_\epsilon(\overline{h_1}?(x).\overline{h_1}!\langle x\rangle.\mathbf{0})_g))\end{aligned}$$

Remarkably, the obtained result reflects the broken down version of process P_1. Following three more reductions, in which c_3, c_4 and c_{13} synchronize with their respective duals, the breakdown of each parallel sub-process will be activated. We refrain from presenting the remaining reduction steps, as they represent a standard case of name-passing. Clearly, the decomposed process $\mathcal{F}(R)$ behaves in a similar manner to the initial process R.

B.2 Appendix to Section 5.2

B.2.1 Proof of Theorem 5.2.1

Lemma 5.2.1 (Typability of Breakdown). *Let P be an initialized π process. If* $\Gamma;\Delta,\Delta_\mu \vdash P \triangleright \diamond$*, then* $\mathcal{H}(\Gamma'),\Phi';\mathcal{H}(\Delta),\Theta' \vdash \mathcal{A}^k_\epsilon(P)_g \triangleright \diamond$*, where:*

- $k > 0$;
- $\widetilde{r} = \mathtt{dom}(\Delta_\mu)$;
- $\Phi' = \prod_{r\in\tilde{r}} c^r : \langle\langle ?(\mathcal{R}'^{\star}(\Delta_\mu(r)));\mathtt{end}\rangle\rangle$;
- $\mathsf{balanced}(\Theta')$ *with*

$$\mathtt{dom}(\Theta') = \{c_k, c_{k+1}, \ldots, c_{k+\lfloor P \rceil -1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\lfloor P \rceil -1}}\}$$

such that $\Theta'(c_k) = ?(\cdot);\mathtt{end}$.

Proof.

$$\begin{aligned}
&\Gamma;\Delta,\Delta_\mu \vdash P \triangleright \diamond && \text{(Assumption)} && \text{(B.1)}\\
&\llparenthesis\Gamma\rrparenthesis^1;\llparenthesis\Delta\rrparenthesis^1,\llparenthesis\Delta_\mu\rrparenthesis^1 \vdash \llbracket P\rrbracket^1_g \triangleright \diamond && \text{(Theorem 5.1 [41], (10))} && \text{(B.2)}\\
&\mathcal{G}(\llparenthesis\Gamma\rrparenthesis^1),\Phi;\ \mathcal{G}(\llparenthesis\Delta\rrparenthesis^1),\Theta \vdash \mathcal{B}^k_\epsilon(\llbracket P\rrbracket^1_g) \triangleright \diamond && \text{(Lemma 4.3.1, (11))} && \text{(B.3)}\\
&\llbracket\mathcal{G}(\llparenthesis\Gamma\rrparenthesis^1)\rrbracket^2,\llbracket\Phi\rrbracket^2;\llbracket\mathcal{G}(\llparenthesis\Delta\rrparenthesis^1)\rrbracket^2,\ \llbracket\Theta\rrbracket^2 \vdash \llbracket\mathcal{B}^k_\epsilon(\llbracket P\rrbracket^1_g)\rrbracket^2 \triangleright \diamond && \text{(Theorem 5.2 [41], (12))} && \text{(B.4)}\\
&\mathcal{H}(\Gamma),\Phi';\mathcal{H}(\Delta),\Theta' \vdash \mathcal{A}^k_\epsilon(P)_g \triangleright \diamond && \text{(Definition of } \mathcal{A}^k_\epsilon(\,\cdot\,)_g, && \text{(B.5)}\\
& && \text{Definitions 5.2.5 and 5.2.8, (4))} &&
\end{aligned}$$

□

B.2.2 Proof of Theorem 5.2.1

Theorem 5.2.1 (Minimality Result for π). *Let P be a closed π process, with* $\widetilde{u} = \mathtt{fn}(P)$ *and* $\widetilde{v} = \mathtt{rn}(P)$*. If* $\Gamma;\Delta,\Delta_\mu \vdash P \triangleright \diamond$*, where Δ_μ only involves recursive session types, then* $\mathcal{H}(\Gamma\sigma);\mathcal{H}(\Delta\sigma),\mathcal{H}(\Delta_\mu\sigma) \vdash \mathcal{F}(P) \triangleright \diamond$*, where* $\sigma = \{^{\mathsf{init}(\tilde{u})}/_{\tilde{u}}\}$.

Proof.

$$\begin{aligned}
&\Gamma;\Delta \vdash P \triangleright \diamond && \text{(Assumption)} && \text{(B.6)}\\
&\Gamma\sigma;\Delta\sigma \vdash P\sigma \triangleright \diamond && \text{(Lemma 2.2.1, (6))} && \text{(B.7)}\\
&\mathcal{H}(\Gamma\sigma),\Phi';\mathcal{H}(\Delta\sigma),\Theta' \vdash \mathcal{A}^k_\epsilon(P\sigma)_g \triangleright \diamond && \text{(}P\text{ is initialized, Lemma 5.2.1)} && \text{(B.8)}
\end{aligned}$$

To complete the proof, let us construct a well-formed derivation tree by using the appropriate typing rules for the higher-order calculus.

$$\text{Par } (|\widetilde{v}|-1 \text{ times})\ \dfrac{\text{for } r\in\widetilde{v} \qquad \mathcal{H}(\Gamma\sigma),\Phi';\widetilde{r}:\mathcal{H}(S) \vdash c^r?(b).(\nu\, s)\,(b!\langle s\rangle.\overline{s}!\langle\widetilde{r}\rangle) \triangleright \diamond}{\mathcal{H}(\Gamma\sigma),\Phi';\mathcal{H}(\Delta_\mu\sigma) \vdash \prod_{r\in\tilde{v}} P^r} \tag{B.9}$$

where $S = \Delta_\mu(r)$. By the definition of Φ' we have $c^r : \langle\langle ?(\mathcal{R}'^{\star}(\Delta_\mu(r)));\mathtt{end}\rangle\rangle \in \Phi'$ and by Definition 5.2.5 we have $\mathcal{R}'^{\star}(\Delta_\mu(r)) = \mathcal{H}(S)$, so the right-hand of (B.9) is well-typed.

$$\text{Par}\ \dfrac{\text{Send}\ \dfrac{\text{Nil}\ \dfrac{}{\mathcal{H}(\Gamma\sigma), \Phi'; \mathcal{H}(\Delta\sigma) \vdash \mathbf{0} \triangleright \diamond}}{\mathcal{H}(\Gamma\sigma), \Phi'; \mathcal{H}(\Delta\sigma), \overline{c_k}!\langle\cdot\rangle; \mathtt{end} \vdash \overline{c_k}!\langle\cdot\rangle} \qquad (\text{B.9})}{\mathcal{H}(\Gamma\sigma), \Phi'; \mathcal{H}(\Delta\sigma), \mathcal{H}(\Delta_\mu\sigma), \overline{c_k}!\langle\cdot\rangle; \mathtt{end} \vdash \overline{c_k}!\langle\cdot\rangle.\mathbf{0} \mid \prod_{r\in\tilde{v}} P^r} \tag{B.10}$$

$$\text{PolyResS}\ \dfrac{\text{Par}\ \dfrac{(\text{B.10}) \qquad (\text{B.8})}{\mathcal{H}(\Gamma\sigma), \Phi'; \mathcal{H}(\Delta\sigma), \mathcal{H}(\Delta_\mu\sigma), \overline{c_k}!\langle\cdot\rangle; \mathtt{end}, \Theta' \vdash \overline{c_k}!\langle\cdot\rangle.\mathbf{0} \mid \prod_{r\in\tilde{v}} P^r \mid \mathcal{A}^k_\epsilon(P\sigma)_g \triangleright \diamond}}{\mathcal{H}(\Gamma\sigma); \mathcal{H}(\Delta\sigma), \mathcal{H}(\Delta_\mu\sigma) \vdash (\nu\,\tilde{c})\,(\nu\,\tilde{c}_r)\,(\prod_{r\in\tilde{v}} P^r \mid \overline{c_k}!\langle\rangle.\mathbf{0} \mid \mathcal{A}^k_\epsilon(P\sigma)_g)}$$

□

B.3 Appendix to Section 5.3

B.3.1 Proof of Lemma 5.3.1

Lemma 5.3.1. *If P is in normal form then $\#(P) \geq \frac{5}{3} \cdot \#^*(P)$.*

Proof. Because $\#(Q) \geq \lfloor P \rceil + 2 \cdot |\mathsf{brn}(P)|$, it suffices to show that $\lfloor P \rceil + 2 \cdot |\mathsf{brn}(P)| \geq \frac{5}{3} \cdot \#^*(P)$. The proof is by induction on structure of P. We show one base case (output prefix followed by inaction) and three inductive cases (input, restriction, and parallel composition); other cases are similar:

- Case $P = u!\langle x \rangle.\mathbf{0}$. Then $\lfloor P \rceil = 4$, $|\mathsf{brn}(P)| = 0$, and $\lfloor P \rceil^* = 2$. So, we have $\lfloor P \rceil + 2 \cdot |\mathsf{brn}(P)| \geq \frac{5}{3} \cdot \#^*(P)$
- Case $P = u!\langle x \rangle.P'$ with $P' \not\equiv \mathbf{0}$. By IH we know $\lfloor P' \rceil + \mathsf{brn}(P') \geq \frac{5}{3} \cdot \#^*(P')$. We know $\lfloor P \rceil = \lfloor P' \rceil + 3$, $\lfloor P \rceil^* = \lfloor P' \rceil^* + 1$ and $\#_X(P) = \#_X(P')$. So, $\lfloor P' \rceil + 2 \cdot |\mathsf{brn}(P')| + 3 \geq \frac{5}{3} \cdot \#^*(P) + 1 > \frac{5}{3} \cdot (\#^*(P) + 1)$.
- Case $P = (\nu\, r : S)\, P'$ with $\mathsf{tr}(S)$. Then $\lfloor P \rceil = \lfloor P' \rceil$ (by Definition 5.2.2) and $|\mathsf{brn}(P)| = |\mathsf{brn}(P')| + 1$. Further, we have $\#_X(P) = \#_X(P')$ and $\lfloor P \rceil^* = \lfloor P' \rceil^* + 1$. Now, by IH we can conclude that $\lfloor P \rceil + 2 \cdot (|\mathsf{brn}(P)| + 1) \geq \frac{5}{3} \cdot \#^*(P) + 1 > \frac{5}{3} \cdot (\#^*(P) + 1)$.
- Case $P = P_1 \mid \ldots \mid P_n$. By IH we know

  $$\lfloor P_i \rceil + 2 \cdot |\mathsf{brn}(P_i)| \geq \frac{5}{3} \cdot \#^*(P_i) \tag{B.11}$$

  for $i \in \{1, \ldots, n\}$. We know $\lfloor P \rceil = \sum_{i=1}^n \lfloor P_i \rceil + (n-1)$, $|\mathsf{brn}(P)| = \sum_{i=1}^n |\mathsf{brn}(P_i)|$, and $\lfloor P \rceil^* = \sum_{i=1}^n \#^*(P_i) + (n-1)$. So, we should show

  $$\sum_{i=1}^n \lfloor P_i \rceil + 2 \cdot \sum_{i=1}^n |\mathsf{brn}(P_i)| + (n-1) \geq \frac{5}{3} \cdot (\sum_{i=1}^n \#^*(P_i) + (n-1)) \tag{B.12}$$

  That is,

  $$\sum_{i=1}^n (\lfloor P_i \rceil + 2 \cdot |\mathsf{brn}(P_i)| + 1) \geq \frac{5}{3} \cdot \sum_{i=1}^n (\#^*(P_i) + 1)$$

  Equivalent to

  $$\sum_{i=1}^n (\lfloor P_i \rceil + 2 \cdot |\mathsf{brn}(P_i)| + 1) \geq \sum_{i=1}^n (\frac{5}{3} \cdot (\#^*(P_i) + 1))$$

We show that for each $i = \{1, \ldots, n\}$ the following holds:

$$\lfloor P_i \rceil + 2 \cdot |\mathsf{brn}(P_i)| + 1 \geq \frac{5}{3} \cdot (\#^*(P_i) + 1)$$

That is

$$A = \frac{\lfloor P_i \rceil + 2 \cdot |\mathsf{brn}(P_i)| + 1}{\#^*(P_i) + 1} \geq \frac{5}{3}$$

As P is in the normal form, we know that $P_i \equiv \alpha.P_i'$ where α is some sequential prefix. So, by Definition 5.2.2 and Definition 5.3.3 we know that for some p^* and p we have $\lfloor P_i \rceil^* = p^* + \lfloor P_i' \rceil^*$, $\#_X(P_i) = \#_X(P_i')$, and $\lfloor P_i \rceil = p + \lfloor P_i' \rceil$. We can distinguish two sub-cases: (i) $P_i' \not\equiv \mathbf{0}$ and (ii) otherwise. We consider sub-case (i). By (B.11) we have:

$$A \geq \frac{\frac{5}{3} \cdot \#^*(P_i') + p + 1}{\#^*(P_i') + p^* + 1} \geq \frac{5}{3}$$

We need to find prefix α such that p and $\frac{p}{p^*}$ are the least. We notice that for the output prefix we have $p = 3$ and $\frac{p}{p^*} = \frac{3}{1}$. So, the following holds

$$\frac{\frac{5}{3} \cdot \#^*(P_i') + 4}{\#^*(P_i') + 2} = \frac{5}{3} \cdot \frac{\#^*(P_i') + \frac{12}{5}}{\#^*(P_i') + 2} \geq \frac{5}{3}$$

Now, we consider sub-case (ii) when $P_i' = \mathbf{0}$. In this case we have $\lfloor P_i' \rceil + 2 \cdot \mathsf{brn}(P_i') = \#^*(P_i') = 1$. We pick p and p^* as in the previous sub-case. So, we have

$$\frac{\lfloor P_i \rceil + 2 \cdot |\mathsf{brn}(P_i)| + 1}{\#^*(P_i) + 1} = \frac{1 + 3 + 1}{2 + 1} = \frac{5}{3}$$

So, we can conclude that inequality (B.12) holds.

$\square$

Lemma B.3.1. *Let $\widetilde{r}$ be tuple of channel names and S a recursive session type. If $\widetilde{r} : \mathcal{R}^\star(!\langle C \rangle;S)$ and $k = \iota(!\langle C \rangle;S)$ then $r_k : \mu \mathsf{t}.!\langle \mathcal{G}(C) \rangle;\mathsf{t}$.*

Lemma B.3.2 (Typing Broken-down Variables)**.** *If $\Gamma; \Delta \vdash z_i \triangleright C$ then $\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta) \vdash \widetilde{z} \triangleright \mathcal{H}^*(C)$ where $\widetilde{z} = (z_i, \ldots, z_{i+|\mathcal{H}^*(C)|-1})$.*

B.3.2 Proof of Lemma 5.3.2

Lemma 5.3.2 (Typability of Auxiliary Breakdown: $\mathcal{A}^{k}_{\widetilde{y}}(\cdot)_g$)**.** *Let P be an initialized process. If $\Gamma \cdot X : \Delta_\mu; \Delta \vdash P \triangleright \diamond$ then:*

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \Theta \vdash \mathcal{A}^{k}_{\widetilde{y}}(P)_g \triangleright \diamond \quad (k > 0)$$

where

- $\widetilde{x} \subseteq \mathtt{fn}(P)$ *such that* $\Delta \setminus \widetilde{x} = \emptyset$*;*
- $\widetilde{y} = \widetilde{v} \cdot \widetilde{m}$*, where* $\widetilde{m} = \mathtt{codom}(g)$ *and* $\widetilde{v}$ *is such that* $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{v}, \widetilde{x})$ *holds;*
- $\Theta = \Theta_\mu, \Theta_X(g)$ *where*

- $\mathtt{dom}(\Theta_\mu) = \{c^r_k, c^r_{k+1}, \ldots, c^r_{k+\lfloor P \rceil^* -1}\} \cup \{\overline{c^r_{k+1}}, \ldots, \overline{c^r_{k+\lfloor P \rceil^* -1}}\}$
- *Let* $\widetilde{N} = (\mathcal{G}(\Gamma), \mathcal{G}(\Delta_\mu \cdot \Delta))(\widetilde{y})$. *Then*

$$\Theta_\mu(c^r_k) = \begin{cases} \mu\mathsf{t}.?(\widetilde{N});\mathsf{t} & \textit{if } g \neq \emptyset \\ \langle \widetilde{N} \rangle & \textit{otherwise} \end{cases}$$

- $\mathsf{balanced}(\Theta_\mu)$
- $\Theta_X(g) = \bigcup_{X \in \mathtt{dom}(g)} c^r_X : \langle \widetilde{M}_X \rangle$ *where* $\widetilde{M}_X = (\mathcal{G}(\Gamma), \mathcal{G}(\Delta_\mu))(g(X))$.

Proof. By the induction of the structure of P. We consider five cases, taking $g \neq \emptyset$; the analysis when $g = \emptyset$ is similar.

1. Case $P = X$. The only rule that can be applied here is RVar:

$$\text{RVar}\ \frac{}{\Gamma \cdot X : \Delta; \Delta \vdash X \triangleright \diamond} \tag{B.13}$$

Let $\widetilde{x} = \emptyset$ as $\mathtt{fn}(P) = \emptyset$. So, $\widetilde{y} = \widetilde{m}$ where $\widetilde{m} = g(X)$. Since $\lfloor X \rceil^* = 1$ we have $\Theta_\mu = \{c^r_k : \mu\mathsf{t}.?(\widetilde{N});\mathsf{t}\}$ where $\widetilde{N} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta))(\widetilde{y})$. In this case $\Delta_\mu = \Delta$, thus $\widetilde{N} = \widetilde{M}$. We shall then prove the following judgment:

$$\mathcal{H}^*(\Gamma \setminus x); \Theta \vdash \widehat{\mathcal{A}}^k_{\widetilde{y}}(X)_g \triangleright \diamond \tag{B.14}$$

By Table 5.4, we have:

$$\widehat{\mathcal{A}}^k_{\widetilde{y}}(X)_g = \mu X.c^r_k?(\widetilde{y}).c^r_X!\langle \widetilde{y} \rangle.X$$

$$\text{Req}\ \frac{\text{RVar}\ \dfrac{}{\mathcal{H}^*(\Gamma \setminus x), X : \Theta; \Theta \vdash X} \qquad \text{PolySend}\ \dfrac{}{\mathcal{H}^*(\Gamma \setminus x), X : \Theta; \widetilde{y} : \widetilde{M} \vdash \widetilde{y} \triangleright \widetilde{M}}}{\mathcal{H}^*(\Gamma \setminus x), X : \Theta; c^r_X : \langle \widetilde{M} \rangle, \widetilde{y} : \widetilde{M} \vdash c^r_X!\langle \widetilde{y} \rangle.X} \tag{B.15}$$

The following tree proves this case:

$$\text{Rec}\ \frac{\text{Rcv}\ \dfrac{(\text{B.15}) \qquad \text{PolySend}\ \dfrac{}{\mathcal{H}^*(\Gamma \setminus x), X : \Theta; \widetilde{y} : \widetilde{M} \vdash \widetilde{y} \triangleright \widetilde{M}}}{\mathcal{H}^*(\Gamma \setminus x), X : \Theta; \Theta \vdash c^r_k?(\widetilde{y}).c^r_X!\langle \widetilde{y} \rangle.X \triangleright \diamond}}{\mathcal{H}^*(\Gamma \setminus x); \Theta \vdash \mu X.c^r_k?(\widetilde{y}).c^r_X!\langle \widetilde{y} \rangle.X \triangleright \diamond} \tag{B.16}$$

where $\Theta = \Theta_\mu, \overline{c^r_X} : \mu\mathsf{t}.!\langle \widetilde{M} \rangle;\mathsf{t}$.

2. Case $P = u_i!\langle z_j \rangle.P'$. Let $u_i : C$. We distinguish three sub-cases: (i) $C = S = \mu\mathsf{t}.!\langle C_z \rangle;S'$, (ii) $C = S = !\langle C_z \rangle;S'$, and (iii) $C = \langle C_z \rangle$. We consider first two sub-cases, as (iii) is shown similarly. The only rule that can be applied here is Send:

$$\text{Send}\ \frac{\Gamma \cdot X : \Delta_\mu; \Delta \cdot u_i : S' \vdash P' \triangleright \diamond \qquad \Gamma \cdot X : \Delta_\mu; \Delta_z \vdash z_j \triangleright C}{\Gamma \cdot X : \Delta_\mu; \Delta \cdot u_i : S \cdot \Delta_z \vdash u_i!\langle z_j \rangle.P' \triangleright \diamond} \tag{B.17}$$

Then, by IH on the right assumption of (B.17) we have:

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}'); \Theta' \vdash \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}'}(P')_g \triangleright \diamond \tag{B.18}$$

where $\widetilde{x}' \subseteq \mathtt{fn}(P)$ such that $(\Delta, r : S') \setminus \widetilde{x}' = \emptyset$, and $\widetilde{y}' = \widetilde{v}' \cup \widetilde{m}$ where $\mathtt{indexed}_{\Gamma,\Delta,r:S'}(\widetilde{v}', \widetilde{x}')$. Also, $\Theta' = \Theta'_\mu, \Theta_X$ where Θ'_μ such that $\mathsf{balanced}(\Theta'_\mu)$ with

$$\mathtt{dom}(\Theta'_\mu) = \{c^r_{k+1}, c^r_{k+2}, \ldots, c^r_{k+\lfloor P' \rceil^* +1}\} \cup \{\overline{c^r_{k+2}}, \ldots, \overline{c^r_{k+\lfloor P' \rceil^* -1}}, \overline{c^r_{k+\lfloor P' \rceil^* +1}}\}$$

and $\Theta'_\mu(c^r_{k+1}) = \mu \mathtt{t}.?(\widetilde{N}');\mathtt{t}$ where $\widetilde{N}' = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta_\mu \cdot \Delta \cdot u_i : S))(\widetilde{y}')$. By applying Lemma B.3.2 on the second assumption of (B.17) we have:

$$\mathcal{H}^*(\Gamma \cdot X : \Delta_\mu); \mathcal{H}^*(\Delta_z) \vdash \widetilde{z} \triangleright \mathcal{H}^*(C_z) \triangleright \diamond \tag{B.19}$$

Let $\sigma = \mathtt{next}(u_i)$ and in sub-case (i) $\sigma_1 = \{\widetilde{n}/\widetilde{u}\}$ where $\widetilde{n} = (u_{i+1}, \ldots, u_{i+\mathcal{H}^*(S)})$ and $\widetilde{u} = (u_i, \ldots, u_{i+\mathcal{H}^*(S)-1})$, otherwise (ii) $\sigma_1 = \epsilon$. We define $\widetilde{x} = \widetilde{x}', z$ and $\widetilde{y} = \widetilde{y}'\sigma_1, \widetilde{z} \cdot u_i$. By construction $\widetilde{x} \subseteq P$ and $(\Delta \cdot u_i : S \cdot \Delta_z) \setminus \widetilde{x} = \emptyset$. Further, we may notice that $\widetilde{y} = \widetilde{v} \cdot \widetilde{m}$ such that $\mathtt{indexed}_{\Gamma,\Delta,u_i:S,\Delta_z}(\widetilde{v}, \widetilde{x})$ and $\widetilde{y}' = \widetilde{m} \cup \mathtt{fnb}(P', \widetilde{y})$. Let $\Theta = \Theta_\mu, \Theta_X$ where

$$\Theta_\mu = \Theta'_\mu, c^r_k : \mu \mathtt{t}.?(\widetilde{N});\mathtt{t}, \overline{c^r_{k+1}} : \mu \mathtt{t}.!\langle \widetilde{N}' \rangle;\mathtt{t}$$

where $\widetilde{N} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta_\mu \cdot \Delta \cdot u_i : S \cdot \Delta_z))(\widetilde{y})$. By construction and since $\lfloor P \rceil^* = \lfloor P' \rceil^* + 1$ we have

$$\mathtt{dom}(\Theta_\mu) = \{c^r_k, c^r_{k+1}, \ldots, c^r_{k+\lfloor P \rceil^*-1}\} \cup \{\overline{c^r_{k+1}}, \ldots, \overline{c^r_{k+\lfloor P \rceil^*-1}}\}$$

and $\mathsf{balanced}(\Theta_\mu)$. By Table 5.4 we have:

$$\widehat{\mathcal{A}}^k_{\widetilde{y}}(P)_g = \mu X.c^r_k?(\widetilde{y}).u_l!\langle \widetilde{z} \rangle.c^r_{k+1}!\langle \widetilde{y}'\sigma_1 \rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}'\sigma_1}(P'\sigma)_g \tag{B.20}$$

where in sub-case (i) $l = i$ and in sub-case (ii) $l = \iota(S)$. We shall prove the following judgment:

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \Theta \vdash \widehat{\mathcal{A}}^k_{\widetilde{y}}(P)_g \triangleright \diamond \tag{B.21}$$

Let $\Delta_1 = \Delta, u_i : S, \Delta_z$ and $\widetilde{u}_i = \mathtt{bn}(u_i : S)$. We use some auxiliary sub-trees:

$$\text{RVar}\ \dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta; \Theta \vdash X \triangleright \diamond} \tag{B.22}$$

$$\text{PolySend}\ \dfrac{(B.22) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta; \mathcal{H}^*(\Delta, u_i\sigma : S') \vdash \widetilde{y}'\sigma_1 \triangleright \widetilde{N}'}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta; \Theta, \mathcal{H}^*(\Delta, u_i\sigma : S') \vdash \overline{c^r_{k+1}}!\langle \widetilde{y}'\sigma_1 \rangle.X \triangleright \diamond} \tag{B.23}$$

Here, in typing the right-hand assumption, we may notice that in sub-case (i) $\mathcal{H}^*(u_i : S) = \widetilde{u}_i :!\langle \mathcal{H}^*(C_z) \rangle;\mathtt{end}, \mathcal{H}^*(S')$ and $\mathcal{H}^*(u_i\sigma : S') = \widetilde{u}_{i+1} : \mathcal{H}^*(S')$ where $\widetilde{u}_{i+1} = (u_{i+1}, \ldots, u_{i+|\mathcal{H}^*(S')|})$. So the right-hand side follows by Definition 5.3.9 and Lemma 2.2.1. Otherwise, in sub-case (ii) by Definition 5.3.2 and Figure 5.5 we know $\mathcal{H}^*(u_i : S) = \mathcal{H}^*(u_i\sigma : S')$ and by definition $\widetilde{u}_i \subseteq \widetilde{m} \subseteq \widetilde{y}'\sigma_1$.

$$\text{PolySend}\ \dfrac{(B.23) \qquad (B.19)}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta; \Theta, \mathcal{H}^*(\Delta_1) \vdash u_l!\langle \widetilde{z} \rangle.\overline{c^r_{k+1}}!\langle \widetilde{y}'\sigma_1 \rangle.X \triangleright \diamond} \tag{B.24}$$

$$\text{Rec}\ \dfrac{\text{PolyRcv}\ \dfrac{(B.24) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta; \mathcal{H}^*(\Delta_1) \vdash \widetilde{y} \triangleright \widetilde{N}}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta; \Theta \vdash c^r_k?(\widetilde{y}).u_l!\langle \widetilde{z} \rangle.\overline{c^r_{k+1}}!\langle \widetilde{y}'\sigma_1 \rangle.X \triangleright \diamond}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \Theta \vdash \mu X.c^r_k?(\widetilde{y}).u_l!\langle \widetilde{z} \rangle.\overline{c^r_{k+1}}!\langle \widetilde{y}'\sigma_1 \rangle.X \triangleright \diamond} \tag{B.25}$$

We may notice that by Definition 5.3.2 and Figure 5.5 we have $\mathcal{H}^*(\Delta_1) = \mathcal{H}^*(\Delta), \widetilde{u}_i : \mathcal{H}^*(S), \mathcal{H}^*(\Delta_z)$. So, in sub-case (i) as $u_i = u_l$ we have $\mathcal{H}^*(\Delta_1)(u_l) = !\langle \mathcal{H}^*(C_z) \rangle;\mathtt{end}$. In sub-case (ii), by Lemma B.3.1 and as $u_l = u_{\iota(S)}$ we know $\mathcal{H}^*(\Delta_1)(u_l) = \mu\mathsf{t}.!\langle\mathcal{H}^*(C_z)\rangle;\mathsf{t}$. The following tree proves this case:

$$\text{Par}\ \frac{\text{(B.25)} \qquad \text{(B.18)}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \Theta \vdash \mu X.c^r_k?(\widetilde{y}).u_l!\langle\widetilde{z}\rangle.c^r_{k+1}!\langle\widetilde{y}'\sigma_1\rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}'\sigma_1}(P'\sigma)_g \triangleright \diamond} \tag{B.26}$$

Note that we have used the following for the right assumption of (B.26):

$$\widehat{\mathcal{A}}^{k+1}_{\widetilde{y}'}(P') \equiv_\alpha \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}'\sigma_1}(P'\sigma)$$

3. Case $P = u_i?(z).P'$. We distinguish three sub-cases: (i) $C = S = \mu\mathsf{t}.?(C_z);S'$, (ii) $C = S = ?(C_z);S'$, and (iii) $C = \langle C_z\rangle$. The only rule that can be applied here is Rcv:

$$\text{Rcv}\ \frac{\Gamma \cdot X : \Delta_\mu; \Delta, u_i : S', \Delta_z \vdash P' \triangleright \diamond \qquad \Gamma \cdot X : \Delta_\mu; \Delta_z \vdash z \triangleright C}{(\Gamma \setminus z) \cdot X : \Delta_\mu; \Delta, u_i : S \vdash u_i?(z).P' \triangleright \diamond} \tag{B.27}$$

Let $\widetilde{x}' \subseteq \mathtt{fn}(P')$ such that $(\Delta \cdot u_i : S' \cdot \Delta_z) \setminus \widetilde{x}' = \emptyset$ and $\widetilde{y}' = \widetilde{v} \cup \widetilde{m}$ such that $\mathtt{indexed}_{\Gamma,\Delta,u_i:S'}(\widetilde{v}', \widetilde{x}')$. Also, $\Theta' = \Theta'_\mu, \Theta_X(g)$ where $\mathsf{balanced}(\Theta'_\mu)$ with

$$\mathtt{dom}(\Theta'_\mu) = \{c^r_{k+1}, c^r_{k+2}, \ldots, c^r_{k+\lfloor P'\rceil^*+1}\} \cup \{\overline{c^r_{k+2}}, \ldots, \overline{c^r_{k+\lfloor P'\rceil^*+1}}\}$$

and $\Theta'_\mu(c^r_{k+1}) = \mu\mathsf{t}.?(\widetilde{N}');\mathsf{t}$, where $\widetilde{N}' = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta_\mu \cdot \Delta \cdot u_i : S))(\widetilde{y}')$. Then, by IH on the right assumption of (B.17) we have:

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}'); \Theta'_\mu \vdash \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}'}(P') \triangleright \diamond \tag{B.28}$$

By applying Lemma B.3.2 to the second assumption of (B.17), we have:

$$\mathcal{H}^*(\Gamma) \cdot X : \Theta_\mu; \mathcal{H}^*(\Delta_z) \vdash \widetilde{z} \triangleright \mathcal{H}^*(C) \tag{B.29}$$

Let $\sigma = \mathsf{next}(u_i)$ and in sub-case (i) $\sigma_1 = \{\widetilde{n}/\widetilde{u}\}$ where $\widetilde{n} = (u_{i+1}, \ldots, u_{i+\mathcal{H}^*(S)})$ and $\widetilde{u} = (u_i, \ldots, u_{i+\mathcal{H}^*(S)-1})$, otherwise in sub-case (ii) $\sigma_1 = \epsilon$. We define $\widetilde{x} = \widetilde{x}'\sigma \setminus z$ and $\widetilde{y} = \widetilde{y}'\sigma_1 \setminus \widetilde{z}$ with $|\widetilde{z}| = |\mathcal{H}^*(C)|$. By construction $\widetilde{x} \subseteq \mathtt{fn}(P)$ and $(\Delta, r : S) \setminus \widetilde{x} = \emptyset$. Further, we may notice that $\widetilde{y} = \widetilde{v} \cdot \widetilde{m}$, where $\widetilde{v}$ is such that $\mathtt{indexed}_{\Gamma,\Delta,r:S,\Delta_z}(\widetilde{v}, \widetilde{x})$ and $\widetilde{y} = \widetilde{m} \cup \mathtt{fnb}(P', \widetilde{y}\widetilde{z})$. Let $\Theta = \Theta_\mu, \Theta_X$ where

$$\Theta_\mu = \Theta'_\mu, c^r_k : \mu\mathsf{t}.?(\widetilde{N});\mathsf{t}, \overline{c^r_{k+1}} : \mu\mathsf{t}.!\langle\widetilde{N}'\rangle;\mathsf{t}$$

where $\widetilde{N} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta_\mu \cdot \Delta \cdot u_i : S, \Delta_z))(\widetilde{y})$. By construction and since $\lfloor P\rceil^* = \lfloor P'\rceil^* + 1$ we have

$$\mathtt{dom}(\Theta_\mu) = \{c^r_k, c^r_{k+1}, \ldots, c^r_{k+\lfloor P\rceil^*-1}\} \cup \{\overline{c^r_{k+1}}, \ldots, \overline{c^r_{k+\lfloor P\rceil^*-1}}\}$$

and $\mathsf{balanced}(\Theta_\mu)$. By Table 5.4, we have:

$$\widehat{\mathcal{A}}^k_{\widetilde{y}}(P)_g = \mu X.c^r_k?(\widetilde{y}).u_l?(\widetilde{z}).c^r_{k+1}!\langle\widetilde{y}'\sigma_1\rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}}(P'\sigma)_g \tag{B.30}$$

where in sub-case (i) $l = i$ and in sub-case (ii) $l = \iota(S)$. Let $\Gamma_1 = \Gamma \setminus \widetilde{x}$. We shall prove the following judgment:

$$\mathcal{H}^*(\Gamma_1 \setminus z); \Theta \vdash \widehat{\mathcal{A}}^k_{\widetilde{y}}(P)_g \triangleright \diamond \tag{B.31}$$

We use some auxiliary sub-trees:

$$\text{RVar}\ \frac{}{\mathcal{H}^*(\Gamma)\cdot X : \Theta; \Theta \vdash X \triangleright \diamond} \tag{B.32}$$

$$\text{PolySend}\ \frac{(\text{B.32}) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma)\cdot X : \Theta; \mathcal{H}^*(\Delta, u_l : S', \Delta_z) \vdash \widetilde{y}'\sigma_1 \triangleright \widetilde{N}'}}{\mathcal{H}^*(\Gamma)\cdot X : \Theta; \Theta, \mathcal{H}^*(\Delta_1) \vdash c^r_{k+1}!\langle \widetilde{y}'\sigma_1\rangle.X \triangleright \diamond} \tag{B.33}$$

Here, in typing the right-hand assumption, we may notice that in sub-case (i) $\mathcal{H}^*(u_i : S) = \widetilde{u}_i : ?(\mathcal{H}^*(C_z));\mathtt{end}, \mathcal{H}^*(S')$ and $\mathcal{H}^*(u_i\sigma : S') = \widetilde{u}_{i+1} : \mathcal{H}^*(S')$ where $\widetilde{u}_{i+1} = (u_{i+1}, \ldots, u_{i+|\mathcal{H}^*(S')|})$ and by definition $\widetilde{u}_{i+1} \subseteq \widetilde{y}'\sigma_1$. So the right-hand side follows by Definition 5.3.9 and Lemma 2.2.1. Otherwise, in sub-case (ii) by Definition 5.3.2 and Figure 5.5 we know $\mathcal{H}^*(u_i : S) = \mathcal{H}^*(u_i\sigma : S')$.

$$\text{PolyRcv}\ \frac{(\text{B.33}) \qquad (\text{B.29})}{\mathcal{H}^*(\Gamma \setminus z)\cdot X : \Theta; \Theta, \mathcal{H}^*(\Delta, u_i : S) \vdash u_l?(\widetilde{z}).c^r_{k+1}!\langle \widetilde{y}'\sigma_1\rangle.X \triangleright \diamond} \tag{B.34}$$

$$\text{Rec}\ \frac{\text{PolyRcv}\ \dfrac{(\text{B.34}) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma \setminus z)\cdot X : \Theta; \mathcal{H}^*(\Delta, r : S) \vdash \widetilde{y} \triangleright \widetilde{N}}}{\mathcal{H}^*(\Gamma_1 \setminus z)\cdot X : \Theta; \Theta \vdash c^r_k?(\widetilde{y}).u_l?(\widetilde{z}).c^r_{k+1}!\langle \widetilde{y}'\sigma_1\rangle.X \triangleright \diamond}}{\mathcal{H}^*(\Gamma_1 \setminus z); \Theta \vdash \mu X.c^r_k?(\widetilde{y}).u_l?(\widetilde{z}).c^r_{k+1}!\langle \widetilde{y}'\sigma_1\rangle.X \triangleright \diamond} \tag{B.35}$$

We may notice that by Definition 5.3.2 and Figure 5.5 we have $\mathcal{H}^*(\Delta_1) = \mathcal{H}^*(\Delta), \widetilde{u}_i : \mathcal{H}^*(S), \mathcal{H}^*(\Delta_z)$ So, in sub-case (i) as $u_i = u_l$ we have $\mathcal{H}^*(\Delta_1)(u_l) = ?(\mathcal{H}^*(C_z));\mathtt{end}$. In sub-case (ii), by Lemma B.3.1 and as $u_l = u_{\iota(S)}$ we know $\mathcal{H}^*(\Delta_1)(u_l) = \mu\mathtt{t}.?(\mathcal{H}^*(C_z));\mathtt{t}$. The following tree proves this case:

$$\text{Par}\ \frac{(\text{B.35}) \qquad (\text{B.28})}{\mathcal{H}^*(\Gamma_1 \setminus z); \Theta \vdash \mu X.c^r_k?(\widetilde{y}).u_l?(\widetilde{z}).c^r_{k+1}!\langle \widetilde{y}'\sigma_1\rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}'\sigma_1}(P'\sigma)_g} \tag{B.36}$$

Note that we have used the following for the right assumption of (B.36):

$$\widehat{\mathcal{A}}^{k+1}_{\widetilde{y}'}(P') \equiv_\alpha \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}'\sigma_1}(P'\sigma)$$

4. Case $P = Q_1 \mid Q_2$. The only rule that can be applied here is Par:

$$\text{Par}\ \frac{\Gamma\cdot X : \Delta_\mu; \Delta_1 \vdash Q_1 \triangleright \diamond \qquad \Gamma\cdot X : \Delta_\mu; \Delta_2 \vdash Q_2 \triangleright \diamond}{\Gamma\cdot X : \Delta_\mu; \Delta_1, \Delta_2 \vdash Q_1 \mid Q_2 \triangleright \diamond} \tag{B.37}$$

Here we assume $\mathsf{frv}(Q_1)$. So, by IH on the first and second assumption of (B.37) we have:

$$\mathcal{H}^*(\Gamma \setminus x_1); \Theta_1 \vdash \widehat{\mathcal{A}}^{k}_{\widetilde{y}_1}(Q_1)_g \triangleright \diamond \tag{B.38}$$

$$\mathcal{H}^*(\Gamma \setminus x_1); \Theta_2 \vdash \widehat{\mathcal{A}}^{k+l+1}_{\widetilde{y}_2}(Q_2)_\emptyset \triangleright \diamond \tag{B.39}$$

where for $i \in \{1, 2\}$ we have $\widetilde{x}_i \subseteq \mathtt{fn}(Q_i)$ such that $\Delta_i \setminus \widetilde{x} = \emptyset$, and $\widetilde{y}_i = \widetilde{v}_i \cdot \widetilde{m}_i$, where $\widetilde{v}_i$ is such that $\mathtt{indexed}_{\Gamma,\Delta_i}(\widetilde{v}_1, \widetilde{x}_i)$ and $\widetilde{m}_i = \mathtt{codom}(g_i)$. Further, $\Theta_i = \Theta^i_\mu \cdot \Theta_X(g_i)$ where

$$\mathtt{dom}(\Theta^1_\mu) = \{c^r_{k+1}, c^r_{k+2}, \ldots, c^r_{k+\lfloor Q_1 \rceil^*}\} \cup \{\overline{c^r_{k+2}}, \ldots, \overline{c^r_{k+\lfloor Q_1 \rceil^*}}\}$$

$$\mathtt{dom}(\Theta^2_\mu) = \{c^r_{k+l+1}, c^r_{k+l+2}, \ldots, c^r_{k+l+\lfloor Q_2 \rceil^*}\}$$

and $\Theta^1_\mu(c^r_{k+1}) = \mu\mathsf{t}.?(\widetilde{N}^1);\mathsf{t}$ and $\Theta^2_\mu(c^r_{k+l+1}) = \langle \widetilde{N}^2\rangle$ with $\widetilde{N}^i = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta_\mu \cdot \Delta_i))(\widetilde{y}_i)$. Let $\Delta = \Delta_1 \cdot \Delta_2$. We define $\widetilde{x} = \widetilde{x}_1 \cdot \widetilde{x}_2$ and $\widetilde{y} = \widetilde{y}_1 \cdot \widetilde{y}_2$. By construction $\widetilde{x} \subseteq \mathtt{fn}(P)$ and $(\Delta_1 \cdot \Delta_2) \setminus \widetilde{x} = \emptyset$. Further, we may notice that $\widetilde{y} = \widetilde{v} \cdot \widetilde{m}$, where $\widetilde{m} = \mathtt{codom}(g)$ and $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{v}, \widetilde{x})$. We shall prove the following judgment:

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \Theta \vdash \widehat{\mathcal{A}}^k_{\widetilde{y}}(P)_g \triangleright \diamond \tag{B.40}$$

where $\Theta = \Theta_1 \cdot \Theta_2 \cdot \Theta'$ with

$$\Theta' = c^r_k : \mu\mathsf{t}.?(\widetilde{N});\mathsf{t} \cdot \overline{c^r_{k+1}} : \mu\mathsf{t}.!\langle \widetilde{N}_1\rangle;\mathsf{t}$$

By Table 5.4 we have:

$$\widehat{\mathcal{A}}^k_{\widetilde{y}}(P)_g = \mu X.c^r_k?(\widetilde{y}).(\overline{c^r_{k+1}}!\langle \widetilde{y}_1\rangle.X \mid c^r_{k+l+1}!\langle \widetilde{y}_2\rangle) \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}_1}(Q_1)_g \mid \widehat{\mathcal{A}}^{k+l+1}_{\widetilde{y}_2}(Q_2)_\emptyset$$

We use some auxiliary sub-trees:

$${\scriptstyle\text{Snd}}\,\dfrac{{\scriptstyle\text{Nil}}\,\dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta' \vdash \mathbf{0} \triangleright \diamond} \qquad {\scriptstyle\text{PolyVar}}\,\dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta'; \widetilde{y}_2 : \widetilde{N}_2 \vdash \widetilde{y}_2 \triangleright \widetilde{N}_2}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta'; \cdot\widetilde{y}_2 : \widetilde{N}_2 \vdash c^r_{k+l+1}!\langle \widetilde{y}_2\rangle} \tag{B.41}$$

$${\scriptstyle\text{Par}}\,\dfrac{{\scriptstyle\text{Snd}}\,\dfrac{{\scriptstyle\text{Rvar}}\,\dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta'; \Theta' \vdash X \triangleright \diamond} \qquad {\scriptstyle\text{PolyVar}}\,\dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta'; \widetilde{y}_1 : \widetilde{N}_1 \vdash \widetilde{y}_1 \triangleright \widetilde{N}}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta'; \Theta' \cdot \widetilde{y}_1 : \widetilde{N}_1 \vdash \overline{c^r_{k+1}}!\langle \widetilde{y}_1\rangle.X \triangleright \diamond} \qquad \text{(B.41)}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta'; \Theta' \cdot \widetilde{y} : \widetilde{N} \vdash \overline{c^r_{k+1}}!\langle \widetilde{y}_1\rangle.X \mid \overline{c^r_{k+l+1}}!\langle \widetilde{y}_2\rangle. \triangleright \diamond} \tag{B.42}$$

$${\scriptstyle\text{Rec}}\,\dfrac{{\scriptstyle\text{Rv}}\,\dfrac{\text{(B.42)} \qquad {\scriptstyle\text{PolyVar}}\,\dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta'; \widetilde{y} : \widetilde{N} \vdash \widetilde{y} \triangleright \widetilde{N}}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}) \cdot X : \Theta'; \Theta' \vdash c^r_k?(\widetilde{y}).(\overline{c^r_{k+1}}!\langle \widetilde{y}_1\rangle.X \mid c^r_{k+l+1}!\langle \widetilde{y}_2\rangle) \triangleright \diamond}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \Theta' \vdash \mu X.c^r_k?(\widetilde{y}).(\overline{c^r_{k+1}}!\langle \widetilde{y}_1\rangle.X \mid c^r_{k+l+1}!\langle \widetilde{y}_2\rangle) \triangleright \diamond} \tag{B.43}$$

The following tree proves this case:

$${\scriptstyle\text{Par}}\,\dfrac{\text{(B.43)} \qquad {\scriptstyle\text{Par}}\,\dfrac{\text{(B.38)} \qquad \text{(B.39)}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \Theta \vdash \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}_1}(Q_1)_{g_1} \mid \widehat{\mathcal{A}}^{k+l+1}_{\widetilde{y}_2}(Q_2)_{g_2} \triangleright \diamond}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \Theta_1 \cdot \Theta_2 \vdash \mu X.c^r_k?(\widetilde{y}).(\overline{c^r_{k+1}}!\langle \widetilde{y}_1\rangle.X \mid c^r_{k+l+1}!\langle \widetilde{y}_2\rangle) \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}_1}(Q_1)_g \mid \widehat{\mathcal{A}}^{k+l+1}_{\widetilde{y}_2}(Q_2)_\emptyset \triangleright \diamond}$$

5. Case $P = (\nu\, s : C)\, P'$. We distinguish two sub-cases: (i) $C = S$ and (ii) $C = \langle C'\rangle$. We only consider sub-case (i) as the other is similar. First, we α-convert P as follows:

$$P \equiv_\alpha (\nu\, s_1 : C)\, P'\{s_1\overline{s_1}/s\overline{s}\}$$

The only can that can be applied is ResS:

$${\scriptstyle\text{ResS}}\,\dfrac{\Gamma; \Delta_\mu \cdot \Delta \cdot s_1 : S \cdot \overline{s_1} : \overline{S} \vdash P'\{s_1\overline{s_1}/s\overline{s}\} \triangleright \diamond}{\Gamma; \Delta_\mu \cdot \Delta \vdash (\nu\, s_1 : S)\, P'\{s_1\overline{s_1}/s\overline{s}\} \triangleright \diamond} \tag{B.44}$$

By IH on the assumption of (B.44) we have:

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \Theta' \vdash \widehat{\mathcal{A}}^k_{\widetilde{y}'}(P'\{s_1\overline{s_1}/s\overline{s}\})_g \triangleright \diamond \tag{B.45}$$

where $\widetilde{x}' \subseteq \texttt{fn}(P)$ such that $(\Delta \cdot s_1 : S \cdot \overline{s_1} : \overline{S}) \setminus \widetilde{x}' = \emptyset$, and $\widetilde{y}' = \widetilde{v}' \cup \widetilde{m}$ where $\texttt{indexed}_{\Gamma,\Delta\cdot s_1:S\cdot\overline{s_1}:\overline{S}}(\widetilde{v}',\widetilde{x}')$. Also, $\Theta' = \Theta'_\mu, \Theta_X$, where Θ'_μ such that $\mathsf{balanced}(\Theta'_\mu)$ with

$$\texttt{dom}(\Theta'_\mu) = \{c^r_{k+1}, c^r_{k+2}, \dots, c^r_{k+\lfloor P'\rceil^*+1}\} \cup \{\overline{c^r_{k+2}}, \dots, \overline{c^r_{k+\lfloor P'\rceil^*+1}}\}$$

and $\Theta'_\mu(c^r_{k+1}) = \mu\mathsf{t}.?(\widetilde{N}');\mathsf{t}$ where $\widetilde{N}' = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta_\mu\cdot\Delta\cdot s_1:S\cdot\overline{s_1}:\overline{S}))(\widetilde{y}')$.

We define $\widetilde{x} = \widetilde{x}' \setminus (s_1, \overline{s_1})$ and $\widetilde{y} = \widetilde{y}' \setminus (\widetilde{s}\cdot\widetilde{\overline{s}})$ where $\widetilde{s} = \mathsf{bn}(s_1 : S)$ and $\widetilde{\overline{s}} = \mathsf{bn}(\overline{s_1} : \overline{S})$. By construction $\widetilde{x} \subseteq \texttt{fn}(P)$ and $\Delta \setminus \widetilde{x} = \emptyset$. Further, $\widetilde{y} = \widetilde{v}\cdot\widetilde{m}$ where $\texttt{indexed}_{\Gamma,\Delta}(\widetilde{v},\widetilde{x})$.

Let $\Theta = \Theta', \Theta''_\mu$ where

$$\Theta''_\mu = c^r_k : \mu\mathsf{t}.?(\widetilde{N});\mathsf{t}, \overline{c^r_{k+1}} : \mu\mathsf{t}.!\langle\widetilde{N}'\rangle;\mathsf{t}$$

where $\widetilde{N} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta_\mu\cdot\Delta\cdot u_i : S))(\widetilde{y})$. By construction and since $\lfloor P\rceil^* = \lfloor P'\rceil^* + 1$ we have

$$\texttt{dom}(\Theta_\mu) = \{c^r_k, c^r_{k+1}, \dots, c^r_{k+\lfloor P\rceil^*-1}\} \cup \{\overline{c^r_{k+1}}, \dots, \overline{c^r_{k+\lfloor P\rceil^*-1}}\}$$

By Table 5.4 we have:

$$\widehat{\mathcal{A}}^k_{\widetilde{y}}(P)_g = \mu X.(\nu\,\widetilde{s} : \mathcal{H}^*(C))\, c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.X \mid \widehat{\mathcal{A}}^k_{\widetilde{y}'}(P'\{^{s_1\overline{s_1}}/_{s\overline{s}}\})_g$$

We shall prove the following judgment:

$$\mathcal{H}^*(\Gamma\setminus\widetilde{x});\Theta \vdash \widehat{\mathcal{A}}^k_{\widetilde{y}}(P)_g \triangleright \diamond \tag{B.46}$$

We use the auxiliary sub-tree:

$$\text{PolyVar}\ \frac{}{\mathcal{H}^*(\Gamma)\cdot X : \Theta''_\mu; \mathcal{H}^*(\Delta), \widetilde{s} : \mathcal{H}^*(S), \widetilde{\overline{s}} : \mathcal{H}^*(\overline{S}) \vdash \widetilde{y}' \triangleright \widetilde{N}'} \tag{B.47}$$

$$\text{PolySend}\ \frac{\text{Rvar}\ \dfrac{}{\mathcal{H}^*(\Gamma)\cdot X : \Theta_\mu; \Theta''_\mu \vdash X \triangleright \diamond} \qquad (\text{B.47})}{\mathcal{H}^*(\Gamma)\cdot X : \Theta_\mu; \Theta_\mu\cdot\mathcal{H}^*(\Delta), \widetilde{s} : \mathcal{H}^*(S), \widetilde{\overline{s}} : \mathcal{H}^*(\overline{S}) \vdash \overline{c_{k+1}}!\langle\widetilde{y}'\rangle.X \triangleright \diamond} \tag{B.48}$$

The following tree proves this case:

$$\text{Par}\ \dfrac{\text{Rec}\ \dfrac{\text{PolyResS}\ \dfrac{\text{PolyRcv}\ \dfrac{(\text{B.48}) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma);\mathcal{H}^*(\Delta) \vdash \widetilde{y} \triangleright \widetilde{N}}}{\mathcal{H}^*(\Gamma\setminus\widetilde{x})\cdot X : \Theta''_\mu; \Theta''_\mu\cdot\widetilde{s} : \mathcal{H}^*(S), \widetilde{\overline{s}} : \mathcal{H}^*(\overline{S}) \vdash c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.X \triangleright \diamond}}{\mathcal{H}^*(\Gamma\setminus\widetilde{x})\cdot X : \Theta''_\mu; \Theta''_\mu \vdash (\nu\,\widetilde{s} : \mathcal{H}^*(S))\, c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.X \triangleright \diamond}}{\mathcal{H}^*(\Gamma\setminus\widetilde{x}); \Theta''_\mu \vdash \mu X.(\nu\,\widetilde{s} : \mathcal{H}^*(S))\, c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.X \triangleright \diamond} \qquad (\text{B.45})}{\mathcal{H}^*(\Gamma\setminus\widetilde{x}); \Theta \vdash \mu X.(\nu\,\widetilde{s} : \mathcal{H}^*(S))\, c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.X \mid \widehat{\mathcal{A}}^k_{\widetilde{y}'}(P'\{^{s_1\overline{s_1}}/_{s\overline{s}}\})_g \triangleright \diamond}$$

This concludes the proof.

□

B.3.3 Proof of Lemma 5.3.3

Lemma 5.3.3 (Typability of Breakdown). *Let P be an initialized process. If $\Gamma;\Delta \vdash P \triangleright \diamond$ then*

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x});\mathcal{H}^*(\Delta \setminus \widetilde{x}),\Theta \vdash \mathcal{A}^k_{\widetilde{y}}(P) \triangleright \diamond \quad (k > 0)$$

where

- $\widetilde{x} \subseteq \mathtt{fn}(P)$ *and* $\widetilde{y}$ *such that* $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{y},\widetilde{x})$ *holds.*
- *dom*$(\Theta) = \{c_k, c_{k+1}, \ldots, c_{k+\lfloor P \rceil^*-1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\lfloor P \rceil^*-1}}\}$
- $\Theta(c_k) = ?(\widetilde{M});\mathtt{end}$, *where* $\widetilde{M} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta))(\widetilde{y})$.
- $\mathsf{balanced}(\Theta)$

Proof. By induction on the structure of P. By assumption $\Gamma;\Delta \vdash P \triangleright \diamond$. In total we consider nine cases. We separately treat Input and Output cases depending on whether the subject name of the prefix is recursive or not.

1. Case $P = \mathbf{0}$. The only rule that can be applied here is Nil. By inversion of this rule, we have: $\Gamma;\emptyset \vdash \mathbf{0}$. We shall then prove the following judgment:

$$\mathcal{H}^*(\Gamma);\Theta \vdash \mathbf{A}^k_{\widetilde{y}}(\mathbf{0}) \triangleright \diamond \tag{B.49}$$

where $\widetilde{x} \subseteq \mathtt{fn}(\mathbf{0}) = \epsilon$ and $\Theta = \{c_k :?(\langle \mathtt{end} \rangle);\mathtt{end}\}$. Since by Remark 5 we know that $c_k?().\mathbf{0}$ stands for $c_k?(y).\mathbf{0}$ with $c_k :?(\langle \mathtt{end} \rangle);\mathtt{end}$. By Table 5.4: $\mathbf{A}^k_{\epsilon}(\mathbf{0}) = c_k?().\mathbf{0}$.

The following tree proves this case:

$$\dfrac{\dfrac{\dfrac{}{\Gamma';\emptyset;\emptyset \vdash \mathbf{0} \triangleright \diamond}\,\text{Nil} \qquad c_k \notin \mathtt{dom}(\Gamma)}{\Gamma';\emptyset;c_k : \mathtt{end} \vdash \mathbf{0} \triangleright \diamond}\,\text{End} \qquad \dfrac{}{\Gamma';\emptyset \vdash y \triangleright \langle \mathtt{end} \rangle}\,\text{Sh}}{\mathcal{H}^*(\Gamma), y : \langle \mathtt{end} \rangle;\Theta \vdash c_k?().\mathbf{0} \triangleright \diamond}\,\text{Rcv}$$

where $\Gamma' = \mathcal{H}^*(\Gamma), y : \langle \mathtt{end} \rangle$. We know $c_k \notin \mathtt{dom}(\Gamma)$ since we use reserved names for propagator channels.

2. Case $P = u_i?(z).P'$. We distinguish two sub-cases, depending on whether u_i is linear or not: (i) $u_i \in \mathtt{dom}(\Delta)$ and (ii) $u_i \in \mathtt{dom}(\Gamma)$. We consider sub-case (i) first. For this case Rule Rcv can be applied:

$$\text{Rcv}\ \dfrac{\Gamma;\Delta, u_i : S, \Delta_z \vdash P' \triangleright \diamond \qquad \Gamma;\Delta_z \vdash z \triangleright C}{\Gamma \setminus z;\Delta, u_i :?(C);S \vdash u_i?(z).P' \triangleright \diamond} \tag{B.50}$$

By IH on the first assumption of (B.50) we know:

$$\mathcal{H}^*(\Gamma_1');\mathcal{H}^*(\Delta_1'),\Theta_1 \vdash \mathcal{A}^{k+1}_{\widetilde{y}'}(P') \triangleright \diamond \tag{B.51}$$

where $\widetilde{x}' \subseteq \mathtt{fn}(P')$ and $\widetilde{y}'$ such that $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{y}',\widetilde{x}')$. Also, $\Gamma_1' = \Gamma \setminus \widetilde{x}'$, $\Delta_1' = \Delta \setminus \widetilde{x}'$, and $\mathsf{balanced}(\Theta_1)$ with

$$\mathtt{dom}(\Theta_1) = \{c_{k+1}, \ldots, c_{k+\lfloor P' \rceil^*}\} \cup \{\overline{c_{k+2}}, \ldots, \overline{c_{k+\lfloor P' \rceil^*}}\}$$

and $\Theta_1(c_{k+1}) = ?(\widetilde{M}');\mathtt{end}$ where $\widetilde{M}' = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta, u_i : S, \Delta_z))(\widetilde{y}')$.

By applying Lemma B.3.2 to the second assumption of (B.50) we have:

$$\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_z) \vdash \widetilde{z} \triangleright \mathcal{H}^*(C) \tag{B.52}$$

Let $\widetilde{x} = \widetilde{x}', u \setminus z$ and $\widetilde{y} = \widetilde{y}'\sigma, u_i \setminus \widetilde{z}$ such that $|\widetilde{z}| = \mathcal{H}^*(C)$, where $\sigma = \{\widetilde{n}/\widetilde{u}\}$ with $\widetilde{n} = (u_{i+1}, \ldots, u_{i+|\mathcal{H}^*(S)|})$ and $\widetilde{u} = (u_i, \ldots, u_{i+|\mathcal{H}^*(S)|-1})$. We may notice that by Definition 5.3.9 $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{y}, \widetilde{x})$ holds. We define $\Theta = \Theta_1, \Theta'$, where

$$\Theta' = c_k :?(\widetilde{M});\mathtt{end}, \overline{c_{k+1}} :!\langle \widetilde{M}' \rangle;\mathtt{end}$$

with $\widetilde{M} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta, , u_i :?(C);S))(\widetilde{y})$. By Definition 5.3.3, $\lfloor P \rceil^* = \lfloor P' \rceil^* + 1$ so

$$\mathtt{dom}(\Theta) = \{c_k, \ldots, c_{k+\lfloor P \rceil^* - 1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\lfloor P \rceil^* - 1}}\}$$

and Θ is balanced since $\Theta(c_{k+1})$ $\mathtt{dual}$ $\Theta(\overline{c_{k+1}})$ and Θ_1 is balanced. By Table 5.4:

$$\mathbf{A}^k_{\widetilde{y}}(u_i?(z).P') = c_k?(\widetilde{y}).u_i?(\widetilde{z}).\overline{c_{k+1}}!\langle \widetilde{y}'\sigma \rangle.\mathbf{0} \mid \mathcal{A}^{k+1}_{\widetilde{y}'\sigma}(P'\{u_{i+1}/u_i\})$$

Also, let $\Gamma_1 = \Gamma \setminus \widetilde{x}$ and $\Delta_1 = \Delta \setminus \widetilde{x}$. We may notice that $\Delta_1 = \Delta'_1$. We shall prove the following judgment:

$$\mathcal{H}^*(\Gamma_1 \setminus z); \mathcal{H}^*(\Delta_1), \Theta \vdash \mathcal{A}^k_{\widetilde{y}}(u_i?(z).P')$$

We type sub-process $c_k?(\widetilde{y}).u_i?(\widetilde{z}).\overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0}$ with some auxiliary derivations:

$$\text{End}\ \dfrac{\text{Nil}\ \dfrac{}{\mathcal{H}^*(\Gamma); \emptyset \vdash \mathbf{0} \triangleright \diamond}}{\mathcal{H}^*(\Gamma); \overline{c_{k+1}} : \mathtt{end} \vdash \mathbf{0} \triangleright \diamond} \tag{B.53}$$

$$\text{End}\ \dfrac{\text{PolySend}\ \dfrac{(\text{B.53}) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta \setminus \Delta_1, u_{i+1} : S), \mathcal{H}^*(\Delta_z) \vdash \widetilde{y}'\sigma \triangleright \widetilde{M}'}}{\begin{array}{c}\mathcal{H}^*(\Gamma); \overline{c_{k+1}} :!\langle \widetilde{M}' \rangle;\mathtt{end}, \mathcal{H}^*(\Delta \setminus \Delta_1, u_{i+1} : S), \mathcal{H}^*(\Delta_z) \vdash \\ \overline{c_{k+1}}!\langle \widetilde{y}'\sigma \rangle.\mathbf{0} \triangleright \diamond\end{array}}}{\begin{array}{c}\mathcal{H}^*(\Gamma); \overline{c_{k+1}} :!\langle \widetilde{M}' \rangle;\mathtt{end}, u_i : \mathtt{end}, \mathcal{H}^*(\Delta \setminus \Delta_1, u_{i+1} : S), \mathcal{H}^*(\Delta_z) \vdash \\ \overline{c_{k+1}}!\langle \widetilde{y}'\sigma \rangle.\mathbf{0} \triangleright \diamond\end{array}} \tag{B.54}$$

$$\text{End}\ \dfrac{\text{PolyRcv}\ \dfrac{(\text{B.54}) \qquad (\text{B.52})}{\begin{array}{c}\mathcal{H}^*(\Gamma \setminus z); u_i :?(\mathcal{H}^*(U));\mathtt{end}, \overline{c_{k+1}} :!\langle \widetilde{M}' \rangle;\mathtt{end}, \mathcal{H}^*(\Delta_2) \vdash \\ u_i?(\widetilde{z}).\overline{c_{k+1}}!\langle \widetilde{y}'\sigma \rangle.\mathbf{0} \triangleright \diamond\end{array}}}{\begin{array}{c}\mathcal{H}^*(\Gamma \setminus z); \overline{c_{k+1}} :!\langle \widetilde{M}' \rangle;\mathtt{end}, c_k : \mathtt{end}, \mathcal{H}^*(\Delta_2) \vdash \\ u_i?(\widetilde{z}).\overline{c_{k+1}}!\langle \widetilde{y}'\sigma \rangle.\mathbf{0} \triangleright \diamond\end{array}} \tag{B.55}$$

$$\text{PolyRcv}\ \dfrac{(\text{B.55}) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma \setminus z); \mathcal{H}^*(\Delta_2) \vdash \widetilde{y} \triangleright \widetilde{M}}}{\mathcal{H}^*(\Gamma_1 \setminus z); \Theta' \vdash c_k?(\widetilde{y}).u_i?(z).\overline{c_{k+1}}!\langle \widetilde{y}'\sigma \rangle.\mathbf{0} \triangleright \diamond} \tag{B.56}$$

where $\Delta_2 = \Delta, u_i :?(C);S \setminus \Delta_1$. Using (B.56), the following tree proves this case:

$$\text{Par}\ \dfrac{(\text{B.56}) \qquad \dfrac{(\text{B.51})}{\begin{array}{c}\mathcal{H}^*(\Gamma'_1 \setminus z); \mathcal{H}^*(\Delta_1), \Theta_1 \vdash \\ \mathcal{A}^{k+1}_{\widetilde{y}'\sigma}(P'\{u_{i+1}/u_i\}) \triangleright \diamond\end{array}}}{\begin{array}{c}\mathcal{H}^*(\Gamma_1 \setminus z); \mathcal{H}^*(\Delta_1), \Theta \vdash \\ c_k?(\widetilde{y}).u_i?(\widetilde{z}).\overline{c_{k+1}}!\langle \widetilde{y}'\sigma \rangle.\mathbf{0} \mid \mathcal{A}^{k+1}_{\widetilde{y}'\sigma}(P'\{u_{i+1}/u_i\}) \triangleright \diamond\end{array}} \tag{B.57}$$

Note that we have used the following for the right assumption of (B.57):

$$\mathcal{A}^{k+1}_{\widetilde{y}'}(P') \equiv_\alpha \mathcal{A}^{k+1}_{\widetilde{y}'\sigma}(P'\{{}^{u_{i+1}}/_{u_i}\})$$

Next, we comment the case when $u_i \notin \widetilde{x}$. In this case $\Delta_1 = \Delta \setminus \widetilde{x}, u_i : ?(C);S$. Hence, in the right hand-side of (B.57) instead of $\mathcal{H}^*(\Delta_1)$ we would have $\mathcal{H}^*(\Delta \setminus \widetilde{x}, u_{i+1} : S)$ and in the left-hand side we have $u_i : ?(\mathcal{H}^*(C));\mathtt{end}$ as a linear environment. Then, we would need to apply Lemma 2.2.1 with $\{{}^{u_i}/_{u_{i+1}}\}$ to the right-hand side before invoking (B.51). We remark that similar provisos apply to the following cases when the assumption is $u_i \notin \widetilde{x}$.

This concludes sub-case (i). We now consider sub-case (ii), i.e., $u_i \in \mathtt{dom}(\Gamma)$. Here Rule Acc can be applied:

$$\text{Acc}\ \frac{\Gamma;\emptyset \vdash u_i \triangleright \langle C\rangle \qquad \Gamma;\Delta, z : C \vdash P' \triangleright \diamond \qquad \Gamma; z : C \vdash z \triangleright C}{\Gamma;\Delta \vdash u_i?(z).P' \triangleright \diamond} \tag{B.58}$$

By IH on the second assumption of (B.58) we have:

$$\mathcal{H}^*(\Gamma'_1);\mathcal{H}^*(\Delta'_1),\Theta_1 \vdash \mathcal{A}^{k+1}_{\widetilde{y}'}(P') \triangleright \diamond \tag{B.59}$$

where $\widetilde{x}'$ and $\widetilde{y}'$ are as in sub-case (i). Also, $\Gamma'_1 = \Gamma \setminus \widetilde{x}'$ and $\Delta'_1 = \Delta \setminus \widetilde{y}'$ and $\mathsf{balanced}(\Theta_1)$ with

$$\mathtt{dom}(\Theta_1) = \{c_{k+1}, \ldots, c_{k+\lfloor P'\rceil^*}\} \cup \{\overline{c_{k+2}}, \ldots, \overline{c_{k+\lfloor P'\rceil^*}}\}$$

and $\Theta_1(c_{k+1}) = ?(\widetilde{M}');\mathtt{end}$ where $\widetilde{M}' = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta, z : C))(\widetilde{y}')$.

By applying Lemma B.3.2 to the first and third assumptions of (B.58) we have:

$$\mathcal{H}^*(\Gamma);\emptyset \vdash u_i \triangleright \langle \mathcal{H}^*(C)\rangle \tag{B.60}$$

$$\mathcal{H}^*(\Gamma);\mathcal{H}^*(z : C) \vdash \widetilde{z} \triangleright \mathcal{H}^*(C) \tag{B.61}$$

We define $\widetilde{x} = \widetilde{x}' \cup u \setminus z$ and $\widetilde{y} = \widetilde{y}' \cup u_i \setminus \widetilde{z}$ where $|\widetilde{z}| = |\mathcal{H}^*(C)|$. Notice that $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{y}, \widetilde{x})$. Let $\Gamma_1 = \Gamma \setminus x$. We define $\Theta = \Theta_1, \Theta'$, where

$$\Theta' = c_k : ?(\widetilde{M});\mathtt{end}, \overline{c_{k+1}} : !\langle \widetilde{M}'\rangle;\mathtt{end}$$

with $\widetilde{M} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta))(\widetilde{y})$. By Definition 5.3.3, $\lfloor P\rceil^* = \lfloor P'\rceil^* + 1$ so

$$\mathtt{dom}(\Theta) = \{c_k, \ldots, c_{k+\lfloor P\rceil^*-1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\lfloor P\rceil^*-1}}\}$$

and Θ is balanced since $\Theta(c_{k+1})$ dual $\Theta(\overline{c_{k+1}})$ and Θ_1 is balanced.

By Table 5.4, we have:

$$\mathbf{A}^k_{\widetilde{y}}(u_i?(z).P') = c_k?(\widetilde{y}).u_i?(z).\overline{c_{k+1}}!\langle \widetilde{y}'\rangle.\mathbf{0} \mid \mathcal{A}^{k+1}_{\widetilde{y}'}(P') \tag{B.62}$$

We shall prove the following judgment:

$$\mathcal{H}^*(\Gamma_1);\mathcal{H}^*(\Delta_1),\Theta \vdash \mathcal{A}^k_{\widetilde{y}}(u_i?(z).P') \triangleright \diamond \tag{B.63}$$

To this end, we use some auxiliary derivations:

$$\text{End}\ \frac{\text{Nil}\ \dfrac{}{\mathcal{H}^*(\Gamma);\emptyset;\emptyset \vdash \mathbf{0} \triangleright \diamond}}{\mathcal{H}^*(\Gamma);\emptyset;\overline{c_{k+1}} : \mathsf{end} \vdash \mathbf{0} \triangleright \diamond} \tag{B.64}$$

$$\text{PolySend}\ \dfrac{(\text{B.64}) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_2), z : \mathcal{H}^*(C) \vdash \widetilde{y}' \triangleright \widetilde{M}'}}{\mathcal{H}^*(\Gamma); \overline{c_{k+1}} :!\langle \widetilde{M}'\rangle;\mathtt{end}, \mathcal{H}^*(\Delta_2), z : \mathcal{H}^*(C) \vdash \overline{c_{k+1}}!\langle \widetilde{y}'\rangle.\mathbf{0} \triangleright \diamond} \tag{B.65}$$

$$\text{End}\ \dfrac{\text{PolyAcc}\ \dfrac{(\text{B.60}) \qquad (\text{B.65}) \qquad (\text{B.61})}{\mathcal{H}^*(\Gamma); \overline{c_{k+1}} :!\langle \widetilde{M}'\rangle;\mathtt{end}, \mathcal{H}^*(\Delta_2) \vdash u_i?(z).\overline{c_{k+1}}!\langle \widetilde{y}'\rangle.\mathbf{0} \triangleright \diamond}}{\mathcal{H}^*(\Gamma); \overline{c_{k+1}} :!\langle \widetilde{M}'\rangle;\mathtt{end}, c_k : \mathtt{end}, \mathcal{H}^*(\Delta_2) \vdash u_i?(z).\overline{c_{k+1}}!\langle \widetilde{y}'\rangle.\mathbf{0} \triangleright \diamond} \tag{B.66}$$

$$\text{PolyRcv}\ \dfrac{(\text{B.66}) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_2) \vdash \widetilde{y} \triangleright \widetilde{M}}}{\mathcal{H}^*(\Gamma_1); \Theta' \vdash c_k?(\widetilde{y}).u_i?(z).\overline{c_{k+1}}!\langle \widetilde{y}'\rangle.\mathbf{0} \triangleright \diamond} \tag{B.67}$$

where $\Delta_2 = \Delta \setminus \Delta_1$. Using (B.59) and (B.67), the following tree proves this sub-case:

$$\text{Par}\ \dfrac{(\text{B.67}) \qquad (\text{B.59})}{\mathcal{H}^*(\Gamma_1); \mathcal{H}^*(\Delta_1), \Theta \vdash c_k?(\widetilde{y}).u_i?(z).\overline{c_{k+1}}!\langle \widetilde{y}'\rangle.\mathbf{0} \mid \mathcal{A}^{k+1}_{\widetilde{y}'}(P') \triangleright \diamond} \tag{B.68}$$

Note that if $u \notin \mathtt{fn}(P')$ we need to apply Lemma 2.2.3 with u_i to the right assumption of (B.68) before applying (B.59). This concludes the analysis of the input case.

3. Case $P = u_i!\langle z_j\rangle.P'$. We distinguish two sub-cases: (i) $u_i \in \mathtt{dom}(\Delta)$ and (ii) $u_i \in \mathtt{dom}(\Gamma)$. We consider sub-case (i) first. For this case Rule Send can be applied:

$$\text{Send}\ \dfrac{\Gamma; \Delta_1, u_i : S \vdash P' \triangleright \diamond \qquad \Gamma; \Delta_z \vdash z_j \triangleright C \qquad u_i : S \in \Delta}{\Gamma; \Delta \vdash u_i!\langle z_j\rangle.P' \triangleright \diamond} \tag{B.69}$$

where $\Delta = \Delta_1, \Delta_z, u_i :!\langle C\rangle;S$.

By IH on the first assumption of (B.69) we have:

$$\mathcal{H}^*(\Gamma_1'); \mathcal{H}^*(\Delta_1'), \Theta_1 \vdash \mathcal{A}^{k+1}_{\widetilde{y}'}(P') \triangleright \diamond \tag{B.70}$$

where $\widetilde{x}' \subseteq \mathtt{fv}(P')$ and $\widetilde{y}'$ such that $\mathtt{indexed}_{\Gamma,\Delta_1,u_i:S}(\widetilde{y}', \widetilde{x}')$. Also, $\Gamma_1' = \Gamma \setminus \widetilde{y}'$, $\Delta_1' = \Delta_1 \setminus \widetilde{y}'$, and $\mathsf{balanced}(\Theta_1)$ with

$$\mathtt{dom}(\Theta_1) = \{c_{k+1}, \ldots, c_{k+\lfloor P'\rceil^*}\} \cup \{\overline{c_{k+2}}, \ldots, \overline{c_{k+\lfloor P'\rceil^*}}\}$$

and $\Theta_1(c_{k+1}) = ?(\widetilde{M}_1);\mathtt{end}$ where $\widetilde{M}_1 = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta_1, u_i : S))(\widetilde{y}')$.

By Lemma B.3.2 and the first assumption of (B.69) we have:

$$\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_z) \vdash \widetilde{z} \triangleright \mathcal{H}^*(C) \tag{B.71}$$

where $\widetilde{z} = (z_j, \ldots, z_{j+|\mathcal{H}^*(C)|-1})$. We assume $\widetilde{x} = \widetilde{x}', u, z$. Since $\widetilde{x}' \subseteq \mathtt{fn}(P')$ follows that $\widetilde{x} \subseteq \mathtt{fn}(P)$. Let $\widetilde{y} = \widetilde{y}'\sigma, u_i, \widetilde{z}$ where $\sigma = \{\widetilde{n}/\widetilde{u}\}$ with $\widetilde{n} = (u_{i+1}, \ldots, u_{i+|\mathcal{H}^*(S)|})$ and $\widetilde{u} = (u_i, \ldots, u_{i+|\mathcal{H}^*(S)|-1})$. We have $\widetilde{z} = \mathtt{fnb}(z, \widetilde{y})$. By Definition 5.3.9 follows that $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{y}, \widetilde{x})$. We define $\Theta = \Theta_1, \Theta'$, where:

$$\Theta' = c_k :?(\widetilde{M});\mathtt{end}, \overline{c_{k+1}} :!\langle \widetilde{M}_1\rangle;\mathtt{end}$$

with $\widetilde{M} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta))(\widetilde{y})$. By Definition 5.3.3, we know $\lfloor P\rceil^* = \lfloor P'\rceil^* + 1$, so

$$\mathtt{dom}(\Theta) = \{c_k, \ldots, c_{k+\lfloor P\rceil^*-1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\lfloor P\rceil^*-1}}\}$$

and Θ is balanced since $\Theta(c_{k+1})$ dual $\Theta(\overline{c_{k+1}})$ and Θ_1 is balanced.

By Table 5.4, we have:

$$\mathbf{A}^k_{\widetilde{y}}(u_i!\langle z\rangle.P') = c_k?(\widetilde{y}).u_i!\langle\widetilde{z}\rangle.\overline{c_{k+1}}!\langle\widetilde{y}'\sigma\rangle.\mathbf{0} \mid \mathcal{A}^{k+1}_{\widetilde{y}'\sigma}(P'\{u_{i+1}/u_i\})$$

Let $\Gamma_1 = \Gamma\setminus\widetilde{x} = \Gamma'_1$ and $\Delta\setminus\widetilde{x} = \Delta'_1$. We shall prove the following judgment:

$$\mathcal{H}^*(\Gamma_1);\mathcal{H}^*(\Delta'_1),\Theta \vdash \mathcal{A}^k_{\widetilde{y}}(u_i!\langle z\rangle.P') \triangleright\diamond \tag{B.72}$$

To type the sub-process $c_k?(\widetilde{y}).u_i!\langle\widetilde{z}\rangle.\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.\mathbf{0}$ of $\mathcal{A}^k_{\widetilde{y}}(u_i!\langle z\rangle.P')$ we use the following auxiliary derivations:

$$\text{End}\;\dfrac{\text{Nil}\;\dfrac{}{\mathcal{H}^*(\Gamma);\emptyset \vdash \mathbf{0}\triangleright\diamond}}{\mathcal{H}^*(\Gamma);\overline{c_{k+1}}:\mathsf{end}\vdash \mathbf{0}\triangleright\diamond} \tag{B.73}$$

$$\text{End}\;\dfrac{\text{PolySend}\;\dfrac{(\text{B.73}) \qquad \text{PolyVar}\;\dfrac{}{\mathcal{H}^*(\Gamma);\mathcal{H}^*(\Delta_1,u_{i+1}:S\setminus\Delta'_1)\vdash \widetilde{y}'\sigma\triangleright\widetilde{M}_1}}{\mathcal{H}^*(\Gamma);\overline{c_{k+1}}:!\langle\widetilde{M}_1\rangle;\mathsf{end},\mathcal{H}^*(\Delta_1,u_{i+1}:S\setminus\Delta'_1)\vdash\overline{c_{k+1}}!\langle\widetilde{y}'\sigma\rangle.\triangleright\diamond}}{\mathcal{H}^*(\Gamma);\overline{c_{k+1}}:!\langle\widetilde{M}_1\rangle;\mathsf{end},u_i:\mathsf{end},\mathcal{H}^*(\Delta_1,u_{i+1}:S\setminus\Delta'_1)\vdash\overline{c_{k+1}}!\langle\widetilde{y}'\sigma\rangle.\mathbf{0}\triangleright\diamond} \tag{B.74}$$

$$\text{End}\;\dfrac{\text{Send}\;\dfrac{(\text{B.74}) \qquad (\text{B.71})}{\mathcal{H}^*(\Gamma);\mathcal{H}^*(\Delta\setminus\Delta'_1),\overline{c_{k+1}}:!\langle\widetilde{M}_1\rangle;\mathsf{end}\vdash u_i!\langle\widetilde{z}\rangle.\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.\mathbf{0}\triangleright\diamond}}{\mathcal{H}^*(\Gamma);\mathcal{H}^*(\Delta\setminus\Delta'_1),\overline{c_{k+1}}:!\langle\widetilde{M}_1\rangle;\mathsf{end},c_k:\mathsf{end}\vdash u_i!\langle\widetilde{z}\rangle.\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.\mathbf{0}\triangleright\diamond} \tag{B.75}$$

$$\text{PolyRcv}\;\dfrac{(\text{B.75}) \qquad \text{PolyVar}\;\dfrac{}{\mathcal{H}^*(\Gamma);\mathcal{H}^*(\Delta\setminus\Delta'_1)\vdash\widetilde{y}:\widetilde{M}}}{\mathcal{H}^*(\Gamma_1);\Theta'\vdash c_k?(\widetilde{y}).u_i!\langle\widetilde{z}\rangle.\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.\mathbf{0}\triangleright\diamond} \tag{B.76}$$

Using (B.70) and (B.76), the following tree proves this case:

$$\text{Par}\;\dfrac{(\text{B.76}) \qquad \dfrac{(\text{B.70})}{\mathcal{H}^*(\Gamma_1);\mathcal{H}^*(\Delta'_1),\Theta_1\vdash\mathcal{A}^{k+1}_{\widetilde{y}'\sigma}(P'\{u_{i+1}/u_i\})\triangleright\diamond}}{\mathcal{H}^*(\Gamma_1);\mathcal{H}^*(\Delta'_1),\Theta\vdash\mathcal{A}^k_{\widetilde{y}}(u_i!\langle z\rangle.P')\triangleright\diamond} \tag{B.77}$$

Note that we have used the following for the right assumption of (B.57):

$$\mathcal{A}^{k+1}_{\widetilde{y}'}(P') \equiv_\alpha \mathcal{A}^{k+1}_{\widetilde{y}'\sigma}(P'\{u_{i+1}/u_i\})$$

We now consider sub-case (ii). For this sub-case Rule Req can be applied:

$$\text{Req}\;\dfrac{\Gamma;\emptyset\vdash u\triangleright\langle C\rangle \qquad \Gamma;\Delta_1\triangleright P'\triangleright\diamond \qquad \Gamma;\Delta_z\vdash z\triangleright C}{\Gamma;\Delta_1,\Delta_z\vdash u_i!\langle z\rangle.P'\triangleright\diamond} \tag{B.78}$$

Let $\widetilde{x}'\subseteq\mathtt{fn}(P')$ and $\widetilde{y}$ such that $\mathtt{indexed}_{\Gamma,\Delta_1}(\widetilde{y}',\widetilde{x}')$. Further, let $\Gamma'_1=\Gamma\setminus\widetilde{x}'$ and $\Delta'_1=\Delta_1\setminus\widetilde{x}'$. Also, let Θ_1 be environment defined as in sub-case (i).

By IH on the second assumption of (B.78) we have:

$$\mathcal{H}^*(\Gamma'_1);\mathcal{H}^*(\Delta'_1),\Theta_1\vdash\mathcal{A}^{k+1}_{\widetilde{y}'}(P')\triangleright\diamond \tag{B.79}$$

By Lemma B.3.2 and the first and third assumptions of (B.78) we have:

$$\mathcal{H}^*(\Gamma); \emptyset \vdash u_i \triangleright \mathcal{H}^*(\langle C \rangle) \tag{B.80}$$
$$\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_z) \vdash \widetilde{z} \triangleright \mathcal{H}^*(C) \tag{B.81}$$

We define $\widetilde{x} = \widetilde{x}' \cup z \cup u$ and $\widetilde{y} = \widetilde{y}' \cup \widetilde{z} \cup u_i$ where $|\widetilde{z}| = |\mathcal{H}^*(C)|$. Notice that $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{y}, \widetilde{x})$. Let $\Gamma_1 = \Gamma \setminus \widetilde{x} = \Gamma_1' \setminus u$ and $\Delta \setminus \widetilde{x} = \Delta_1'$.
By Table 5.4, we have:

$$\mathbf{A}^k_{\widetilde{y}}(u_i!\langle z \rangle.P') = c_k?(\widetilde{y}).u_i!\langle \widetilde{z} \rangle.\overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \mid \mathcal{A}^{k+1}_{\widetilde{y}'}(P')$$

We shall prove the following judgment:

$$\mathcal{H}^*(\Gamma_1); \mathcal{H}^*(\Delta_1'), \Theta \vdash \mathcal{A}^k_{\widetilde{y}}(u_i!\langle z \rangle.P')$$

We use some auxiliary derivations:

$$\dfrac{\dfrac{\dfrac{}{\mathcal{H}^*(\Gamma); \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond}\ \text{Nil}}{\mathcal{H}^*(\Gamma); \emptyset; \overline{c_{k+1}} : \mathsf{end} \vdash \mathbf{0} \triangleright \diamond}\ \text{End} \qquad \dfrac{}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_1 \setminus \Delta_1') \vdash \widetilde{y}' : \widetilde{M}_1}\ \text{PolyVar}}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_1 \setminus \Delta_1'), \overline{c_{k+1}} :!\langle \widetilde{M}_1 \rangle;\mathsf{end} \vdash \overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \triangleright \diamond}\ \text{PolySend} \tag{B.82}$$

$$\dfrac{(\text{B.80}) \qquad (\text{B.82}) \qquad (\text{B.81})}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_1 \setminus \Delta_1'), \overline{c_{k+1}} :!\langle \widetilde{M}_1 \rangle;\mathsf{end}, \widetilde{z} : \mathcal{H}^*(C) \vdash u_i!\langle \widetilde{z} \rangle.\overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \triangleright \diamond}\ \text{PolyReq} \tag{B.83}$$

$$\dfrac{(\text{B.83}) \qquad \dfrac{}{\mathcal{H}^*(\Gamma); \Delta_1 \setminus \Delta_1', \widetilde{z} : \mathcal{H}^*(C) \vdash \widetilde{y} \triangleright \widetilde{M}}\ \text{PolyVar}}{\mathcal{H}^*(\Gamma_1); \Theta' \vdash c_k?(\widetilde{y}).u_i!\langle \widetilde{z} \rangle.\overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \triangleright \diamond}\ \text{PolyRcv} \tag{B.84}$$

The following tree proves this case:

$$\dfrac{(\text{B.84}) \qquad \dfrac{(\text{B.79})}{\mathcal{H}^*(\Gamma_1); \mathcal{H}^*(\Delta_1'), \Theta_1 \vdash \mathcal{A}^{k+1}_{\widetilde{y}'}(P') \triangleright \diamond}}{\mathcal{H}^*(\Gamma_1); \mathcal{H}^*(\Delta_1'), \Theta \vdash \mathcal{A}^k_{\widetilde{y}}(u_i!\langle z \rangle.P') \triangleright \diamond}\ \text{Par} \tag{B.85}$$

We remark that if $u_i \notin \mathtt{fn}(P')$ we need to apply Lemma 2.2.3 with u_i to the right assumption of (B.85) before applying (B.79). This concludes the analysis for the output case $P = u_i!\langle z \rangle.P'$.

4. Case $P = (\nu\, s : C)\, P'$. We distinguish two sub-cases: (i) $C = S$ and (ii) $C = \langle C \rangle$. First, we α-convert P as follows:

$$P \equiv_\alpha (\nu\, s_1 : C)\, P'\{^{s_1 \overline{s_1}}/_{s \overline{s}}\}$$

We consider sub-case (i) first. For this case Rule ResS can be applied:

$$\dfrac{\Gamma; \Delta, s_1 : S, \overline{s_1} : \overline{S} \vdash P'\{^{s_1 \overline{s_1}}/_{s \overline{s}}\} \triangleright \diamond}{\Gamma; \Delta \vdash (\nu\, s_1 : S)\, P'\{^{s_1 \overline{s_1}}/_{s \overline{s}}\} \triangleright \diamond}\ \text{ResS} \tag{B.86}$$

By IH on the assumption of (B.86) we have:

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta \setminus \widetilde{x}, s_1 : S, \overline{s_1} : \overline{S}), \Theta_1 \vdash \mathcal{A}^k_{\widetilde{y}}(P'\{^{s_1 \overline{s_1}}/_{s \overline{s}}\}) \triangleright \diamond \tag{B.87}$$

where $\widetilde{x} \subseteq \texttt{fn}(P')$ such that $s_1, \overline{s_1} \notin \widetilde{x}$ and $\widetilde{y}$ such that $\texttt{indexed}_{\Gamma,\Delta}(\widetilde{y}, \widetilde{x})$. Also, $\mathsf{balanced}(\Theta_1)$ with

$$\texttt{dom}(\Theta_1) = \{c_k, \ldots, c_{k+\lfloor P' \rceil^* - 1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\lfloor P' \rceil^* - 1}}\}$$

and $\Theta_1(c_k) = ?(\widetilde{M});\text{end}$ with $\widetilde{M} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta))(\widetilde{y})$.
Note that we take $\Theta = \Theta_1$ since $\lfloor P \rceil^* = \lfloor P' \rceil^*$. By Definitions 5.3.1 and 5.3.2 and (B.87), we know that:

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta \setminus \widetilde{x}), \widetilde{s} : \mathcal{H}^*(S), \widetilde{\overline{s}} : \mathcal{H}^*(\overline{S}) \vdash \mathcal{A}^k_{\widetilde{y}}(P'\{^{s_1 \overline{s_1}}/_{s \overline{s}}\}) \triangleright \diamond \tag{B.88}$$

where $\widetilde{s} = (s_1, \ldots, s_{|\mathcal{H}^*(S)|})$ and $\widetilde{\overline{s}} = (\overline{s_1}, \ldots, \overline{s_{|\mathcal{H}^*(S)|}})$. By Table 5.4, we have:

$$\mathcal{A}^k_{\widetilde{y}}((\nu\, s)\, P') = (\nu\, \widetilde{s} : \mathcal{H}^*(S))\, \mathcal{A}^k_{\widetilde{y}}(P'\{^{s_1 \overline{s_1}}/_{s \overline{s}}\})$$

The following tree proves this sub-case:

$$\text{PolyResS}\ \frac{(\text{B.88})}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta \setminus \widetilde{x}) \vdash (\nu\, \widetilde{s} : \mathcal{H}^*(S))\, \mathcal{A}^k_{\widetilde{y}}(P'\{^{s_1 \overline{s_1}}/_{s \overline{s}}\}) \triangleright \diamond} \tag{B.89}$$

We now consider sub-case (ii). Similarly to sub-case (i) we first α-convert P as follows:

$$P \equiv_\alpha (\nu\, s_1)\, P'\{^{s_1}/_s\}$$

For this sub-case Rule Res can be applied:

$$\text{Res}\ \frac{\Gamma, s_1 : \langle C \rangle; \Delta \vdash P'\{^{s_1}/_s\} \triangleright \diamond}{\Gamma; \Delta \vdash (\nu\, s_1)\, P'\{^{s_1}/_s\} \triangleright \diamond} \tag{B.90}$$

By IH on the first assumption of (B.90) we have:

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}, s_1 : \langle C \rangle); \mathcal{H}^*(\Delta \setminus \widetilde{x}), \Theta_1 \vdash \mathcal{A}^k_{\widetilde{y}}(P'\{^{s_1}/_s\}) \triangleright \diamond \tag{B.91}$$

where $\widetilde{x} \subseteq \texttt{fn}(P')$ such that $s_1 \notin \widetilde{x}$ and $\widetilde{y}$ such that $\texttt{indexed}_{\Gamma,\Delta}(\widetilde{y}, \widetilde{x})$. Also, $\mathsf{balanced}(\Theta_1)$ with

$$\texttt{dom}(\Theta_1) = \{c_k, \ldots, c_{k+\lfloor P' \rceil^* - 1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\lfloor P' \rceil^* - 1}}\}$$

and $\Theta_1(c_k) = ?(\widetilde{M});\text{end}$ with $\widetilde{M} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta))(\widetilde{y})$.
Here we also take $\Theta = \Theta_1$ since $\lfloor P \rceil^* = \lfloor P' \rceil^*$. We notice that by Definitions 5.3.1 and 5.3.2 and (B.91):

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}), s_1 : \mathcal{H}^*(\langle C \rangle); \mathcal{H}^*(\Delta \setminus \widetilde{x}), \Theta_1 \vdash \mathcal{A}^k_{\widetilde{y}}(P'\{^{s_1}/_s\}) \triangleright \diamond \tag{B.92}$$

By Table 5.4, we have:

$$\mathcal{A}^k_{\widetilde{y}}((\nu\, s)\, P') = (\nu\, s_1 : \mathcal{H}^*(\langle C \rangle))\, \mathcal{A}^k_{\widetilde{y}}(P'\{^{s_1}/_s\})$$

The following tree proves this sub-case:

$$\text{PolyRes}\ \frac{(\text{B.92})}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta \setminus \widetilde{x}), \Theta \vdash (\nu\, s_1 : \mathcal{H}^*(\langle C \rangle))\, \mathcal{A}^k_{\widetilde{y}}(P'\{^{s_1}/_s\}) \triangleright \diamond} \tag{B.93}$$

5. Case $P = Q \mid R$. For this case only Rule Par can be applied:

$$\text{Par}\ \frac{\Gamma; \Delta_1 \vdash Q \triangleright \diamond \qquad \Gamma; \Delta_2 \vdash R \triangleright \diamond}{\Gamma; \Delta_1, \Delta_2 \vdash Q \mid R \triangleright \diamond} \tag{B.94}$$

By IH on the first assumption of (B.94) we have:

$$\mathcal{H}^*(\Gamma_1'); \mathcal{H}^*(\Delta_1'), \Theta_1 \vdash \mathcal{A}^{k+1}_{\widetilde{y}_1}(Q) \triangleright \diamond \tag{B.95}$$

where $\widetilde{x}_1 \subseteq \mathtt{fn}(Q)$ and $\widetilde{y}_1$ such that $\mathtt{indexed}_{\Gamma,\Delta_1}(\widetilde{y}_1, \widetilde{x}_1)$. Also, $\Gamma_1' = \Gamma \setminus \widetilde{x}_1$, $\Delta_1' = \Delta_1 \setminus \widetilde{x}_1$, and $\mathsf{balanced}(\Theta_1)$ with

$$\mathtt{dom}(\Theta_1) = \{c_{k+1}, \ldots, c_{k+\lfloor Q \rceil^*}\} \cup \{\overline{c_{k+2}}, \ldots, \overline{c_{k+\lfloor Q \rceil^*}}\}$$

and $\Theta_1(c_{k+1}) = ?(\widetilde{M}_1);\mathtt{end}$ with $\widetilde{M}_1 = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta_1))(\widetilde{y}_1)$.

By IH on the second assumption of (B.94) we have:

$$\mathcal{H}^*(\Gamma_2'); \mathcal{H}^*(\Delta_2'), \Theta_2 \vdash \mathcal{A}^{k+l+1}_{\widetilde{y}_2}(R) \triangleright \diamond \tag{B.96}$$

where $\widetilde{x}_2 \subseteq \mathtt{fn}(R)$ and $\widetilde{y}_2$ such that $\mathtt{indexed}_{\Gamma,\Delta_2}(\widetilde{y}_2, \widetilde{x}_2)$ and $l = |Q|$. Also, $\Gamma_2' = \Gamma \setminus \widetilde{x}_2$, $\Delta_2' = \Delta_2 \setminus \widetilde{x}_2$ and $\mathsf{balanced}(\Theta_2)$ with

$$\mathtt{dom}(\Theta_2) = \{c_{k+l+1}, \ldots, c_{k+l+\lfloor R \rceil^*}\} \cup \{\overline{c_{k+l+2}}, \ldots, \overline{c_{k+l+\lfloor R \rceil^*}}\}$$

and $\Theta_2(c_{k+l+1}) = ?(\widetilde{M}_2);\mathtt{end}$ with $\widetilde{M}_2 = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta_2))(\widetilde{y}_2)$.

We define $\widetilde{x} = \widetilde{x}_1 \cup \widetilde{x}_2$. We may notice that $\widetilde{x} \subseteq \mathtt{fn}(P)$ since $\mathtt{fn}(P) = \mathtt{fn}(Q) \cup \mathtt{fn}(R)$. Accordingly, we define $\widetilde{y} = \widetilde{y}_1, \widetilde{y}_2$. By definition, $\mathtt{indexed}_{\Gamma,\Delta_1,\Delta_2}(\widetilde{y}, \widetilde{x})$ holds. Further, let $\widetilde{M} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta_1, \Delta_2))(\widetilde{y})$. We define $\Theta = \Theta_1, \Theta_2, \Theta'$ where:

$$\Theta' = c_k :?(\widetilde{M});\mathtt{end}, \overline{c_{k+1}} :!\langle \widetilde{M}_1 \rangle;\mathtt{end}, \overline{c_{k+l+1}} :!\langle \widetilde{M}_2 \rangle;\mathtt{end}$$

By construction Θ is balanced since $\Theta(c_{k+1})$ dual $\Theta(\overline{c_{k+1}})$, $\Theta(c_{k+l+1})$ dual $\Theta(\overline{c_{k+l+1}})$, and Θ_1 and Θ_2 are balanced.

By Table 5.4 we have:

$$\mathcal{A}^k_{\widetilde{y}}(Q \mid R) = c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle \widetilde{y}_1 \rangle.\overline{c_{k+l+1}}!\langle \widetilde{y}_2 \rangle. \mid \mathcal{A}^{k+1}_{\widetilde{y}_1}(Q) \mid \mathcal{A}^{k+l+1}_{\widetilde{y}_2}(R)$$

We may notice that $\widetilde{y}_1 = \mathtt{fnb}(Q, \widetilde{y})$ and $\widetilde{y}_2 = \mathtt{fnb}(R, \widetilde{y})$ hold by the construction of $\widetilde{y}$. Let $\Gamma_1 = \Gamma \setminus \widetilde{x}$. We shall prove the following judgment:

$$\mathcal{H}^*(\Gamma_1); \mathcal{H}^*(\Delta_1', \Delta_2'), \Theta \vdash c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle \widetilde{y}_1 \rangle.\overline{c_{k+l+1}}!\langle \widetilde{y}_2 \rangle. \mid \mathcal{A}^{k+1}_{\widetilde{y}_1}(Q) \mid \mathcal{A}^{k+l+1}_{\widetilde{y}_2}(R) \triangleright \diamond$$

To type $c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle \widetilde{y}_1 \rangle.\overline{c_{k+l+1}}!\langle \widetilde{y}_2 \rangle.\mathbf{0}$, we use some auxiliary derivations:

$$\text{End}\ \dfrac{\text{PolySend}\ \dfrac{\text{End}\ \dfrac{\text{Nil}\ \dfrac{}{\mathcal{H}^*(\Gamma); \emptyset \vdash \mathbf{0}}}{\mathcal{H}^*(\Gamma); \overline{c_{k+l+1}} : \mathtt{end} \vdash \mathbf{0}} \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_2), \vdash \widetilde{z} \triangleright \widetilde{M}_2}}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_2'), \overline{c_{k+l+1}} :!\langle \widetilde{M}_2 \rangle;\mathtt{end} \vdash \overline{c_{k+l+1}}!\langle \widetilde{z} \rangle.\mathbf{0} \triangleright \diamond}}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_2'), \overline{c_{k+l+1}} :!\langle \widetilde{M}_2 \rangle;\mathtt{end}, \overline{c_{k+1}} : \mathtt{end} \vdash \overline{c_{k+l+1}}!\langle \widetilde{z} \rangle.\mathbf{0} \triangleright \diamond} \tag{B.97}$$

$$\text{End}\ \dfrac{\text{PolySend}\ \dfrac{(B.97) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_1) \vdash \widetilde{y}_1 \triangleright \widetilde{M}_1}}{\begin{array}{c}\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_1', \Delta_2'), \overline{c_{k+1}} :!\langle \widetilde{M}_1 \rangle;\mathtt{end}, \overline{c_{k+l+1}} :!\langle \widetilde{M}_2 \rangle;\mathtt{end} \vdash \\ \overline{c_{k+1}}!\langle \widetilde{y}_1 \rangle.\overline{c_{k+l+1}}!\langle \widetilde{y}_2 \rangle.\mathbf{0} \triangleright \diamond\end{array}}}{\begin{array}{c}\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_1', \Delta_2'), \overline{c_{k+1}} :!\langle \widetilde{M}_1 \rangle;\mathtt{end}, \overline{c_{k+l+1}} :!\langle \widetilde{M}_2 \rangle;\mathtt{end}, c_k : \mathtt{end} \vdash \\ \overline{c_{k+1}}!\langle \widetilde{y}_1 \rangle.\overline{c_{k+l+1}}!\langle \widetilde{y}_2 \rangle.\mathbf{0} \triangleright \diamond\end{array}} \tag{B.98}$$

$$\text{PolyRcv}\ \dfrac{(B.98) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_1', \Delta_2') \vdash \widetilde{x} \triangleright \widetilde{M}}}{\mathcal{H}^*(\Gamma_1); \Theta' \vdash c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle \widetilde{y}_1 \rangle.\overline{c_{k+l+1}}!\langle \widetilde{y}_2 \rangle.\mathbf{0} \triangleright \diamond} \tag{B.99}$$

$$\text{(Lemma 2.2.2) with } \Gamma_1' \setminus \Gamma_1\ \dfrac{(B.95)}{\mathcal{H}^*(\Gamma_1); \mathcal{H}^*(\Delta_1), \Theta_1 \vdash \mathcal{A}^{k+1}_{\tilde{y}_1}(Q) \triangleright \diamond} \tag{B.100}$$

$$\text{(Lemma 2.2.2) with } \Gamma_2' \setminus \Gamma_1\ \dfrac{(B.96)}{\mathcal{H}^*(\Gamma_1); \mathcal{H}^*(\Delta_2'), \Theta_2 \vdash \mathcal{A}^{k+l+1}_{\tilde{y}_2}(R) \triangleright \diamond} \tag{B.101}$$

$$\text{Par}\ \dfrac{(B.100) \qquad (B.101)}{\mathcal{H}^*(\Gamma_1); \mathcal{H}^*(\Delta_1', \Delta_2'), \Theta_1, \Theta_2 \vdash \mathcal{A}^{k+1}_{\tilde{y}}(Q) \mid \mathcal{A}^{k+l+1}_{\tilde{z}}(R) \triangleright \diamond} \tag{B.102}$$

The following tree proves this case:

$$\text{Par}\ \dfrac{(B.99) \qquad (B.102)}{\mathcal{H}^*(\Gamma_1); \mathcal{H}^*(\Delta_1', \Delta_2'), \Theta \vdash c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle \widetilde{y}_1 \rangle.\overline{c_{k+l+1}}!\langle \widetilde{y}_2 \rangle. \mid \mathcal{A}^{k+1}_{\tilde{y}_1}(Q) \mid \mathcal{A}^{k+l+1}_{\tilde{y}_2}(R) \triangleright \diamond}$$

6. Case $P = \mu X.P'$. The only rule that can be applied here is Rec:

$$\dfrac{\Gamma, X : \Delta; \Delta \vdash P' \triangleright \diamond}{\Gamma; \Delta \vdash \mu X.P'} \tag{B.103}$$

We remark that by (B.103) we know $x : S \in \Delta \implies \mathsf{tr}(S)$. Then, by applying Lemma 5.3.2 on the assumption of (B.103) we have:

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \Theta \vdash \widehat{\mathcal{A}}^{k+1}_{\tilde{y}'}(P')_g \triangleright \diamond \tag{B.104}$$

We take $\widetilde{x}' = \mathtt{fn}(P')$, so we have $\Delta \setminus x' = \emptyset$. Further, $\widetilde{y}'$ is such that $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{y}', \widetilde{x}')$. Let $\Theta' = \Theta_\mu', \Theta_X(g)$ with $\Theta_X(g) = c^r_X : \langle \widetilde{M}' \rangle$ with $\widetilde{M}' = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta))(\widetilde{y}')$. Further, $\widetilde{x}' \subseteq P'$ and $\widetilde{y}'$ such that $\Delta \setminus x' = \emptyset$ and $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{y}', \widetilde{x}')$. Also, $\mathsf{balanced}(\Theta_\mu')$ with

$$\mathtt{dom}(\Theta_\mu') = \{c^r_{k+1}, \ldots, c^r_{k+\lfloor P' \rceil^*}\} \cup \{\overline{c^r_{k+1}}, \ldots, \overline{c^r_{k+\lfloor P' \rceil^*}}\}$$

where $\Theta_\mu'(c_{k+1}) = c_k :?(\widetilde{M}');\mathtt{end}$.

Let $l = \lfloor P' \rceil^*$ and $\widetilde{z}$ such that $|\widetilde{y}'| = |\widetilde{z}|$. Further, let $\widetilde{x} \subseteq \mathtt{fn}(P)$ and $\widetilde{y} = \mathsf{bn}(\widetilde{x} : \widetilde{S})$. By definition we have $\widetilde{x} \subseteq \widetilde{x}'$. By Table 5.4, we have:

$$\mathcal{A}^k_{\tilde{y}}(P) = (\nu\, c^r_X)\, (c_k?(\widetilde{y}).\overline{c^r_{k+1}}!\langle \widetilde{y}' \rangle.\mu X.c^r_X?(\widetilde{z}).\overline{c^r_{k+1}}!\langle \widetilde{z} \rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\tilde{y}'}(P')_g)$$

Let $\Theta_\mu = \Theta'_\mu \cdot \Theta''_\mu$ where:

$$\Theta''_\mu = \overline{c^r_k} : \mu\mathsf{t}.?(\widetilde{M});\mathsf{t} \cdot \overline{c^r_{k+1}} : \mu\mathsf{t}.!\langle \widetilde{M}' \rangle;\mathsf{t}$$

We shall prove the following judgment:

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta \setminus \widetilde{x}) \cdot \Delta_\mu \vdash \mathcal{A}^k_{\widetilde{y}}(P) \triangleright \diamond \tag{B.105}$$

Let $\Theta'_X = c^r_X : \mu\mathsf{t}.?(\widetilde{M}');\mathsf{t} \cdot \overline{c^r_X} : \mu\mathsf{t}.!\langle \widetilde{M}' \rangle;\mathsf{t}$. We use some auxiliary sub-trees:

$$\dfrac{\dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}), X : \Theta''; \Theta'' \vdash X \triangleright \diamond}\,\text{RVar} \qquad \dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}), X : \Theta''; \mathcal{H}^*(\Delta) \vdash \widetilde{z} \triangleright \widetilde{M}}\,\text{PolyVar}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}), X : \Theta''; \mathcal{H}^*(\Delta), \Theta'' \vdash \overline{c^r_{k+1}}!\langle \widetilde{z} \rangle.X \triangleright \diamond}\,\text{Send} \tag{B.106}$$

$$\dfrac{\dfrac{(\text{B.106}) \qquad \dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta) \vdash \widetilde{z} \triangleright \widetilde{M}}\,\text{PolyVar}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}), X : \Theta''; \Theta'' \vdash c^r_X?(\widetilde{z}).\overline{c^r_{k+1}}!\langle \widetilde{z} \rangle.X \triangleright \diamond}\,\text{PolyRcv}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \Theta'' \vdash \mu X.c^r_X?(\widetilde{z}).\overline{c^r_{k+1}}!\langle \widetilde{z} \rangle.X \triangleright \diamond}\,\text{Rec} \tag{B.107}$$

where $\Theta'' = \overline{c^r_{k+1}} : \mu\mathsf{t}.!\langle \widetilde{M} \rangle;\mathsf{t}, c^r_X : \mu\mathsf{t}.?(\widetilde{M});\mathsf{t}$.

$$\dfrac{(\text{B.107}) \qquad \dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta) \vdash \widetilde{y}' \triangleright \widetilde{M}'}\,\text{PolyVar}}{\begin{array}{c}\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta), \overline{c^r_{k+1}} : \mu\mathsf{t}.!\langle \widetilde{M} \rangle;\mathsf{t}, c^r_X : \mu\mathsf{t}.?(\widetilde{M});\mathsf{t} \vdash \\ \overline{c^r_{k+1}}!\langle \widetilde{y} \rangle.\mu X.c^r_X?(\widetilde{z}).\overline{c^r_{k+1}}!\langle \widetilde{z} \rangle.X \triangleright \diamond\end{array}}\,\text{PolySend} \tag{B.108}$$

$$\dfrac{(\text{B.108}) \qquad \dfrac{}{\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \widetilde{y} : \widetilde{M} \vdash \widetilde{y} \triangleright \widetilde{M}}\,\text{PolyVar}}{\begin{array}{c}\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta \setminus \widetilde{x}) \cdot \Theta''_\mu \cdot c^r_X : \mu\mathsf{t}.?(\widetilde{M});\mathsf{t} \vdash \\ c_k?(\widetilde{y}).\overline{c^r_{k+1}}!\langle \widetilde{y}' \rangle.\mu X.c^r_X?(\widetilde{z}).\overline{c^r_{k+1}}!\langle \widetilde{z} \rangle.X \triangleright \diamond\end{array}}\,\text{PolyRcv} \tag{B.109}$$

The following tree proves this case:

$$\dfrac{\dfrac{(\text{B.109}) \qquad (\text{B.104})}{\begin{array}{c}\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta \setminus \widetilde{x}) \cdot \Theta_\mu \cdot \Theta'_X \vdash \\ c_k?(\widetilde{y}).\overline{c^r_{k+1}}!\langle \widetilde{y}' \rangle.\mu X.c^r_X?(\widetilde{z}).\overline{c^r_{k+1}}!\langle \widetilde{z} \rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}}(P') \triangleright \diamond\end{array}}\,\text{Par}}{\begin{array}{c}\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta \setminus \widetilde{x}) \cdot \Theta_\mu \vdash \\ (\nu\, c^r_X)\,(c_k?(\widetilde{y}).\overline{c^r_{k+1}}!\langle \widetilde{y}' \rangle.\mu X.c^r_X?(\widetilde{z}).\overline{c^r_{k+1}}!\langle \widetilde{z} \rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{y}'}(P')) \triangleright \diamond\end{array}}\,\text{ResS} \tag{B.110}$$

7. Case $P = r!\langle z \rangle.P'$ when $\mathsf{tr}(r)$. For this case Rule Send can be applied:

$$\dfrac{\Gamma; \Delta, r : S' \vdash P' \triangleright \diamond \qquad \Gamma; \Delta_z \vdash z \triangleright C}{\Gamma; \Delta, r : S, \Delta_z \vdash r!\langle z \rangle.P' \triangleright \diamond}\,\text{Send} \tag{B.111}$$

where $S = !\langle C \rangle;S'$.

Then, by IH on the first assumption of (B.111) we have:

$$\mathcal{H}^*(\Gamma'_1); \mathcal{H}^*(\Delta'_1), \Theta_1 \vdash \mathcal{A}^{k+1}_{\widetilde{y}'}(P') \triangleright \diamond \tag{B.112}$$

where $\widetilde{x}' \subseteq \mathtt{fn}(P')$ such that $r \in \widetilde{x}'$ and $\widetilde{y}'$ such that $\mathtt{indexed}_{\Gamma,\Delta,r:S}(\widetilde{y}', \widetilde{x}')$. By this follows that $(r_1, \ldots, r_{|\mathcal{H}^*(S)|}) \subseteq \widetilde{y}'$. Also, $\Gamma_1' = \Gamma \setminus \widetilde{x}'$, $\Delta_1' = \Delta \setminus \widetilde{y}'$, and $\mathsf{balanced}(\Theta_1)$ with

$$\mathtt{dom}(\Theta_1) = \{c_{k+1}, \ldots, c_{k+\lfloor P' \rceil^*}\} \cup \{\overline{c_{k+2}}, \ldots, \overline{c_{k+\lfloor P' \rceil^*}}\}$$

and $\Theta_1(c_{k+1}) = ?(\widetilde{M}_1);\mathtt{end}$ where $\widetilde{M}_1 = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta, r : S))(\widetilde{y}')$.

By applying Lemma B.3.2 on the assumption of (B.69) we have:

$$\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_z) \vdash \widetilde{z} \triangleright \mathcal{H}^*(C) \tag{B.113}$$

We assume $\widetilde{x} = \widetilde{x}', z$. Since $\widetilde{x}' \subseteq \mathtt{fn}(P')$ follows that $\widetilde{x} \subseteq \mathtt{fn}(P)$. Let $\widetilde{y} = \widetilde{y}', \widetilde{z}$ where $|\widetilde{z}| = |\mathcal{H}^*(C)|$. By Definition 5.3.9 it follows that $\mathtt{indexed}_{\Gamma,\Delta,r:S,\Delta_z}(\widetilde{y}, \widetilde{x})$. We define $\Theta = \Theta_1, \Theta'$, where:

$$\Theta' = c_k :?(\widetilde{M});\mathtt{end}, \overline{c_{k+1}} :!\langle \widetilde{M}_1 \rangle;\mathtt{end}$$

with $\widetilde{M} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta, r : S, \Delta_z))(\widetilde{y})$. By Definition 5.3.3, we know $\lfloor P \rceil^* = \lfloor P' \rceil^* + 1$, so

$$\mathtt{dom}(\Theta) = \{c_k, \ldots, c_{k+\lfloor P \rceil^*-1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\lfloor P \rceil^*-1}}\}$$

and Θ is balanced since $\Theta(c_{k+1})$ $\mathtt{dual}$ $\Theta(\overline{c_{k+1}})$ and Θ_1 is balanced.

By Table 5.4, we have:

$$\mathbf{A}^k_{\widetilde{y}}(r!\langle z \rangle.P') = c_k?(\widetilde{y}).r_{\iota(S)}!\langle \widetilde{z} \rangle.\overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \mid \mathcal{A}^{k+1}_{\widetilde{y}'}(P')$$

Let $\Gamma_1 = \Gamma \setminus \widetilde{x} = \Gamma_1'$ and $\Delta \setminus \widetilde{x} = \Delta_1'$. We shall prove the following judgment:

$$\mathcal{H}^*(\Gamma_1); \mathcal{H}^*(\Delta_1'), \Theta \vdash \mathcal{A}^k_{\widetilde{y}}(r!\langle z \rangle.P') \triangleright \diamond \tag{B.114}$$

Let $\Delta_1 = \Delta, r : S, \Delta_z$. To type the left-hand side component of $\mathcal{A}^k_{\widetilde{y}}(r!\langle z \rangle.P')$ we use some auxiliary derivations:

$$\dfrac{\text{End}\;\dfrac{\text{Nil}\;\dfrac{}{\mathcal{H}^*(\Gamma); \emptyset \vdash \mathbf{0} \triangleright \diamond}}{\mathcal{H}^*(\Gamma); \overline{c_{k+1}} : \mathtt{end} \vdash \mathbf{0} \triangleright \diamond} \qquad \text{PolyVar}\;\dfrac{}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta, r : S \setminus \Delta_1') \vdash \widetilde{y}' \triangleright \widetilde{M}_1}}{\mathcal{H}^*(\Gamma); \overline{c_{k+1}} :!\langle \widetilde{M}_1 \rangle;\mathtt{end}, \mathcal{H}^*(\Delta, r : S \setminus \Delta_1') \vdash \overline{c_{k+1}}!\langle \widetilde{y}' \rangle. \triangleright \diamond}\;\text{PolySend} \tag{B.115}$$

$$\text{End}\;\dfrac{\text{PolySend}\;\dfrac{(\text{B.115}) \qquad (\text{B.113})}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_1 \setminus \Delta_1'), \overline{c_{k+1}} :!\langle \widetilde{M}_1 \rangle;\mathtt{end} \vdash r_{\iota(S)}!\langle \widetilde{z} \rangle.\overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \triangleright \diamond}}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_1 \setminus \Delta_1'), \overline{c_{k+1}} :!\langle \widetilde{M}_1 \rangle;\mathtt{end}, c_k : \mathtt{end} \vdash r_{\iota(S)}!\langle \widetilde{z} \rangle.\overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \triangleright \diamond} \tag{B.116}$$

$$\text{PolyRcv}\;\dfrac{(\text{B.116}) \qquad \text{PolyVar}\;\dfrac{}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_1 \setminus \Delta_1') \vdash \widetilde{y} \triangleright \widetilde{M}}}{\mathcal{H}^*(\Gamma_1); \Theta' \vdash c_k?(\widetilde{y}).r_{\iota(S)}!\langle \widetilde{z} \rangle.\overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \triangleright \diamond} \tag{B.117}$$

We may notice that by Definition 5.3.2 and Figure 5.5 we have $\mathcal{H}^*(\Delta_1 \setminus \Delta_1') = \mathcal{H}^*(\Delta), \widetilde{r} : \mathcal{R}^*(S), \mathcal{H}^*(\Delta_z) \setminus \mathcal{H}^*(\Delta_1')$. Further, by Lemma B.3.1 we know $\mathcal{H}^*(\Delta_1)(r_{\iota(S)}) = \mu\mathsf{t}.!\langle \mathcal{H}^*(C) \rangle;\mathsf{t}$.

The following tree proves this case:

$$\text{Par}\ \dfrac{(\text{B.117}) \qquad \dfrac{(\text{B.112})}{\mathcal{H}^*(\Gamma_1); \mathcal{H}^*(\Delta_1'), \Theta_1 \vdash \mathcal{A}^{k+1}_{\widetilde{y}'}(P') \triangleright \diamond}}{\mathcal{H}^*(\Gamma_1); \mathcal{H}^*(\Delta_1'), \Theta \vdash \mathcal{A}^{k}_{\widetilde{y}}(r!\langle z\rangle.P') \triangleright \diamond} \qquad (\text{B.118})$$

8. Case $P = r?(z).P'$ when $\mathsf{tr}(r)$. For this case Rule Rcv can be applied:

$$\text{Rcv}\ \dfrac{\Gamma; \Delta, r : S', \Delta_z \vdash P' \triangleright \diamond \qquad \Gamma; \Delta_z \vdash z \triangleright C}{\Gamma \setminus z; \Delta, r : S \vdash r?(z).P' \triangleright \diamond} \qquad (\text{B.119})$$

where $S = ?(C);S'$.

Then, by IH on the first assumption of (B.50) we know:

$$\mathcal{H}^*(\Gamma_1'); \mathcal{H}^*(\Delta_1'), \Theta_1 \vdash \mathcal{A}^{k+1}_{\widetilde{y}'}(P') \triangleright \diamond \qquad (\text{B.120})$$

where $\widetilde{x}' \subseteq \mathtt{fn}(P')$ such that $r \in \widetilde{x}'$ and $\widetilde{y}'$ such that $\mathtt{indexed}_{\Gamma,\Delta,r:S}(\widetilde{y}', \widetilde{x}')$. By this it follows that $(r_1, \dots, r_{|\mathcal{H}^*(S)|}) \subseteq \widetilde{y}'$.

Also, $\Gamma_1' = \Gamma \setminus \widetilde{x}'$, $\Delta_1' = \Delta \setminus \widetilde{x}'$, and $\mathsf{balanced}(\Theta_1)$ with

$$\mathtt{dom}(\Theta_1) = \{c_{k+1}, \dots, c_{k+\lfloor P' \rceil^*}\} \cup \{\overline{c_{k+2}}, \dots, \overline{c_{k+\lfloor P' \rceil^*}}\}$$

and $\Theta_1(c_{k+1}) = ?(\widetilde{M}');\mathsf{end}$ where $\widetilde{M}' = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta, r : S, \Delta_z))(\widetilde{y}')$.

By applying Lemma B.3.2 to the second assumption of (B.50) we have:

$$\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta_z) \vdash z \triangleright \mathcal{H}^*(C) \qquad (\text{B.121})$$

Let $\widetilde{x} = \widetilde{x}' \setminus z$ and $\widetilde{y} = \widetilde{y}' \setminus \widetilde{z}$ such that $|\widetilde{z}| = \mathcal{H}^*(C)$ where. We may notice that by Definition 5.3.9 $\mathtt{indexed}_{\Gamma,\Delta,r:S}(\widetilde{y}, \widetilde{x})$ holds. We define $\Theta = \Theta_1, \Theta'$, where

$$\Theta' = c_k : ?(\widetilde{M});\mathsf{end}, \overline{c_{k+1}} : !\langle \widetilde{M}' \rangle;\mathsf{end}$$

with $\widetilde{M} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta, , u_i : ?(C);S))(\widetilde{y})$. By Definition 5.3.3, $\lfloor P \rceil^* = \lfloor P' \rceil^* + 1$ so

$$\mathtt{dom}(\Theta) = \{c_k, \dots, c_{k+\lfloor P \rceil^* - 1}\} \cup \{\overline{c_{k+1}}, \dots, \overline{c_{k+\lfloor P \rceil^* - 1}}\}$$

and Θ is balanced since $\Theta(c_{k+1})$ dual $\Theta(\overline{c_{k+1}})$ and Θ_1 is balanced. By Table 5.4:

$$\mathbf{A}^k_{\widetilde{y}}(r?(z).P') = c_k?(\widetilde{y}).r_{\iota(S)}?(\widetilde{z}).\overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \mid \mathcal{A}^{k+1}_{\widetilde{y}'}(P')$$

Also, let $\Gamma_1 = \Gamma \setminus \widetilde{x}$ and $\Delta_1 = \Delta \setminus \widetilde{x}$. We may notice that $\Delta_1 = \Delta_1'$. We shall prove the following judgment:

$$\mathcal{H}^*(\Gamma_1 \setminus z); \mathcal{H}^*(\Delta_1), \Theta \vdash \mathcal{A}^k_{\widetilde{y}}(r?(z).P')$$

Let $\Delta_1 = \Delta, r : S$. The left-hand side component of $\mathcal{A}^k_{\widetilde{y}}(r?(z).P')$ is typed using some auxiliary derivations:

$$\text{PolySend}\ \dfrac{\text{End}\ \dfrac{\text{Nil}\ \dfrac{}{\mathcal{H}^*(\Gamma); \emptyset \vdash \mathbf{0} \triangleright \diamond}}{\mathcal{H}^*(\Gamma); \overline{c_{k+1}} : \mathsf{end} \vdash \mathbf{0} \triangleright \diamond} \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma); \mathcal{H}^*(\Delta \setminus \Delta_1) \vdash \widetilde{y}' \triangleright \widetilde{M}'}}{\mathcal{H}^*(\Gamma); \overline{c_{k+1}} : !\langle \widetilde{M}' \rangle;\mathsf{end}, \mathcal{H}^*(\Delta \setminus \Delta_1) \vdash \overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \triangleright \diamond} \qquad (\text{B.122})$$

$$\text{End}\;\dfrac{\text{Rcv}\;\dfrac{(\text{B.122}) \qquad (\text{B.121})}{\mathcal{H}^*(\Gamma \setminus z); \overline{c_{k+1}} :!\langle\widetilde{M'}\rangle;\mathtt{end}, \mathcal{H}^*(\Delta_2) \vdash r_{\iota(S)}?(\widetilde{z}).\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.\mathbf{0} \triangleright \diamond}}{\mathcal{H}^*(\Gamma \setminus z); \overline{c_{k+1}} :!\langle\widetilde{M'}\rangle;\mathtt{end}, c_k : \mathtt{end}, \mathcal{H}^*(\Delta_2) \vdash r_{\iota(S)}?(\widetilde{z}).\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.\mathbf{0} \triangleright \diamond} \tag{B.123}$$

$$\text{PolyRcv}\;\dfrac{(\text{B.123}) \qquad \text{PolyVar}\;\dfrac{}{\mathcal{H}^*(\Gamma \setminus z); \mathcal{H}^*(\Delta_2) \vdash \widetilde{y} \triangleright \widetilde{M}}}{\mathcal{H}^*(\Gamma_1 \setminus z); \Theta' \vdash c_k?(\widetilde{y}).r_{\iota(S)}?(z).\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.\mathbf{0} \triangleright \diamond} \tag{B.124}$$

where $\Delta_2 = \Delta, r : S \setminus \Delta_1$. We may notice that by Definition 5.3.2 and Figure 5.5 we have $\mathcal{H}^*(\Delta_2) = \mathcal{H}^*(\Delta), \widetilde{r} : \mathcal{R}^\star(S) \setminus \mathcal{H}^*(\Delta_1)$. Further, by Lemma B.3.1 we know $\mathcal{H}^*(\Delta_2)(r_{\iota(S)}) = \mu\mathtt{t}.?(\mathcal{H}^*(C));\mathtt{t}$.

The following tree proves this case:

$$\text{Par}\;\dfrac{(\text{B.124}) \qquad \dfrac{(\text{B.120})}{\mathcal{H}^*(\Gamma_1' \setminus z); \mathcal{H}^*(\Delta_1), \Theta_1 \vdash \mathcal{A}^{k+1}_{\widetilde{y}'}(P') \triangleright \diamond}}{\mathcal{H}^*(\Gamma_1 \setminus z); \mathcal{H}^*(\Delta_1), \Theta \vdash c_k?(\widetilde{y}).r_{\iota(S)}?(\widetilde{z}).\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.\mathbf{0} \mid \mathcal{A}^{k+1}_{\widetilde{y}'}(P') \triangleright \diamond} \tag{B.125}$$

9. Case $P = (\nu\, s : \mu\mathtt{t}.S)\, P'$. For this case Rule ResS can be applied:

$$\text{ResS}\;\dfrac{\Gamma; \Delta, s : \mu\mathtt{t}.S, \overline{s} : \overline{\mu\mathtt{t}.S} \vdash P'}{\Gamma; \Delta \vdash (\nu\, s : \mu\mathtt{t}.S)\, P'} \tag{B.126}$$

By IH on the assumption of (B.126) we have:

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}'); \mathcal{H}^*(\Delta \setminus \widetilde{x}'), \Theta_1 \vdash \mathcal{A}^{k+1}_{\widetilde{y}'}(P') \tag{B.127}$$

where we take $\widetilde{y}'$ such that $\widetilde{s}, \widetilde{\overline{s}} \subseteq \widetilde{y}'$ with $\widetilde{s} = (s_1, \ldots, s_{|\mathcal{R}(S)|})$ and $\widetilde{\overline{s}} = (\overline{s}_1, \ldots, \overline{s}_{|\mathcal{R}(S)|})$. . Accordingly, $\widetilde{x}'$ is such that $s, \overline{s} \subseteq \widetilde{x}'$. Since $\mathtt{lin}(s)$ and $\mathtt{lin}(\overline{s})$ we know $\widetilde{x}' \subseteq \mathtt{fn}(P')$. Also, $\mathtt{indexed}_{\Gamma,\Delta_1}(\widetilde{y}', \widetilde{x}')$ where $\Delta_1 = \Delta, s : \mu\mathtt{t}.S, \overline{s} : \overline{\mu\mathtt{t}.S}$. Also, $\mathsf{balanced}(\Theta_1)$ with

$$\mathtt{dom}(\Theta_1) = \{c_{k+1}, \ldots, c_{k+\lfloor P' \rceil^*}\} \cup \{\overline{c_{k+2}}, \ldots, \overline{c_{k+\lfloor P' \rceil^*}}\}$$

and $\Theta_1(c_{k+1}) = ?(\widetilde{M'});\mathtt{end}$ where $\widetilde{M} = (\mathcal{H}^*(\Gamma), \mathcal{H}^*(\Delta))(\widetilde{y}')$.

Let $\widetilde{y} = \widetilde{y}' \setminus (\widetilde{s}, \widetilde{\overline{s}})$ and $\widetilde{x} = \widetilde{x}' \setminus (s, \overline{s})$. Since $s, \overline{s} \notin \mathtt{fn}(P)$ we know $\widetilde{x} \subseteq \mathtt{fn}(P)$ and $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{y}, \widetilde{x})$.

We define $\Theta = \Theta_1, \Theta'$ where

$$\Theta' = c_k :?(\widetilde{M});\mathtt{end}, \overline{c_{k+1}} :!\langle\widetilde{M'}\rangle;\mathtt{end}$$

By Table 5.4, we have:

$$\mathbf{A}^k_{\widetilde{y}}(P) = (\nu\, \widetilde{s} : \mathcal{R}(S))\, (c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle\widetilde{y}'\rangle.\mathbf{0} \mid \mathcal{A}^k_{\widetilde{y}'}(P'))$$

We should prove the following judgment:

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x}); \mathcal{H}^*(\Delta \setminus \widetilde{x}), \Theta \vdash \mathcal{A}^k_{\widetilde{y}}(P) \triangleright \diamond$$

We use some auxiliary sub-tree:

$$\text{PolySend}\ \dfrac{\text{Nil}\ \dfrac{}{\mathcal{H}^*(\Gamma);\emptyset \vdash \mathbf{0} \triangleright \diamond} \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma);\mathcal{H}^*(\Delta_1 \cap \widetilde{x}') \vdash \widetilde{y}' \triangleright \widetilde{M}'}}{\mathcal{H}^*(\Gamma);\mathcal{H}^*(\Delta \cap \widetilde{x}),\widetilde{s}:\mathcal{R}(S),\widetilde{\overline{s}}:\mathcal{R}(\overline{S}) \vdash \overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \triangleright \diamond} \tag{B.128}$$

here we may notice that

$$\mathcal{H}^*(\Delta \cap \widetilde{x}),\widetilde{s}:\mathcal{R}(S),\widetilde{\overline{s}}:\mathcal{R}(\overline{S}) = \mathcal{H}^*(\Delta_1 \cap \widetilde{x}')$$

The following tree proves this case:

$$\text{PolyResS}\ \dfrac{\text{Par}\ \dfrac{\text{PolyRcv}\ \dfrac{(\text{B.128}) \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma);\mathcal{H}^*(\Delta \cap \widetilde{x}) \vdash \widetilde{y} \triangleright \widetilde{M}}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x});\Theta',\widetilde{s}:\mathcal{R}(S),\widetilde{\overline{s}}:\mathcal{R}(\overline{S}) \vdash c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \triangleright \diamond} \qquad (\text{B.127})}{\mathcal{H}^*(\Gamma \setminus \widetilde{x});\mathcal{H}^*(\Delta \setminus \widetilde{x}),\Theta,\widetilde{s}:\mathcal{R}(S),\widetilde{\overline{s}}:\mathcal{R}(\overline{S}) \vdash c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \mid \mathcal{A}^k_{\widetilde{y}'}(P') \triangleright \diamond}}{\mathcal{H}^*(\Gamma \setminus \widetilde{x});\mathcal{H}^*(\Delta \setminus \widetilde{x}),\Theta \vdash (\nu\, \widetilde{s}:\mathcal{R}(S))\,(c_k?(\widetilde{y}).\overline{c_{k+1}}!\langle \widetilde{y}' \rangle.\mathbf{0} \mid \mathcal{A}^k_{\widetilde{y}'}(P')) \triangleright \diamond} \tag{B.129}$$

□

B.3.4 Proof of Theorem 5.3.1

Theorem 5.3.1 (Minimality Result for π, Optimized)**.** *Let P be a process with $\widetilde{u} = \mathtt{fn}(P)$. If $\Gamma;\Delta \vdash P \triangleright \diamond$ then $\mathcal{H}^*(\Gamma\sigma);\mathcal{H}^*(\Delta\sigma) \vdash \mathcal{F}^*(P) \triangleright \diamond$, where $\sigma = \{\mathsf{init}(\widetilde{u})/\widetilde{u}\}$.*

Proof. By assumption, $\Gamma;\Delta \vdash P \triangleright \diamond$. Then by applying Lemma 2.2.1 we have:

$$\Gamma\sigma;\Delta\sigma \vdash P\sigma \triangleright \diamond \tag{B.130}$$

By Definition 5.3.7, we shall prove the following judgment:

$$\mathcal{H}^*(\Gamma\sigma);\mathcal{H}^*(\Delta\sigma) \vdash (\nu\, \widetilde{c})\,(\overline{c_k}!\langle \widetilde{r} \rangle.\mathbf{0} \mid \mathcal{A}^k_{\widetilde{r}}(P\sigma)) \triangleright \diamond \tag{B.131}$$

where $\widetilde{c} = (c_k,\ldots,c_{k+\lfloor P \rceil^*-1})$; $k > 0$; and $\widetilde{r} = \bigcup_{v\in\widetilde{v}}\{v_1,\ldots,v_{|\mathcal{H}^*(S)|}\}$ with $v:S$. Since $\widetilde{v} \subseteq \mathtt{fn}(P)$ we know $\mathtt{indexed}_{\Gamma\sigma,\Delta\sigma}(\widetilde{r},\widetilde{v})$. Since $P\sigma$ is an initialized process, we apply Lemma 5.3.3 to (B.130) to get:

$$\mathcal{H}^*(\Gamma\sigma);\mathcal{H}^*(\Delta\sigma \setminus \widetilde{v}),\Theta \vdash \mathcal{A}^k_{\widetilde{r}}(P\sigma) \triangleright \diamond \tag{B.132}$$

where Θ is balanced with $\mathtt{dom}(\Theta) = \{c_k,c_{k+1},\ldots,c_{k+\lfloor P \rceil^*-1}\} \cup \{\overline{c_{k+1}},\ldots,\overline{c_{k+\lfloor P \rceil^*-1}}\}$ and $\Theta(c_k) = ?(\widetilde{M});\mathtt{end}$ with $\widetilde{M} = \mathcal{H}^*(\Delta)(\widetilde{r})$.

The following tree proves this case:

$$\text{ResS}\ \dfrac{\text{Par}\ \dfrac{\text{Send}\ \dfrac{\text{End}\ \dfrac{\text{Nil}\ \dfrac{}{\mathcal{H}^*(\Gamma\sigma);\emptyset \vdash \mathbf{0} \triangleright \diamond}}{\mathcal{H}^*(\Gamma\sigma);c_k:\mathsf{end} \vdash \mathbf{0} \triangleright \diamond} \qquad \text{PolyVar}\ \dfrac{}{\mathcal{H}^*(\Gamma\sigma);\widetilde{r}:\widetilde{M} \vdash \widetilde{r} \triangleright \widetilde{M}}}{\mathcal{H}^*(\Gamma\sigma);\overline{c_k}:!\langle \widetilde{M} \rangle;\mathtt{end},\widetilde{r}:\widetilde{M} \vdash \overline{c_k}!\langle \widetilde{r} \rangle.\mathbf{0} \triangleright \diamond} \qquad (\text{B.132})}{\mathcal{H}^*(\Gamma\sigma);\mathcal{H}^*(\Delta\sigma),\overline{c_k}:!\langle \widetilde{M} \rangle;\mathtt{end},\Theta \vdash \overline{c_k}!\langle \widetilde{r} \rangle.\mathbf{0} \mid \mathcal{A}^k_{\widetilde{r}}(P\sigma) \triangleright \diamond}}{\mathcal{H}^*(\Gamma\sigma);\mathcal{H}^*(\Delta\sigma) \vdash (\nu\, \widetilde{c})\,(\overline{c_k}!\langle \widetilde{r} \rangle.\mathbf{0} \mid \mathcal{A}^k_{\widetilde{r}}(P\sigma)) \triangleright \diamond}$$

□

B.3.5 An Auxiliary Definition

For convenience, we define function $\widehat{\mathcal{C}}^{-}_{-}(\,\cdot\,)$ relying on $\mathcal{C}^{-}_{-}(\,\cdot\,)$ as follows:

Definition B.3.1 (Function $\widehat{\mathcal{C}}^{-}_{-}(\,\cdot\,)$). Let P be a process, ρ be a name substitution, and σ be an indexed name substitution. We define $\widehat{\mathcal{C}}^{\rho}_{\sigma}(P)$ as follows:

$$\widehat{\mathcal{C}}^{\rho}_{\sigma}(P_1) = \mathcal{C}^{\widetilde{u_*}}_{\widetilde{x_*}}(P\sigma)$$
$$\text{with } \rho = \{\tilde{u}/\tilde{x}\}, \widetilde{u_*} = \mathtt{bn}(\tilde{u}\sigma : \widetilde{C}), \widetilde{x_*} = \mathtt{bn}(\tilde{x}\sigma : \widetilde{C})$$

where $\widetilde{u} : \widetilde{C}$.

B.3.6 Proof of Lemma 5.3.7

Lemma 5.3.7. *Assume $P_1\{\tilde{u}/\tilde{x}\}$ is a process and $P_1\{\tilde{u}/\tilde{x}\}\,\mathcal{S}\,Q_1$.*

1. *Whenever $P_1\{\tilde{u}/\tilde{x}\}\xrightarrow{(\nu\,\widetilde{m_1})\,n!\langle v:C_1\rangle}P_2$, such that $\overline{n} \notin P_1\{\tilde{u}/\tilde{x}\}$, then there exist Q_2 and σ_v such that $Q_1\xRightarrow{(\nu\,\widetilde{m_2})\,\breve{n}!\langle\widetilde{v}:\mathcal{H}^*(C_1)\rangle}Q_2$ where $v\sigma_v \diamond \widetilde{v}$ and, for a fresh t,*
$$(\nu\,\widetilde{m_1})(P_2 \parallel t \Leftarrow_{\mathtt{C}} v : C_1) \;\; \mathcal{S} \;\; (\nu\,\widetilde{m_2})(Q_2 \parallel t_1 \Leftarrow_{\mathtt{m}} v\sigma_v : C_1)$$

2. *Whenever $P_1\{\tilde{u}/\tilde{x}\}\xrightarrow{n?(v)}P_2$, such that $\overline{n} \notin P_1\{\tilde{u}/\tilde{x}\}$, then there exist Q_2 and σ_v such that $Q_1\xRightarrow{\breve{n}?(\widetilde{v})}Q_2$ where $v\sigma_v \diamond \widetilde{v}$ and $P_2 \;\; \mathcal{S} \;\; Q_2$,*

3. *Whenever $P_1\xrightarrow{\tau}P_2$ then there exists Q_2 such that $Q_1\xRightarrow{\tau}Q_2$ and $P_2 \;\; \mathcal{S} \;\; Q_2$.*

Proof. By transition induction. Let $\rho_1 = \{\tilde{u}/\tilde{x}\}$. By inversion of $P_1\,\mathcal{S}\,Q_1$ we know there is $\sigma_1 \in \mathsf{index}(\mathtt{fn}(P_1) \cup \tilde{u} \cup \tilde{x})$, such that $Q_1 \in \widehat{\mathcal{C}}^{\rho_1}_{\sigma_1}(P_1)$. We need the following assertion on index substitutions. If $P_1\rho_1 \xrightarrow{\ell} P_2\rho_2$ and $\mathtt{subj}(\ell) = n$ then there exists Q_2 such that $Q_1 \xrightarrow{\breve{\ell}} Q_2$ with $\mathtt{subj}(\breve{\ell}) = n_i$ and $Q_2 \in \widehat{\mathcal{C}}^{\rho_2}_{\sigma_2}(P_2)$ such that $\sigma_2 \in \mathsf{index}(P_2\rho_2)$, $\mathsf{next}(n_i) \in \sigma_2$, and $\sigma_1 \cdot (\sigma_2 \setminus \mathsf{next}(n_i)) = (\sigma_2 \setminus \mathsf{next}(n_i)) \cdot \sigma_1$.

First, we consider two base cases: Rules $\mathtt{Snd}$ and $\mathtt{Rv}$. Then, we consider inductive cases: Rules $\mathtt{Par_L}$ and $\mathtt{Tau}$. We omit inductive cases $\mathtt{New}$ and $\mathtt{Res}$ as they follow directly by the inductive hypothesis and the definition of the restriction case in $\mathcal{C}^{-}_{-}(\,\cdot\,)$ (Table 5.6). Finally, we separately treat cases when a process is recursive. We show two cases ($\mathtt{Rv}$ and $\mathtt{Par_L}$) emphasizing specifics of the recursion breakdown.

1. Case $\mathtt{Snd}$. We note that we only consider the case when P_1 is a pure process, as the case when P_1 is a trigger collection follows directly by Lemma 5.3.5.

   We distinguish two sub-cases: (i) $P_1 = n!\langle v\rangle.P_2$ and (ii) $P_1 = n!\langle y\rangle.P_2$ with $\{v/y\} \in \rho_1$. In both sub-cases we know there is $\rho_2 = \{\tilde{u}_2/\tilde{x}_2\}$ such that
$$P_1\rho_1 = n\rho_1!\langle v\rangle.P_2\rho_2$$

   We have the following transition:
$$\langle\mathtt{Snd}\rangle\,\frac{}{P_1\rho_1 \xrightarrow{n\rho_1!\langle v\rangle} P_2\rho_2}$$

   Let $\sigma_1 \in \mathsf{index}(\widetilde{p})$ where $\widetilde{p} = \mathsf{index}(\mathtt{fn}(P_1) \cup \tilde{u} \cup \tilde{x})$ such that $\{n_i/n\} \in \sigma_1$ and $\sigma_2' = \sigma_2 \cdot \{t_1/t\} \cdot \{t_1/t\}$. Here we take $\sigma_v = \sigma_1$, as we know $v \in \mathtt{fn}(P_1)$. Further, let

$\widetilde{u_*} = \mathsf{bn}(\widetilde{u}\sigma_1 : \widetilde{C})$ and $\widetilde{x_*} = \mathsf{bn}(\widetilde{x}\sigma_1 : \widetilde{C})$ with $\widetilde{u} : \widetilde{C}$. Also, let $\widetilde{z} = \mathtt{fnb}(P_2, \widetilde{x_*} \setminus \widetilde{w})$ where $\widetilde{w} = \{n_i\}$ if $\mathtt{lin}(n_i)$ otherwise $\widetilde{w} = \epsilon$. Then, in both sub-cases, there are two possibilities for the shape of Q_1, namely:

$$\begin{aligned} Q_1^1 &= (\nu\, \widetilde{c}_k)\, (\overline{c_k}!\langle\widetilde{u_*}\rangle \mid \mathcal{A}^k_{\widetilde{x_*}}(P_1\sigma_1)) \\ Q_1^2 &= (\nu\, \widetilde{c}_{k+1})\, n_i\rho_*!\langle\widetilde{v}\rangle.\overline{c_{k+1}}!\langle\widetilde{z}\rho_*\rangle \mid \mathcal{A}^{k+1}_{\widetilde{z}}(P_2\sigma_2) \end{aligned}$$

where $v\sigma_v \diamond \widetilde{v}$ and $\rho_* = \{\widetilde{u}_*/\widetilde{x}_*\}$ By Lemma 5.3.6 we know that $Q_1^1 \xrightarrow{\tau} Q_1^2$. Thus, we only consider how Q_1^2 evolves. We can infer the following:

$$Q_1^2 \xrightarrow{n_i\rho_*!\langle\widetilde{v}\rangle} Q_2$$

where $Q_2 = (\nu\, \widetilde{c}_{k+1})\, \overline{c_{k+1}}!\langle\widetilde{z}\rho_*\rangle \mid \mathcal{A}^{k+1}_{\widetilde{z}}(P_2\sigma_2)$. Then, we should show the following:

$$(P_2 \parallel t \Leftarrow_{\mathtt{C}} v : C_1)\rho_2 \;\mathcal{S}\; (Q_2 \parallel t_1 \Leftarrow_{\mathtt{m}} v\sigma_v : C_1) \tag{B.133}$$

By Table 5.6 we can see that

$$Q_2 \in \mathcal{C}^{\widetilde{z}\rho_*}_{\widetilde{z}}(P_2\sigma_2) \tag{B.134}$$

Here we remark that our assertion holds by the definition as we have $\sigma_2 = \sigma_1 \cdot \mathsf{next}(n_i)$. Next, by Lemma 5.3.4 we have

$$(t \Leftarrow_{\mathtt{C}} v : C_1)\sigma_2 \diamond (t_1 \Leftarrow_{\mathtt{m}} v\sigma_v : C_1)$$

That is, we have

$$(t_1 \Leftarrow_{\mathtt{m}} v\sigma_v : C_1) \in \mathcal{C}^{\widetilde{z}\rho_*}_{\widetilde{z}}((t \Leftarrow_{\mathtt{C}} v : C_1)\sigma_2) \tag{B.135}$$

Thus, by (B.134) and (B.135) we have

$$(Q_2 \parallel t_1 \Leftarrow_{\mathtt{m}} v\sigma_v : C_1) \in \mathcal{C}^{\widetilde{z}\rho_*}_{\widetilde{z}}((P_2 \parallel t \Leftarrow_{\mathtt{C}} v : C_1)\sigma_2) \tag{B.136}$$

Now, by Definition 5.3.4 and Definition 5.3.26 we know that

$$\widetilde{z} = \mathtt{fnb}(P_2, \mathsf{bn}(\widetilde{x} : \widetilde{C}) \setminus \widetilde{w}) = \mathsf{bn}(\widetilde{x}_2\sigma_2 : \widetilde{C}_2)$$

with $\widetilde{x}_2 : \widetilde{C}_2$. Similarly, we have

$$\widetilde{z}\rho_* = \mathsf{bn}(\widetilde{x}_2\rho_2\sigma_2 : \widetilde{C}_2)$$

Now, by the assumption $\overline{n} \notin \widetilde{v}$ we know $\{\overline{n}_i/\overline{n}\} \notin \sigma_2$. Thus, by $\sigma_2 = \sigma_1 \cdot \mathsf{next}(n_i) \cdot \{t_1/t\}$ and Definition 5.3.27, we have

$$\sigma_2 \in \mathsf{index}(\mathtt{fn}(P_2 \parallel t \Leftarrow_{\mathtt{C}} v : C_1) \cup \widetilde{u}_2 \cup \widetilde{x}_2)$$

By this and (B.136) the goal (B.133) follows. This concludes $\mathtt{Snd}$ case.

2. Case Rv. As in above case (Snd), we only consider the case when P_1 is a pure process, as the case when P_1 is a trigger collection follows directly by Lemma 5.3.5.

   Here we know $P_1 = n?(y).P_2$. We know there is $\rho_2 = \{\tilde{u}_2/\tilde{x}_2\}$ such that

$$P_1\rho_1 = n\rho_1?(y).P_2\rho_2$$

   The transition is as follows:

$$\langle \texttt{Rv} \rangle \frac{}{n\rho_1?(y).P_2\rho_2 \xrightarrow{n?(v)} P_2\rho_2 \cdot \{v/y\}}$$

   Let $\sigma_1 \in \mathsf{index}(\mathtt{fn}(P_1) \cup \tilde{u} \cup \tilde{x})$ and $\sigma_2 = \sigma_1 \cdot \mathsf{next}(n_i) \cdot \{y_1/y\}$. Further, let $\widetilde{u_*} = \mathsf{bn}(\tilde{u}\sigma_1 : \widetilde{C})$ and $\widetilde{x_*} = \mathsf{bn}(\tilde{x}\sigma_1 : \widetilde{C})$ with $\tilde{u} : \widetilde{C}$. Also, let $\tilde{y} = (y_1, \ldots, y_{|\mathcal{H}^*(S)|})$ and $\tilde{z} = \mathtt{fnb}(P_2, \widetilde{x_*}\tilde{y} \backslash \tilde{w})$ where $\tilde{w} = \{n_i\}$ if $\mathtt{lin}(n_i)$ otherwise $\tilde{w} = \epsilon$. Then, there are two possibilities for the shape of Q_1, namely:

$$\begin{aligned} Q_1^1 &= (\nu\, \tilde{c}_k)\, (\overline{c_k}!\langle \widetilde{u_*} \rangle \mid \mathcal{A}^k_{\widetilde{x_*}}(P_1\sigma_1)) \\ Q_1^2 &= (\nu\, \tilde{c}_{k+1})\, n_i\rho?(\tilde{y}).\overline{c_{k+1}}!\langle \tilde{z}\rho \rangle \mid \mathcal{A}^{k+1}_{\tilde{z}}(P_2\sigma_2) \end{aligned}$$

   By Lemma 5.3.6 we know that $Q_1^1 \xrightarrow{\tau} Q_1^2$. Thus, we only consider how Q_1^2 evolves. We can infer the following:

$$Q_1^2 \xrightarrow{n_i\rho?\langle \tilde{v} \rangle} Q_2$$

   where $Q_2 = (\nu\, \tilde{c}_{k+1})\, \overline{c_{k+1}}!\langle \tilde{z}\rho \cdot \{\tilde{v}/\tilde{y}\} \rangle \mid \mathcal{A}^{k+1}_{\tilde{z}}(P_2\sigma_2)$ and $v\sigma_v \diamond \tilde{v}$ for some $\sigma_v \in \mathsf{index}(v)$. Now, we should show the following:

$$P_2\rho_2 \cdot \{v/y\}\, \mathcal{S}\, Q_2 \tag{B.137}$$

   By Table 5.6 we can see that

$$Q_2 \in \mathcal{C}^{\tilde{z}\rho_2 \cdot \{\tilde{v}/\tilde{y}\}}_{\tilde{z}}(P_2\sigma_2 \cdot \sigma_v) \tag{B.138}$$

   We may notice that

$$\sigma_2 \cdot \sigma_v \in \mathsf{index}(\mathtt{fn}(P_2) \cup \tilde{u}_2 \cup \tilde{x}_2)$$

   Futher, by the definition we have $(\sigma_2 \cdot \sigma_v) \setminus \mathsf{next}(n_i) = \sigma_1 \cdot \sigma_v$. As $v \notin (\mathtt{fn}(P_1) \cup \tilde{u} \cup \tilde{x}) = \mathtt{codom}(\sigma_1)$, we have $\sigma_1 \cdot \sigma_v = \sigma_v \cdot \sigma_1$. That is, our assertion on index substitutions holds.

   By Definition 5.3.4 and Definition 5.3.26 we have that

$$\begin{aligned} \tilde{z} &= \mathtt{fnb}(P_2, \mathsf{bn}(\tilde{x} : \widetilde{C}) \cdot \tilde{y} \setminus \tilde{w}) \\ &= \mathsf{bn}(\tilde{x}y(\sigma_1 \cdot \mathsf{next}(n_i) \cdot \{y_1/y\}) : \widetilde{C}S) \end{aligned}$$

   with $y : S$. Similarly, we have

$$\tilde{z}\rho_2 \cdot \{\tilde{v}/\tilde{y}\} = \mathsf{bn}(\tilde{x}y\rho_2 \cdot \{v/y\}(\sigma_2 \cdot \sigma_v) : \widetilde{C}S)$$

   Thus, by this and (B.138) the goal (B.137) follows. This concludes Rv case (and the base cases). We remark that base cases concerning triggers collection processes follow by Lemma 5.3.5. Next, we consider inductive cases.

3. Case $\mathtt{Par_L}$. Here we know $P_1 = P_1' \mid P_1''$. Let ρ_1' and ρ_1'' be such that

$$P_1\rho_1 = P_1'\rho_1' \mid P_1''\rho_1''$$

The final rule in the inference tree is as follows:

$$\langle \mathtt{Par}_L \rangle \ \frac{P_1'\rho_1' \xrightarrow{\ell} P_2'\rho_2' \qquad \mathtt{bn}(\ell) \cap \mathtt{fn}(P_1'') = \emptyset}{P_1'\rho_1' \mid P_1''\rho_1'' \xrightarrow{\ell} P_2'\rho_2' \mid P_1''\rho_1''}$$

Let $\sigma_1 \in \mathsf{index}(\mathtt{fn}(P_1) \cup \widetilde{u} \cup \widetilde{x})$. Further, let σ_1' and σ_1'' such that

$$P_1\rho_1\sigma_1 = P_1'\rho_1'\sigma_1' \mid P_1''\rho_1''\sigma_1''$$

Further, let $\widetilde{y} = \mathtt{fnb}(P_1', \widetilde{x_*})$, $\widetilde{z} = \mathtt{fnb}(P_1'', \widetilde{x_*})$, $\widetilde{u_*} = \mathsf{bn}(\widetilde{u}\sigma_1 : \widetilde{C})$, $\widetilde{x_*} = \mathsf{bn}(\widetilde{x}\sigma_1 : \widetilde{C})$ with $\widetilde{u} : \widetilde{C}$, and $\rho_{\mathtt{m}} = \{\tilde{u}_*/\tilde{x}_*\}$. In sub-case (i), by the definition of $\mathcal{S}$ (Table 5.6), there are following possibilities for Q_1:

$$Q_1^1 = (\nu\, \widetilde{c}_k)\, (\overline{c_k}!\langle \widetilde{u_*}\rangle \mid \mathcal{A}^k_{\widetilde{x_*}}(P_1\sigma_1'))$$
$$Q_1^2 = \overline{c_k}!\langle \widetilde{y}\rho_{\mathtt{m}}\rangle.\overline{c_{k+l}}!\langle \widetilde{z}\rho_{\mathtt{m}}\rangle \mid \mathcal{A}^k_{\widetilde{y}}(P_1'\sigma_1') \mid \mathcal{A}^{k+l}_{\widetilde{z}}(P_1''\sigma_1'')$$
$$N_1^3 = \Big\{(R_1' \mid R_1'') : \mathcal{C}^{\widetilde{y}\rho_{\mathtt{m}}}_{\widetilde{y}}(P_1'\sigma_1'), R_1'' \in \mathcal{C}^{\widetilde{z}\rho_{\mathtt{m}}}_{\widetilde{z}}(P_1''\sigma_1'')\Big\}$$

By Lemma 5.3.6 there exist

$$Q_1' \in \widehat{\mathcal{C}}^{\rho_1'}_{\sigma_1'}(P_1') \tag{B.139}$$
$$Q_1'' \in \mathcal{C}^{\rho_1''}_{\sigma_1''}(P_1'') \tag{B.140}$$

such that

$$Q_1^1 \overset{\tau}{\Rightarrow} Q_1^2 \overset{\tau}{\Rightarrow} Q_1' \mid Q_1''$$

Thus, in both cases we consider how $Q_1' \mid Q_1''$ evolves. By the definition of $\mathcal{S}$ we have

$$P_1'\rho_1' \ \mathcal{S}\ Q_1' \tag{B.141}$$
$$P_1''\rho_1'' \ \mathcal{S}\ Q_1'' \tag{B.142}$$

To apply IH we do the case analysis on the action ℓ:

- Sub-case $\ell \equiv n?\langle v\rangle$. If $v \in \widetilde{u}$ then we take $\sigma_v = \sigma_1$, otherwise $\sigma_v = \{v_j/v\}$ for $j > 0$. Then, by applying IH to (B.141) we know there is Q_2' such that $Q_1' \xRightarrow{n_i?\langle\widetilde{v}\rangle} Q_2'$ and

$$P_2'\{\widetilde{u}_2'/\widetilde{y}_2\} \ \ \mathcal{S}\ \ Q_2' \tag{B.143}$$

where $v\sigma_v \diamond \widetilde{v}$ and $\widetilde{u}_2' = \widetilde{u}_1' \cdot \widetilde{v}$. We should show that

$$P_2'\rho_2' \mid P_1''\rho_1'' \,\mathcal{S}\, Q_2' \mid Q_1'' \tag{B.144}$$

Now, by the assertion on index substitutions, we have $\sigma_2' \in \mathsf{index}(\mathtt{fn}(P_2') \cup \rho_2')$ such that $\mathsf{next}(n_i) \in \sigma_2'$, $\sigma_1' \cdot (\sigma_2' \setminus \mathsf{next}(n_i)) = (\sigma_2' \setminus \mathsf{next}(n_i)) \cdot \sigma_1'$, and

$$Q_2' \in \widehat{\mathcal{C}}^{\rho_2'}_{\sigma_2'}(P_2') \tag{B.145}$$

Now, as by the definition we have $\sigma_1' \cdot \sigma_1'' = \sigma_1'' \cdot \sigma_1'$, it follows $\sigma_1'' \cdot (\sigma_2' \setminus \mathsf{next}(n_i)) = (\sigma_2' \setminus \mathsf{next}(n_i)) \cdot \sigma_1''$.

Thus, the following holds

$$\widehat{\mathcal{C}}^{\rho_1''\cdot\rho_2'}_{\sigma_2'\cdot\sigma_1''}(P_1'') = \widehat{\mathcal{C}}^{\rho_1''}_{\sigma_1''}(P_1'')$$
$$\widehat{\mathcal{C}}^{\rho_1''\cdot\rho_2'}_{\sigma_2'\cdot\sigma_1''}(P_2') = \widehat{\mathcal{C}}^{\rho_2'}_{\sigma_2'}(P_2')$$

By this, (B.145), and (B.140) we have

$$Q_2' \mid Q_1'' \in \widehat{\mathcal{C}}^{\rho_1''\cdot\rho_2'}_{\sigma_1''\cdot\sigma_2'}(P_1'' \mid P_2') \tag{B.146}$$

Further, we may notice that $\sigma_1'' \cdot \sigma_2' \in \mathsf{index}(\mathtt{fn}(P_2' \mid P_1'') \cup \rho_2' \cup \rho_1'')$ and $\mathsf{next}(n_i) \in \sigma_1'' \cdot \sigma_2'$ as by the assumption we have $\bar{n} \notin \mathtt{fn}(P_2' \mid P_1'') \cup \rho_2' \cup \rho_1''$. Thus, our assertion holds. Hence, by this and (B.146) the goal (B.144) follows.

- Sub-case $\ell = \tau$. By applying IH to (B.141) we know there is Q_2' such that $Q_1' \overset{\ell}{\Rightarrow} Q_2'$ and

$$P_2'\rho_2' \;\mathcal{S}\; Q_2' \tag{B.147}$$

where $\rho_2' = \{\tilde{u}_2'/\tilde{y}_2\}$. We should show that

$$P_2'\rho_2' \mid P_1''\rho_1'' \,\mathcal{S}\, Q_2' \mid Q_1'' \tag{B.148}$$

By (B.147), we know there is $\sigma_2 \in \mathsf{index}(\mathtt{fn}(P_2) \cup \rho_2')$ such that

$$Q_2' \in \widehat{\mathcal{C}}^{\rho_2'}_{\sigma_2}(P_2')$$

Further, by remark we know that either $\sigma_2' = \sigma_1 \cdot \{n_{i+1}/n_i\} \cdot \{\overline{n}_{i+1}/\overline{n}_i\}$ for some $n_i, \overline{n}_i \in \mathtt{fn}(P_2'\rho_2')$ and $n_i, \overline{n}_i \notin \mathtt{fn}(P_1''\rho_1'')$ or $\sigma_2' = \sigma_1$ such that

$$\widehat{\mathcal{C}}^{\rho_2'}_{\sigma_2}(P_2') = \widehat{\mathcal{C}}^{\rho_2'}_{\sigma_2'}(P_2')$$

By this we have that

$$\widehat{\mathcal{C}}^{\rho_1'}_{\sigma_2'}(P_1') = \widehat{\mathcal{C}}^{\rho_1'}_{\sigma_1}(P_1')$$

So, we know that

$$Q_2' \mid Q_1'' \in \widehat{\mathcal{C}}^{\rho_2'\cdot\rho_1''}_{\sigma_2'}(P_2' \mid P_1'')$$

By (B.147) and (B.140) and the definition of σ_2' the goal (B.148) follows.

- Sub-case $\ell \equiv (\nu\,\widetilde{m}_1)\, n!\langle v : C_1\rangle$. Here we omit details on substitutions as they are similar to the first sub-case. By applying IH to (B.141) we know there is Q_2' such that $Q_1' \overset{\ell}{\Rightarrow} Q_2'$ and

$$(\nu\,\widetilde{m_1})(P_2' \parallel t \Leftarrow_{\mathtt{C}} v : C_1) \;\mathcal{S} \tag{B.149}$$
$$(\nu\,\widetilde{m_2})(Q_2' \parallel t_1 \Leftarrow_{\mathtt{m}} v\sigma_1 : C_1) \tag{B.150}$$

We should show that

$$(\nu\,\widetilde{m_1})(P_2' \mid P_1'' \parallel t \Leftarrow_{\mathtt{C}} v : C_1) \;\mathcal{S} \tag{B.151}$$
$$(\nu\,\widetilde{m_2})(Q_2' \mid P_1'' \parallel t_1 \Leftarrow_{\mathtt{m}} v\sigma_1 : C_1) \tag{B.152}$$

By Definition 5.3.28 and Table 5.6 from (B.149) we can infer the following:

$$P_2'\{\tilde{u}_1'/\tilde{y}'\} \;\mathcal{S}\; Q_2' \tag{B.153}$$

So, by (B.142) and (B.153) the goal (B.151) follows.

This concludes $\mathtt{Par_L}$ case.

4. Case $\mathtt{Tau}$. Here we know $P_1 = P_1' \mid P_1''$. Without the loss of generality, we assume $\ell_1 = (\nu\, \widetilde{m}_1)\, \overline{n}!\langle v_1 \rangle$ and $\ell_2 = n?(v_1)$. Let $\rho_1' = \{\tilde{u}_1'/\tilde{y}\}$ and $\rho_1'' = \{\tilde{u}_1''/\tilde{z}\}$ such that

$$P_1\rho_1 = P_1'\rho_1' \mid P_1''\rho_1''$$

Then, the final rule in the inference tree is as follows:

$$\langle \mathtt{Tau} \rangle \dfrac{P_1'\rho_1' \xrightarrow{\ell_1} P_2'\rho_2' \qquad P_1''\rho_1'' \xrightarrow{\ell_2} P_2''\rho_2' \qquad \ell_1 \asymp \ell_2}{P_1'\rho_1' \mid P_1''\rho_1'' \xrightarrow{\tau} (\nu\, \widetilde{m}_1)\, (P_2'\rho_2' \mid P_2''\rho_2'')}$$

Let σ_1, $\widetilde{y}$, $\widetilde{z}$, $\widetilde{u_*}$, $\widetilde{x_*}$, and $\rho_{\mathtt{m}}$ be defined as in the previous case. Further, Q_1 can have shapes: Q_1^1, Q_1^2, and N_1^3 as in the previous case. As in the previous case, we know that there are

$$Q_1' \in \mathcal{C}_{\tilde{y}}^{\tilde{y}\rho_{\mathtt{m}}}(P_1'\sigma_1) \tag{B.154}$$
$$Q_1'' \in \mathcal{C}_{\tilde{z}}^{\tilde{z}\rho_{\mathtt{m}}}(P_1''\sigma_1) \tag{B.155}$$

such that

$$Q_1^1 \overset{\tau}{\Rightarrow} Q_1^2 \overset{\tau}{\Rightarrow} Q_1' \mid Q_1''$$

Thus, in both cases we consider how $Q_1' \mid Q_1''$ evolves. By the definition of $\mathcal{S}$ we have

$$P_1'\rho_1'\, \mathcal{S}\, Q_1' \tag{B.156}$$
$$P_1''\rho_1''\, \mathcal{S}\, Q_1'' \tag{B.157}$$

We apply IH component-wise:

- By applying IH to (B.156) we know there is Q_2' such that

  $$Q_1' \xRightarrow{(\nu\, \widetilde{m}_2)\, \overline{n}_i!\langle \widetilde{v} \rangle} Q_2' \tag{B.158}$$

  and

  $$(\nu\, \widetilde{m_1})(P_2' \parallel t \Leftarrow_{\mathtt{C}} v : C_1)\rho_2' \;\; \mathcal{S}\, (\nu\, \widetilde{m_2})(Q_2' \parallel t \Leftarrow_{\mathtt{m}} v\sigma_1 : C_1) \tag{B.159}$$

  where $v\sigma_v \diamond \widetilde{v}$ such that $\sigma_v \subseteq \sigma_1$.

  Now, by (B.159) we can infer:

  $$P_2'\rho_2'\, \mathcal{S}\, Q_2'$$

  Now, by the assertion on index substitutions, we have $\sigma_2' \in \mathsf{index}(\mathtt{fn}(P_2') \cup \rho_2')$ such that $\mathsf{next}(\overline{n}_i) \in \sigma_2'$, $\sigma_1' \cdot (\sigma_2' \setminus \mathsf{next}(\overline{n}_i)) = (\sigma_2' \setminus \mathsf{next}(\overline{n}_i)) \cdot \sigma_1'$, and

  $$Q_2' \in \widehat{\mathcal{C}}_{\sigma_2'}^{\rho_2'}(P_2')$$

  More precisely, here we know that $\sigma_2' \subseteq \sigma_1' \cdot \mathsf{next}(\overline{n}_i) \subseteq \sigma_1 \mathsf{next}(\overline{n}_i)$. So, we have

  $$Q_2' \in \widehat{\mathcal{C}}_{\sigma_1 \cdot \mathsf{next}(\overline{n}_i)}^{\rho_2'}(P_2') \tag{B.160}$$

- By applying IH to (B.157) we know there is Q''_2 such that

$$Q''_1 \xRightarrow{n_j?\langle\tilde{v}\rangle} Q''_2 \tag{B.161}$$

and

$$P''_2\rho''_2 \ \mathcal{S} \ Q''_2 \tag{B.162}$$

Now, by the assertion on index substitutions, we have $\sigma''_2 \in \mathsf{index}(\mathtt{fn}(P''_2) \cup \rho''_2)$ such that $\mathsf{next}(n_j) \in \sigma''_2$, $\sigma'_1 \cdot (\sigma''_2 \setminus \mathsf{next}(n_j)) = (\sigma''_2 \setminus \mathsf{next}(n_j)) \cdot \sigma'_1$, and

$$Q''_2 \in \widehat{\mathcal{C}}^{\rho''_2}_{\sigma''_2}(P''_2) \tag{B.163}$$

More precisely, we know that $\sigma''_2 = \sigma''_1 \cdot \sigma_v \cdot \mathsf{next}(n_j) \subseteq \sigma_1 \cdot \mathsf{next}(n_j)$. So, we have

$$Q''_2 \in \widehat{\mathcal{C}}^{\rho''_2}_{\sigma_1\cdot\mathsf{next}(n_j)}(P''_2) \tag{B.164}$$

Now, by (B.158) we know there is R' such that

$$Q'_1 \overset{\tau}{\Rightarrow} R' \xrightarrow{\breve{\ell}_1} Q'_2$$

where $\breve{\ell}_1 = (\nu\,\widetilde{m}_2)\,\overline{n}_i!\langle\tilde{v}\rangle$. Similarly, by (B.162) there is R'' such that

$$Q''_1 \overset{\tau}{\Rightarrow} R'' \xrightarrow{\breve{\ell}_2} Q''_2$$

where $\breve{\ell}_2 = n_j?(\tilde{w})$.

Now, to proceed we must show $\breve{\ell}_1 \asymp \breve{\ell}_2$, which boils down to showing that indices of $\overline{n}_i$ and n_j match. For this, we distinguish two sub-cases: (i) $\neg\mathsf{tr}(\overline{n}_i)$ and $\neg\mathsf{tr}(n_j)$ and (ii) $\mathsf{tr}(\overline{n}_i)$ and $\mathsf{tr}(n_j)$. In the former sub-case, we have $\{\overline{n}_i/n\} \in \sigma_1$ and $\{n_j/n\} \in \sigma_1$, where $\sigma_1 = \mathsf{index}(\widetilde{u})$. Further, by this and Definition 5.3.27 we know that $i = j$. Now, we consider the later case. By assumption that $P_1\{\tilde{u}/\tilde{x}\}$ is well-typed, we know there Γ_1, Λ_1, and Δ_1 such that $\Gamma_1; \Lambda_1; \Delta_1 \vdash P_1\{\tilde{u}/\tilde{x}\} \triangleright \diamond$ with $\mathsf{balanced}(\Delta_1)$, Thus, we have $n : S \in \Delta_1$ and $\overline{n} : T \in \Delta_1$ such that S dual T.

Hence, we can infer the following transition:

$$\langle\mathtt{Tau}\rangle\ \frac{R' \xrightarrow{\breve{\ell}_1} Q'_2 \qquad R'' \xrightarrow{\breve{\ell}_2} Q''_2 \qquad \breve{\ell}_1 \asymp \breve{\ell}_2}{(R' \mid R'') \xrightarrow{\tau} (\nu\,\widetilde{m}_2)\,(Q'_2 \mid Q''_2)}$$

Now, we should show that

$$(\nu\,\widetilde{m}_1)\,(P'_2\rho'_2 \mid P''_2\rho''_2) \ \mathcal{S} \ (\nu\,\widetilde{m}_2)\,(Q'_2 \mid Q''_2) \tag{B.165}$$

Now, by (B.160), (B.164), and $(P'_2 \mid P''_2)\rho'_2 \cdot \rho''_2 = P'_2\rho'_2 \mid P'_2\rho''_2$ we have

$$Q'_2 \mid Q''_2 \in \widehat{\mathcal{C}}^{\rho'_2\cdot\rho''_2}_{\sigma_1\cdot\mathsf{next}(n_i)\cdot\mathsf{next}(\overline{n}_i)}(P'_2 \mid P''_2)$$

Further, we have $\sigma_1 \cdot \mathsf{next}(n_i) \cdot \mathsf{next}(\overline{n}_i) \in \mathsf{index}(\mathtt{fn}(P'_2 \mid P''_2) \cup \rho'_2 \cup \rho''_2)$ This follow directly by the definition of σ_1, that is $\sigma_1 \in \mathsf{index}(\mathtt{fn}(P_1) \cup \widetilde{u} \cup \widetilde{x})$, and Definition 5.3.27.

Finally, by this and the definition of $\mathcal{S}$ (Definition 5.3.28) the goal (B.165) follows. This concludes $\mathtt{Tau}$ case.

Recursion cases Now, we consider cases where $P_1' \equiv \mu X.P_1^*$ is a parallel component of P_1. We focus on cases that highlight specifics of the breakdown of recursive processes and omit details that are similar to the corresponding non-recursive cases.

1. Case `Rv`. Here we know $P_1 = n?(y).P_2$. Further, we know there exist P^* such that $\mathcal{D}_X(P, P^*, d)$ (Definition 5.3.25). Now, we unfold this definition. Let

$$P_1^1 = \alpha_d.\alpha_{d-1}.\dots.\alpha_1.(X \mid R)$$

where R is some processes, and $\alpha_d = n?(y)$. We know that there is $\mu X.P^*$ such that P_1^1 is its sub-processes and

$$P_1 \equiv P_1^1\{^{\mu X.P^*}/X\}$$

Here we can distinguish two sub-cases: (i) $p > 0$ and (ii) $p = 0$ and $R \equiv \mathbf{0}$. We know there is $\rho_2 = \{^{\tilde{u}_2}/\tilde{x}_2\}$ such that

$$P_1\rho_1 = n\rho_1?(y).P_2\rho_2$$

The transition is as follows:

$$\langle \texttt{Rv} \rangle \frac{}{P_1^1\{^{\mu X.P^*}/X\} \xrightarrow{n?(v)} P_2\rho_2 \cdot \{^v/y\}}$$

Let $\sigma_1 = \sigma' \cdot \{^{n_i}/n\}$ where $\sigma' \in \mathsf{index}(\mathtt{fn}(P_1) \cup \tilde{u} \cup \tilde{x})$ and $\sigma_2 = \sigma_1 \cdot \sigma$ where $\sigma = \mathsf{next}(n_i) \cdot \{^{y_1}/y\}$. Further, let $\widetilde{u_*} = \mathsf{bn}(\tilde{u}\sigma_1 : \widetilde{C})$ and $\widetilde{x_*} = \mathsf{bn}(\tilde{x}\sigma_1 : \widetilde{C})$ with $\tilde{u} : \widetilde{C}$. Also, let $\widetilde{y} = (y_1, \dots, y_{|\mathcal{H}^*(S)|})$ and $\widetilde{z} = \mathtt{fnb}(P_2, \widetilde{x_*}\widetilde{y} \setminus \widetilde{w})$ where $\widetilde{w} = \{n_i\}$ if $\mathtt{lin}(n_i)$ otherwise $\widetilde{w} = \epsilon$.

We could see that $d = \delta(\alpha_d, \alpha_{d-1}, \dots, \alpha_1.\mu X.P_1^*)$. So, by Table 5.7 there are following possibilities for the shape of Q_1, namely elements in N defined as:

$$N = \{(\nu\, \tilde{c})\, R : R \in \widehat{\mathcal{C}}^{\tilde{u_*}}_{\tilde{x_*},\rho}(\mu X.P^*)^d\}$$

Let

$$Q_1^1 = (\nu\, \tilde{c})\, B_1 \mid n_l?(\widetilde{y}).\overline{c^r_{k+1}}!\langle \widetilde{z}\rho\rangle.\mu X.c^r_k?(\widetilde{x}).n_l?(\widetilde{y}).\overline{c^r_{k+1}}!\langle \widetilde{z}\rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(\alpha_d, \alpha_{d-1}.P_1^2\sigma)_g$$

where B_1 is, intuitively, trios mimicking prefixes before α_d and P_1^2 is such that $P_1^1 = \alpha_d.\alpha_{d-1}.P_1^2$. By unfolding the definition of $\widehat{\mathcal{C}}^{\tilde{u_*}}_{\tilde{x_*},\rho}(\mu X.P^*)^d$ we have that $Q_1^1 \in N$. Further, if $R \in \widehat{\mathcal{C}}^{\tilde{u_*}}_{\tilde{x_*},\rho}(\mu X.P^*)^d$ then $R \overset{\tau}{\Rightarrow} Q_1^1$. So, we only analyze how Q_1^1 evolves. We can infer the following:

$$Q_1^1 \xrightarrow{n_i\rho?\langle \widetilde{v}\rangle} Q_2$$

where

$$Q_2 = B_1 \mid \overline{c^r_{k+1}}!\langle \widetilde{z}\rho \cdot \{^{\tilde{v}}/\tilde{y}\}\rangle.\mu X.c^r_k?(\widetilde{x}).n_l?(\widetilde{y}).\overline{c^r_{k+1}}!\langle \widetilde{z}\rangle.X \mid \widehat{\mathcal{A}}^{k+1}_{\widetilde{z}}(\alpha_{d-1}.P_1^2\sigma)_g$$

with $v\sigma_v \diamond \tilde{v}$ for some $\sigma_v \in \mathsf{index}(v)$. Now, we should show that

$$P_2\rho_2 \cdot \{^v/y\}\, \mathcal{S}\, Q_2 \tag{B.166}$$

Let $\widetilde{u}_z = \tilde{z}\rho_2 \cdot \{\tilde{v}/\tilde{y}\}$. In sub-case (i) we should show that $Q_2 \in \mathcal{C}^{\widetilde{u}_z}_{\tilde{z}}(\alpha_{d-1}.P_1^2)$. So, we should show that

$$Q_2 \in \widehat{\mathcal{C}}^{\widetilde{u}_z}_{\tilde{z},\rho_2\cdot\{\tilde{v}/\tilde{y}\}}(P^*)^{d-1}_g$$

This follows by inspecting the definition $\widehat{\mathfrak{J}}^{k}_{\tilde{z},\rho_2\cdot\{\tilde{v}/\tilde{y}\}}(P^*)^{d-1}_g$. That is, we can notice that

$$Q_2^1 = B_1 \mid B_2$$

where $B_2 \in \widehat{\mathfrak{J}}^{k}_{\tilde{z},\rho_2\cdot\{\tilde{v}/\tilde{y}\}}(\alpha_{d-1}.P_1^2\sigma)^{d-1}_g$ as $(d-1)+1 = \delta(\alpha_d.\alpha_{d-1}.P_1^2\sigma)$. So, (B.166) follows. This concludes sub-case (i).

In sub-case (ii) we know $P_2 \equiv \alpha_{d-1}.P_1^2 \mid R$. Now, by Table 5.7 we have $Q_2 \xrightarrow{\tau} Q_2^2$ where

$$Q_2^2 = (\nu\,\widetilde{c})\,V_1 \mid c^r_{k+l+1}!\langle\widetilde{u}_{z_2}\rangle \mid V_2$$

where

$$V_1 = B_1 \mid \mu X.c^r_k?(\widetilde{x}).n_l?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X \mid \overline{c^r_{k+1}}!\langle\widetilde{u}_{z_1}\rangle.\mu X.c^r_k?(\widetilde{x}).(\overline{c^r_{k+1}}!\langle\widetilde{z}_1\rangle.X \mid c^r_{k+l+1}!\langle\widetilde{z}_2\rangle)$$
$$V_2 = \widehat{\mathcal{A}}^{k+1}_{\tilde{z}_1}(P_X)_{g_1} \mid \widehat{\mathcal{A}}^{k+l+1}_{\tilde{z}_2}(R)_\emptyset$$

Now, we should show that $Q_2^2 \in \mathcal{C}^{\widetilde{u}_z}_{\tilde{z}}(\alpha_{d-1}.P_1^2 \mid R)$. By Table 5.7 we have that

$$\{R_1 \mid R_2 : R_1 \in \widehat{\mathcal{C}}^{\widetilde{u}_{z_1}}_{\tilde{z}_1}(\alpha_{d-1}.P_1^2),\ R_2 \in \mathcal{C}^{\widetilde{u}_{z_2}}_{\tilde{z}_2}(R)\} \subseteq \mathcal{C}^{\widetilde{u}_z}_{\tilde{z}}(\alpha_{d-1}.P_1^2 \mid R) \tag{B.167}$$

Further, we know that

$$(\nu\,\widetilde{c}_{k+l+1})\,c^r_{k+l+1}!\langle\widetilde{u}_z\rangle \mid \widehat{\mathcal{A}}^{k+l+1}_{\tilde{z}_2}(R)_\emptyset \in \mathcal{C}^{\widetilde{u}_{z_2}}_{\tilde{z}_2}(R) \tag{B.168}$$
$$V_1 \mid V_2 \in \widehat{\mathcal{C}}^{\widetilde{u}_{z_1}}_{\tilde{z}_1}(\alpha_{d-1}.P_1^2) \tag{B.169}$$

Thus, by (B.167), (B.168), and (B.169) we have the following:

$$V_1 \mid V_2 \mid (\nu\,\widetilde{c}_{k+l+1})\,c^r_{k+l+1}!\langle\widetilde{u}_z\rangle \mid \widehat{\mathcal{A}}^{k+l+1}_{\tilde{z}_2}(R)_\emptyset \in \mathcal{C}^{\widetilde{u}_z}_{\tilde{z}}(\alpha_{d-1}.P_1^2 \mid R)$$

Now, we can notice that $\widetilde{c}_{k+l+1} \cap \mathsf{fpn}(V_1) = \emptyset$ and $\widetilde{c}_{k+l+1} \cap \mathsf{fpn}(\widehat{\mathcal{A}}^{k+1}_{\tilde{z}_1}(P_X)_{g_1}) = \emptyset$. Further, we have

$$(\nu\,\widetilde{c}_{k+l+1})\,\widehat{\mathbf{A}}^{k+l+1}_{\tilde{z}_2}(R)_\emptyset \approx^{\mathsf{C}} \mathbf{0}$$

(where $\approx^{\mathsf{C}}$ is as in Definition 5.3.14) as first shared trigger c_{k+l+1} in the breakdown of R is restricted so it could not get activated.

Thus, we have

$$Q_2^2 \equiv V_1 \mid V_2 \mid (\nu\,\widetilde{c}_{k+l+1})\,c^r_{k+l+1}!\langle\widetilde{u}_z\rangle \mid \widehat{\mathcal{A}}^{k+l+1}_{\tilde{z}_2}(R)_\emptyset$$

This concludes `Rv` case. Now, we consider the inductive case.

2. Case $\mathtt{Par_L}$. Here we know $P_1 = P_1' \mid P_1''$. Further, we know there exist P^* such that $\mathcal{D}_X(P, P^*, d)$ (Definition 5.3.25). Similarly to the previous case, let

$$P_1^1 = \alpha_d.\alpha_{d-1}.\ldots.\alpha_1.(X \mid R)$$

We know there is $\mu X.P^*$ such that P_1^1 is its subprocess and

$$P_1' \equiv P_1^1\{{}^{\mu X.P^*}/X\}$$

Here, we can distinguish two sub-cases: (i) $P_1'' \equiv R$ and (ii) $P_1'' \not\equiv R$. Here, we consider sub-case (i) as it is an interesting case. The sub-case (ii) is similar to the corresponding case of non-recursive process. As in the previous case, we can further distinguish cases in which $p = 0$ and $p > 0$. We consider $p = 0$ and $\ell = n?\langle v\rangle$.

The final rule in the inference tree is as follows:

$$\langle \mathtt{Par}_L\rangle\ \frac{P_1'\{{}^{\tilde{u}_1'}/\tilde{y}_1\} \xrightarrow{\ell} P_2'\{{}^{\tilde{u}_2'}/\tilde{y}_2\} \mid R\{{}^{\tilde{w}_1}/\tilde{z}_1\} \cdot \{{}^{\tilde{u}'_R}/\tilde{w}_1\} \qquad \mathtt{bn}(\ell) \cap \mathtt{fn}(P_1'') = \emptyset}{P_1'\{{}^{\tilde{u}_1'}/\tilde{y}_1\} \mid R\{{}^{\tilde{u}_1''}/\tilde{z}_1\} \xrightarrow{\ell} P_2'\{{}^{\tilde{u}_2'}/\tilde{y}_2\} \mid R\{{}^{\tilde{u}_1''}/\tilde{z}_1\} \mid R\{{}^{\tilde{w}_1}/\tilde{z}_1\} \cdot \{{}^{\tilde{u}'_R}/\tilde{w}_1\}}$$

Let $\sigma_1 \in \mathsf{index}(\mathtt{fn}(P_1) \cup \widetilde{u} \cup \widetilde{x})$. Further, let $\widetilde{y_{*1}} = \mathtt{fnb}(P_1', \widetilde{x_*})$, $\widetilde{z}_{*1} = \mathtt{fnb}(R, \widetilde{x_*})$, $\widetilde{u_*} = \mathsf{bn}(\widetilde{u}\sigma_1 : \widetilde{C})$ where $\widetilde{x_*} = \mathsf{bn}(\widetilde{x}\sigma_1 : \widetilde{C})$ with $\widetilde{u} : \widetilde{C}$, and $\rho_{\mathtt{m}} = \{{}^{\tilde{u}_*}/\tilde{x}_*\}$. By the definition of $\mathcal{S}$ (Table 5.7), there are following possibilities for Q_1:

$$\begin{aligned}
Q_1^1 =& (\nu\, \widetilde{c}_k)\, (\overline{c_k}!\langle\widetilde{u_*}\rangle \mid \mathcal{A}^k_{\tilde{x}_*}(P_1\sigma_1))\\
Q_1^2 =& (\nu\, \widetilde{c}_k)\, \overline{c_k}!\langle\widetilde{u_{*2}'}\rangle.\overline{c_{k+l}}!\langle\widetilde{u_{*1}''}\rangle \mid \mathcal{A}^k_{\tilde{y}_{*1}}(P_1'\sigma_1) \mid \mathcal{A}^{k+l}_{\tilde{z}_{*1}}(R\sigma_1)\\
N_1^3 =& \{(R_1' \mid R_1'') : \widehat{\mathcal{C}}^{\tilde{u}_{*2}'}_{\tilde{y}_{*1}}(\mu X.P^*\sigma_1)^d_g,\ R_1'' \in \mathcal{C}^{\tilde{u}_{*1}''}_{\tilde{z}_{*1}}(R\sigma_1)\}
\end{aligned}$$

By Lemma 5.3.6 there exist

$$Q_1' \in \widehat{\mathcal{C}}^{\tilde{u}_{*2}'}_{\tilde{y}_{*1}}(\mu X.P^*\sigma_1)^d_g \tag{B.170}$$

$$Q_1'' \in \mathcal{C}^{\tilde{u}_{*1}''}_{\tilde{z}_{*1}}(R\sigma_1) \tag{B.171}$$

such that

$$Q_1^1 \overset{\tau}{\Rightarrow} Q_1^2 \overset{\tau}{\Rightarrow} Q_1' \mid Q_1''$$

The interesting case is to consider a process Q_1^3 defined as:

$$Q_1^3 = (\nu\, \widetilde{c})\, V_1 \mid c^r_{k+l+1}!\langle\tilde{u}_{*1}''\rangle \mid V_2$$

where

$$\begin{aligned}
V_1 &= B_1 \mid \overline{c^r_{k+1}}!\langle\widetilde{u_{*2}'}\rangle.\mu X.c^r_k?(\widetilde{y_{*1}}).(\overline{c^r_{k+1}}!\langle\widetilde{y_{*2}}\rangle.X \mid c^r_{k+l+1}!\langle\widetilde{z}_{*2}\rangle)\\
V_2 &= \widehat{\mathcal{A}}^{k+1}_{\tilde{y}_{*2}}(P_X)_{g_1} \mid \widehat{\mathcal{A}}^{k+l+1}_{\tilde{w}_{*2}}(R\{{}^{\tilde{w}_2}/\tilde{z}_2\})_{\emptyset}
\end{aligned}$$

We have $Q_1^3 \approx^{\mathtt{C}} Q_1' \mid Q_1''$ (where $\approx^{\mathtt{C}}$ is as in Definition 5.3.14). We may notice that Q_1^3 can be a descendent of the recursive process (following the similar reasoning as in the previous case). So, we consider how Q_1^3 evolves. As in the corresponding case of non-recursive processes, we do the case analysis on ℓ. If $v \in \widetilde{u}$ then we take $\sigma_v = \sigma_1$, otherwise $\sigma_v = \{{}^{v_j}/v\}$ for $j > 0$. Now, we could see that

$$Q_1^3 \xRightarrow{n?\langle\tilde{v}\rangle} Q_2$$

where

$$Q_2 = (\nu\, \widetilde{c})\, V_1 \mid \overline{c^r_{k+1}}!\langle\widetilde{u_{*2}'}\rangle.\mu X.c^r_k?(\widetilde{y_{*1}}).(\overline{c^r_{k+1}}!\langle\widetilde{y_{*2}}\rangle.X \mid c^r_{k+l+1}!\langle\widetilde{z}_{*1}\rangle) \mid c^r_{k+l+1}!\langle\widetilde{u_{*R}'}\rangle \mid V_2$$

We define Q_2^1 as follows

$$Q_2^1 = (\nu\,\widetilde{c})\,V_1 \mid \overline{c^r_{k+1}}!\langle\widetilde{u_*{}'_2}\rangle.\mu X.c^r_k?(\widetilde{y_{*1}}).(\overline{c^r_{k+1}}!\langle\widetilde{y_{*2}}\rangle.X \mid c^r_{k+l+1}!\langle\widetilde{z}_{*1}\rangle) \mid V_2 \mid \\ (\nu\,\widetilde{c}_k)\,(c^r_k!\langle\widetilde{u_*{}''_1}\rangle \mid \widehat{\mathcal{A}}^{k}_{\tilde{z}_{*1}}(R)_\emptyset) \mid (\nu\,\widetilde{c}_k)\,(c^r_k!\langle\widetilde{u_*{}'_R}\rangle \mid \widehat{\mathcal{A}}^{k}_{\tilde{w}_{*1}}(R\{\tilde{w}_1/\tilde{z}_1\})_\emptyset)$$

By definition, we could see that $Q_2^1 \in \mathcal{C}^{\tilde{u}_{*2}}_{\tilde{z}_{*1}}(P_2' \mid R \mid R\{\tilde{w}_1/\tilde{z}_1\})$. Now, by the definition of $\widehat{\mathcal{A}}^{-}_{-}(\,\cdot\,)_\emptyset$ we have

$$c^r_{k+l+1}!\langle\widetilde{u_*{}''_1}\rangle \mid c^r_{k+l+1}!\langle\widetilde{u_*{}'_R}\rangle \mid \widehat{\mathcal{A}}^{k+l+1}_{\tilde{z}_{*1}}(R)_\emptyset \approx^{\mathtt{C}} \\ (\nu\,\widetilde{c}_k)\,(c^r_k!\langle\widetilde{u_*{}''_1}\rangle \mid \widehat{\mathcal{A}}^{k}_{\tilde{z}_{*1}}(R)_\emptyset) \mid (\nu\,\widetilde{c}_k)\,(c^r_k!\langle\widetilde{u_*{}'_R}\rangle \mid \widehat{\mathcal{A}}^{k}_{\tilde{w}_{*1}}(R\{\tilde{w}_1/\tilde{z}_1\})_\emptyset$$

As each trio in $\widehat{\mathcal{A}}^{k+l+1}_{\tilde{z}_{*1}}(R)_\emptyset$ makes a replica of itself when triggered along a propagator. So, finally we have

$$Q_2 \approx^{\mathtt{C}} Q_2^1$$

This concludes the proof of Lemma 5.3.7.

□

B.3.7 Proof of Lemma 5.3.9

Lemma 5.3.9. *Assume $P_1\{\tilde{u}/\tilde{x}\}$ is a well-formed process and $P_1\{\tilde{u}/\tilde{x}\}\,\mathcal{S}\,Q_1$.*

1. *Whenever $Q_1 \xrightarrow{(\nu\,\widetilde{m_2})\,n_i!\langle\widetilde{v}:\mathcal{H}^*(C_1)\rangle} Q_2$, such that $\overline{n}_i \notin \mathtt{fn}(Q_1)$, then there exist P_2 and σ_v such that $P_1\{\tilde{u}/\tilde{x}\} \xrightarrow{(\nu\,\widetilde{m_1})\,n!\langle v:C_1\rangle} P_2$ where $v\sigma_v \diamond \widetilde{v}$ where $v\sigma_v \diamond \widetilde{v}$ and, for a fresh t,*

$$(\nu\,\widetilde{m_1})(P_2 \parallel t \Leftarrow_{\mathtt{C}} v : C_1)\ \ \mathcal{S}\ \ (\nu\,\widetilde{m_2})(Q_2 \parallel t_1 \Leftarrow_{\mathtt{m}} v\sigma_v : C_1)$$

2. *Whenever $Q_1 \xrightarrow{\breve{n}?(\tilde{v})} Q_2$ then there exist P_2 and σ_v such that $P_1\{\tilde{u}/\tilde{x}\} \xrightarrow{n?(v)} P_2$ where $v\sigma_v \diamond \widetilde{v}$ and $P_2\ \mathcal{S}\ Q_2$,*

3. *Whenever $Q_1 \xrightarrow{\tau} Q_2$ either (i) $P_1\{\tilde{u}/\tilde{x}\}\ \mathcal{S}\ Q_2$ or (ii) there exists P_2 such that $P_1 \xrightarrow{\tau} P_2$ and $P_2\ \mathcal{S}\ Q_2$.*

Proof (Sketch). Following Parrow, we refer to prefixes corresponding to prefixes of the original process as *essential prefixes*. We remark that a prefix in $\mathcal{D}(P)$ is non-essential if and only if is a prefix on a propagator name. First, we discuss case when transition is inferred without any actions from essential prefixes. In this case we know that an action can only involve propagator prefixes and by inspection of definition of $\mathcal{C}^{\tilde{u}}_{\tilde{x}}(P_1)$ that $\ell = \tau$. This concerns the sub-case (i) of Part 3 and it directly follows by Lemma 5.3.6.

Now, assume $Q_1 \xrightarrow{\ell} Q_2$ when ℓ involves essential prefixes. This concerns Part 1, Part 2, and sub-case (ii) of Part 3. This case is mainly the inverse of the proof of Lemma 5.3.7. As Parrow, here we note that an essential prefix is unguarded in Q_1 if and only if it is unguarded in P_1. That is, by inspection of the definition, function $\mathcal{C}^{\tilde{u}}_{\tilde{x}}(P_1)$ does not unguard essential prefixes of P_1 that its members mimic (the propagators serve as guards). □