

Orderly Communication in the Ambient Calculus

January 29, 2002

Torben Amtoft
Boston University
www.cs.bu.edu/associates/tamtoft

Assaf J. Kfoury
Boston University
www.cs.bu.edu/~kfoury

Santiago M. Pericas-Geertsen
Boston University
cs-people.bu.edu/santiago

Abstract

*The Ambient Calculus (henceforth, **AC**) was developed by Cardelli and Gordon as a formal framework to study issues of mobility and migrant code [9]. We present a type system for **AC** that allows the type of exchanged data within the same ambient to vary over time. Our type system assigns what we call behaviors to processes; a denotational semantics of behaviors is proposed, here called trace semantics, underlying much of the remaining analysis. We state and prove a Subject Reduction property for our typed version of **AC**. Based on techniques borrowed from finite automata theory, type checking of fully type-annotated processes is shown to be decidable. We show that the typed version of **AC** originally proposed by Cardelli and Gordon [10] can be naturally embedded into our typed version of **AC**.*

1 Introduction

1.1 Background and Motivation

Current technology in computer networking makes it possible to view computers as parts of a global computation platform, sharing their software and data resources. The possibility of sharing resources across a wide-area network has given rise to a new paradigm, the *mobile computation* paradigm, combining aspects of concurrent and distributed programming. Programmers are given the means to create and deploy mobile agents, which can move across the network and use local resources of the sites they visit.

One purpose for designing a type system for a programming language is to provide programmers with various *safety* and *security* guarantees. A safe program means a program that does not corrupt the run-time system, whereby its execution produces behavior and results conforming to its intended semantics; in particular, safety is assurance against accidental mismatch of operators and operands. Safety is arguably a precondition for security, which deals with a broader range of issues related to secrecy, integrity and prevention of malevolent attacks. A program that conforms to its intended semantics can be further required to satisfy more stringent security requirements against external threats. Type systems have been shown to be powerful means to eliminate safety breaches and security bugs proactively before they do any damage.

Our long-time interest is the design and implementation of a strongly-typed language for mobile computation. The starting point of our investigation is the *Ambient Calculus* (henceforth, **AC**) of Cardelli and Gordon [9]. **AC** is one of several foundational calculi that have been proposed in the literature for the study of mobile computation (see, among others, [12], [17, 18], [19], [21]). Type systems for each of these calculi have also been proposed to enforce various safety and security properties. Our choice of **AC** is partly motivated by its concise formulation, and the relative simplicity and elegance of the type systems devised for it by Cardelli, Ghelli, and Gordon.

The type system we propose in this paper for **AC** can be viewed as an extension of Cardelli’s and Gordon’s original type system for **AC** and some of its later modifications (see [10, 7, 8]), obtained by incorporating a mechanism to enforce “safe communication between ambients” (in a sense to be made precise). Our idea is that an ambient should not be forced to enclose a single type of communication during the whole of its lifetime. Instead, from our perspective, the type of exchanged messages inside an ambient (what Cardelli and Gordon have aptly called the ambient’s “topic of conversation”) is allowed to change over time; this time-dependent nature of an ambient’s topic of conversation is recorded by a notion of “behavior” (the syntax of which is defined simultaneously with types), and the interpretation of a behavior is a set of “traces”. Precise definitions of these notions are in the rest of the paper.

Here is a small example to explain our idea a little more. That the type of exchanged data can *safely* change over time, as the computation proceeds inside an ambient, is illustrated by the following process:

$$m[\langle 7 \rangle \mid (x).\text{open } n.\langle x = 42 \rangle \mid n[(y).P]]$$

where the type of the equality test “ $x = 42$ ” is `bool`. Initially, the topic of conversation in the ambient m is `int`. After the output $\langle 7 \rangle$ is received by the input variable x , we obtain the process:

$$m[\text{open } n.\langle 7 = 42 \rangle \mid n[(y).P]]$$

at which point the ambient n is opened, resulting in the process

$$m[\langle 7 = 42 \rangle \mid (y).P]$$

and now the topic of conversation in the ambient m becomes `bool`. Assuming that the unspecified process $(y).P$ can be executed safely whenever it inputs a boolean value, the execution of the entire process enclosed in the ambient m can proceed safely. What takes place in the ambient m is a case of what we call *orderly communication*¹. By contrast, in the original type system proposed by Cardelli and Gordon [10] and in its later variations, the topic of conversation in an ambient remains fixed throughout its lifetime.

Returning to the question of safety versus security, we note that orderly communication as formulated in the present paper is limited to certify the safe behavior of a wider class of processes. It does not yet address security issues as normally understood, which are chiefly about protection against external willful threats. This will require further extensions of our type system, in future work [1], where the behavior of a process (in the precise technical sense of this paper) will be an annotation that also keeps track of movements of ambients.

1.2 Contribution of Our Research

We denote by **AC*** our typed version of the Ambient Calculus. In summary our main accomplishments in the present paper are:

- We design a type system for **AC*** where exchanged messages are assigned types and processes are assigned what we call *behaviors*. Behaviors are intended to model our notion of orderly communication between mobile ambients.
- We develop a straightforward and easy-to-understand denotational semantics of behaviors, which we call their *trace semantics*. Behavior equivalence and behavior subsumption are defined relative to this trace semantics².

¹We thank Benjamin Pierce for suggesting the apt expression “orderly communication”.

²We are knowingly overloading words that are extensively used, not always with the same meaning, in the literature on concurrency. Such are the words “behavior” and “trace”. Although our use is still different (alas!), these words are also suggestive and appropriately describe the formal notions they refer to in our work.

- Behavior subsumption is shown to be a decidable relation. The deterministic time-complexity of our decision procedure is at least exponential³.

The proof of this last result is of independent interest; it is an adaptation of techniques from finite automata theory where, by contrast, decision procedures typically have low-degree polynomial time complexities.

- Using the trace semantics of behaviors, we prove that our typed calculus \mathbf{AC}^* satisfies a Subject Reduction property.
- Employing the decidability of behavior subsumption, we show that *type-checking* is decidable for fully type-annotated terms of \mathbf{AC}^* . The deterministic time-complexity of the decision procedure is at least exponential in the worst-case.

The more difficult problem of *type-inference* for (un-annotated) terms of \mathbf{AC}^* is left for future work.

- There is a natural embedding of the typed version of \mathbf{AC} originally proposed by Cardelli and Gordon [10] into our typed calculus \mathbf{AC}^* , in the sense that every process typable in the former is easily translated into a process typable in our system (but not the other way around).

Orderly communication bears a strong resemblance to what has been called “session types” in the π -calculus [13]. Leaving aside differences between the underlying calculi, orderly communication and session types are both motivated by the need to keep track of the order in which communication events take place. There are nevertheless important differences between the two, which are discussed further in Sect. 8.3.

Orderly communication can also be viewed as a way of polymorphically typing ambients, in the sense that the same ambient is allowed to hold different topics of conversation, consecutively at different times. A broader view of polymorphically typed ambients, which includes orderly communication as well as other cases of polymorphism, is considered in our technical report [2] (downloadable from the Church Project web site at <http://types.bu.edu/reports/>).

2 Motivating Example

We give a short example to illustrate the expressive power and convenience of orderly communication. The reader is referred to Example 5.5 for an explanation of how to type this example. The syntax of ambients is identical to that first proposed by Cardelli and Gordon [9] with the addition of a co-capability “*coopen* n ”, also employed⁴ in, e.g., [11, 3, 14]. This addition is motivated by the fact that the opening capability is problematic for developing a precise analysis, as seen by the reduction rule

$$\text{open } n.P \mid n[Q] \rightarrow P \mid Q$$

which shows that *after* opening n , the unleashed process Q will be able to communicate with the opening process P . Therefore we essentially need to split⁵ the work of Q into two parts: what is done before n is opened (not influencing P), and what is done after n is opened (influencing P). This is facilitated by the *coopen* n construct, which enables an ambient to specify exactly at which point of its computation it will allow itself to be dissolved (cf. (Red Open) in Fig. 2). We shall use $n\{P\}$ as an abbreviation, namely

$$n\{P\} \triangleq n[\text{coopen } n \mid P]$$

for every ambient name n and every process P . Thus, if we write $n\{P\}$, we mean that the ambient n is *openable* at all times.

³This result in itself is no reason for discomfiture. Hope for efficiency in practice is supported by other similar situations. For example, ML type-inference shows an extreme disparity between worst-case performance in theory (exponential time) and actual performance in practice (very fast).

⁴General co-capabilities, including also “*coin* n ” and “*coout* n ”, were proposed by Levi and Sangiorgi in [16] and have been used, e.g., also in [4]. These are irrelevant, however, for the present paper, as our interest is in tracking communication and not in tracking ambient interferences.

⁵In [7], where the focus is on tracking ambient movements, the point is made that conservatively assuming that all actions of Q *might* take place *after* n is opened will not allow one to declare immobile an ambient that opens an entering (hence mobile) ambient.

<p>Expressions</p> $M \in \text{Exp} ::= n \mid c \mid \text{in } M \mid \text{out } M \mid \text{open } M \mid \text{coopen } M \mid \epsilon \mid M_1.M_2$
<p>Processes</p> $P \in \text{Proc} ::= \mathbf{0} \mid P_1 \mid P_2 \mid !P \mid (\nu n : \tau).P \mid M.P \mid M[P] \mid (n_1 : \tau_1, \dots, n_k : \tau_k).P \mid \langle M_1, \dots, M_k \rangle \quad (k \geq 0)$

Figure 1. Syntax of AC^* .

EXAMPLE 2.1 (SOURCE ROUTING). A packet enters a router and requests to be routed to a specific destination. A router reads the destination name (denoted by the string “bu”) and then communicates a path (a sequence of in and out capabilities) back to the packet. The packet uses this path to route itself to the destination. Orderly communication is needed since inside the packet there are *two* topics of conversation: first strings (the destination “bu”), and next capabilities (the path)⁶.

$$\begin{aligned} & \text{router}[\text{!route}\{\text{in } \text{packet}.\langle \text{dst} \rangle.\text{open } \text{hop}.\langle \text{lookup-route}(\text{dst}) \rangle\}] \mid \\ & \text{packet}[\text{in } \text{router}.\text{open } \text{route}.\langle \text{“bu”} \rangle \mid \text{hop}\{(x).x\}] \end{aligned}$$

Notice that the packet reads and exercises the path by means of its subterm $(x).x$. Despite its simplicity, the term $(x).x$ is not typable in the Cardelli-Gordon type system for AC nor, to the best of our knowledge, in any of the type systems for AC available in the literature. At first, it appears that a type derivation for $(x).x$ consists of an instance of the rule (Exp n) followed by (Proc Input) [10]. However, this is not the case since $(x).x$ is a shorthand for $(x).x.\mathbf{0}$. A close examination of the type derivation for $(x).x.\mathbf{0}$ reveals that x requires a type T such that $T = \text{cap}[T]$, but no such type exists in the Cardelli-Gordon system. In that system, the only way to type a process that reads and exercises a capability is by using an extra ambient. Specifically, the process $(x).x$ must be written as $(x).n[x]$ for some ambient name n . \square

3 The Calculus

Figure 1 shows the syntax of our language AC^* . A process $P \in \text{Proc}$ is basically as in [10]: there are constructs for parallel composition ($P_1 \mid P_2$), replication ($!P$), restriction ($(\nu n : \tau).P$); and there also are constructs for input (where the names $n_1 \dots n_k$ must be distinct) and output. Note that communication is asynchronous, in that an output process has no “continuation”; a communication can thus (cf. the metaphor in [6]) be viewed as the placement, and subsequent removal, of a Post-It note on a message board. On the other hand we believe that our development would carry through, with the obvious modifications, for an extension of AC^* in which synchronous communication is allowed.

An expression $M \in \text{Exp}$ is either a constant c (such as an integer or a boolean), an ambient name n (we shall also use the letter m), a capability (such as $\text{in } M$ or $\text{open } M$), or a path of capabilities. We assume that the set of constants ranged by c is open-ended. Note that in the binding constructs $(\nu n : \tau).P$ and $(n_1 : \tau_1, \dots, n_k : \tau_k).P$, the names being bound are annotated with a type (to be defined in Sect. 4).

The set of names occurring free in P is denoted $\text{fn}(P)$; the set of all names occurring in P is denoted $\text{names}(P)$. We say that a process P is non-conflicting with a set of names X if (i) no name is bound more than once in P , and (ii) a name bound in P does not occur in X . For all P and X , we can clearly find P' such that P' is non-conflicting with X and such that P' and P are equal modulo consistent renaming of bound names.

⁶For the purpose of this example, we use an extended syntax (cf. Section 3) in which we allow function applications as part of the syntax of expressions. We assume that the function `lookup-route` takes a string as input and produces a capability path as output.

Reduction Labels	
$\ell ::= \epsilon$	no visible communications
$ \text{comm}(\times(\tau_1, \dots, \tau_k))$	a tuple with values of type $\tau_1 \dots \tau_k$ is communicated at top-level
Reduction Rules	
In (Red Comm) we demand that $(n_1 : \tau_1, \dots, n_k : \tau_k).P$ is non-conflicting with $\text{names}(M_1, \dots, M_k)$.	
$n[\text{in } m.P \mid Q] \mid m[R] \xrightarrow{\epsilon} m[n[P \mid Q] \mid R]$	(Red In)
$m[n[\text{out } m.P \mid Q] \mid R] \xrightarrow{\epsilon} n[P \mid Q] \mid m[R]$	(Red Out)
$\text{open } n.P \mid n[\text{coopen } n.Q \mid R] \xrightarrow{\epsilon} P \mid Q \mid R$	(Red Open)
$(n_1 : \tau_1, \dots, n_k : \tau_k).P \mid \langle M_1, \dots, M_k \rangle \xrightarrow{\text{comm}(\sigma)} P[n_i := M_i]$ where $\sigma = \times(\tau_1, \dots, \tau_k)$	(Red Comm)
$!P \xrightarrow{\epsilon} P \mid !P$	(Red Repl)
$\text{If } P \xrightarrow{\ell} Q \text{ then } P \mid R \xrightarrow{\ell} Q \mid R$	(Red Par)
$\text{If } P \xrightarrow{\ell} Q \text{ then } (\nu n : \tau).P \xrightarrow{\ell} (\nu n : \tau).Q$	(Red Res)
$\text{If } P \xrightarrow{\ell} Q \text{ then } n[P] \xrightarrow{\epsilon} n[Q]$	(Red Amb)
$\text{If } P' \equiv P, P \xrightarrow{\ell} Q, Q \equiv Q' \text{ then } P' \xrightarrow{\ell} Q'$	(Red \equiv)

Figure 2. Operational Semantics.

3.1 Operational Semantics

We now present a small-step semantics for AC^* . The relation $P_1 \xrightarrow{\ell} P_2$, with the interpretation that P_1 reduces in one step to P_2 by performing an action described by the reduction label ℓ , is defined as the least one satisfying the clauses given in Fig. 2. Reducing inside an ambient is given a special treatment in (Red Amb), where the reduction label “disappears” due to the fact that communications are invisible outside ambients. Note that $P \xrightarrow{\ell} Q$ does not imply that $M.P \xrightarrow{\ell} M.Q$ since M (which has to be a capability) must be executed before P can be activated.

For the rule (Red \equiv), we employ an auxiliary relation $P_1 \equiv P_2$ denoting that P_1 and P_2 are equivalent modulo consistent renaming of bound names (which may be needed to satisfy the side condition in (Red Comm)) and modulo “syntactic rearrangement”. The relation is given as the least one satisfying the clauses presented in Fig. 3, and is as in [10] except that, for reasons mentioned in Sect. 6 (the remark after Lemma 6.5), we omit the rule $!P \equiv P \mid !P$ and instead allow this “unfolding” to take place via the rule (Red Repl).

4 Types and Behaviors

The syntax of types ($\tau \in \text{Typ}$) and the syntax of behaviors ($b \in \text{Beh}$) are recursively defined in Fig. 4. The first five behavior constructs capture the intuition that we want to keep track of the relationship (sequential or parallel) between occurrences of input and output operations. (See Sect. 8.2 for a discussion of how also to keep track of ambient movements.)

An ambient n has a type of the form $\text{amb}[b]$, where b can be viewed as an upper estimate of the behavior of a process “unleashed” by opening n . For example, in the process $n[\langle 7 \rangle \mid (x : \text{int}).\text{coopen } n.\langle x = 42 \rangle]$ we expect n to have type $\text{amb}[\text{put}(\text{bool})]$, reflecting the fact that when n is opened the expression 7 has already been communicated—something we would not know if we did not have the explicit occurrence of $\text{coopen } n$, which we keep track of using the behavior diss .

A capability has a type of the form $\text{cap}[B]$ where B is a *behavior context*, that is, a behavior with a hole inside. Capabilities are exercised only when they become part of a process. Thus, there is always a

$P \equiv P$	(Struct Refl)
If $P \equiv Q$ then $Q \equiv P$	(Struct Symm)
If $P \equiv Q$ and $Q \equiv R$ then $P \equiv R$	(Struct Trans)
If $P \equiv Q$ then $(\nu n : \tau).P \equiv (\nu n : \tau).Q$	(Struct Res)
If $P \equiv Q$ then $P \mid R \equiv Q \mid R$	(Struct Par)
If $P \equiv Q$ then $!P \equiv !Q$	(Struct Repl)
If $P \equiv Q$ then $M[P] \equiv M[Q]$	(Struct Amb)
If $P \equiv Q$ then $M.P \equiv M.Q$	(Struct Action)
If $P \equiv Q$ then $(n_1 : \tau_1, \dots, n_m : \tau_m).P \equiv (n_1 : \tau_1, \dots, n_m : \tau_m).Q$	(Struct Input)
$P \mid Q \equiv Q \mid P$	(Struct ParComm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct ParAssoc)
$(\nu n_1 : \tau_1).(\nu n_2 : \tau_2).P \equiv (\nu n_2 : \tau_2).(\nu n_1 : \tau_1).P$ if $n_1 \neq n_2$	(Struct ResRes)
$(\nu n : \tau).(P \mid Q) \equiv P \mid (\nu n : \tau).Q$ if $n \notin \text{fn}(P)$	(Struct ResPar)
$(\nu n : \tau).m[P] \equiv m[(\nu n : \tau).P]$ if $n \neq m$	(Struct ResAmb)
$P \mid \mathbf{0} \equiv P$	(Struct ZeroPar)
$(\nu n : \tau).\mathbf{0} \equiv \mathbf{0}$	(Struct ZeroRes)
$!\mathbf{0} \equiv \mathbf{0}$	(Struct ZeroRepl)
$\epsilon.P \equiv P$	(Struct ϵ)
$(M.M').P \equiv M.(M'.P)$	(Struct \cdot)
$P \equiv Q$ if P and Q are equal modulo consistent renaming of bound names	(Struct α -rename)

Figure 3. Structural Congruence.

continuation that follows the exercise of a capability. The hole inside a capability of type $\text{cap}[B]$ is filled by the behavior of the corresponding continuation as shown by the rule (Proc Action) in Fig 6. For example, consider a process $P = \text{open } n.P'$ where P' has behavior b' and n has type $\text{amb}[b]$. When P is executed, P' will run in parallel with a process of behavior b , so P should be assigned the behavior $b \mid b'$, which can be written as $(b \mid \square)[b']$. This is why it makes sense to assign open n the capability type $\text{cap}[b \mid \square]$.

The first six behavior constructs in Fig. 4 alone, are sufficient to write a type system satisfying a subject reduction property (Sect. 6), but they do not enable the typing of processes performing an unbounded number of input or output operations (e.g. $!\langle 7 \rangle$) or the typing of an input name which might receive output from two sources (e.g., one emitting a capability of type $\text{cap}[\text{put}(\text{int}) \mid \square]$ and the other emitting a capability of type $\text{cap}[\text{get}(\text{int}) \mid \square]$). Among many possible options for (approximating) constructs expressing recursion and choice, in this paper we have settled for a simple one: the construct $\text{fromnow}(\sigma)$, which can be thought of as the “union” of all behaviors composed of $\text{put}(\sigma)$ and $\text{get}(\sigma)$. As to be demonstrated in Sect. 8.1, this construct actually makes it possible to embed the type system presented in [10] into our system.

4.1 Trace Semantics of Behaviors

In the subsequent sections we shall define several relations on behaviors, in particular, an ordering relation. We have taken a semantic rather than an axiomatic approach, in that we associate a set of *traces* to each behavior⁷. This is motivated by the observation that choosing the “right” set of axioms is often a somewhat ad-hoc exercise. An added advantage of the semantic approach is that in our case it considerably facilitates type checking, as illustrated by the analysis in Sect. 7.

Definition 4.1 (Traces). A trace $tr \in \text{Trace}$ is a finite sequence of actions, where an action $a \in \text{Act}$ is a behavior that is either $\text{put}(\sigma)$, $\text{get}(\sigma)$, or diss . We use Tr to range over sets of traces. \square

⁷In fact, it is possible to formulate the type system in terms of traces, as done in [1]. Still, it is convenient to be able to name certain sets of traces.

Types

$\tau \in \text{Typ} ::= \text{bool} \mid \text{int} \mid \text{string} \mid \dots$ type constant
 $\mid \text{amb}[b]$ type of ambient name
 $\mid \text{cap}[B]$ type of capability

$\sigma \in \text{Topic} = \times(\tau_1, \dots, \tau_k) \quad (k \geq 0)$

Behaviors

$b \in \text{Beh} ::= \varepsilon$ no traceable action
 $\mid b_1.b_2$ first b_1 then b_2
 $\mid b_1 \mid b_2$ parallel composition
 $\mid \text{put}(\sigma)$ output of tuple with type σ
 $\mid \text{get}(\sigma)$ input of tuple with type σ
 $\mid \text{diss}$ ambient dissolution
 $\mid \text{fromnow}(\sigma)$ unordered communication of tuple with type σ
 $\mid \dots$ other behaviors according to need,
 provided Lemmas 4.2 and 7.5 can still be established

$B \in \text{BehCont} ::= \square \mid b.B \mid B.b \mid b \mid B \mid B \mid b$

When there is no ambiguity, we write τ for $\times(\tau)$.

Sequential composition “.” binds stronger than parallel composition “|”.

$B[b]$ is the behavior resulting from replacing \square with b in B ; similarly for the behavior context $B[B_1]$.

Figure 4. Syntax of Types and Behaviors.

The semantics $\llbracket b \rrbracket$ of a behavior b belongs to the powerset $\mathcal{P}(\text{Trace})$, and is given by

$$\begin{aligned}
 \llbracket \varepsilon \rrbracket &= \{\bullet\} & \llbracket \text{diss} \rrbracket &= \{\text{diss}\} \\
 \llbracket \text{put}(\sigma) \rrbracket &= \{\text{put}(\sigma)\} & \llbracket \text{get}(\sigma) \rrbracket &= \{\text{get}(\sigma)\} \\
 \llbracket b_1.b_2 \rrbracket &= \llbracket b_1 \rrbracket \diamond \llbracket b_2 \rrbracket & \llbracket b_1 \mid b_2 \rrbracket &= \llbracket b_1 \rrbracket \parallel \llbracket b_2 \rrbracket \\
 \llbracket \text{fromnow}(\sigma) \rrbracket &= \{tr \mid \forall a \text{ occurring in } tr : a \in \{\text{put}(\sigma), \text{get}(\sigma)\}\}
 \end{aligned}$$

Here \bullet denotes the empty sequence, $tr_1 \diamond tr_2$ denotes the concatenation of tr_1 and tr_2 which trivially lifts to sets of traces, and $Tr_1 \parallel Tr_2$ denotes all traces that can be formed by arbitrarily interleaving a trace in Tr_1 with a trace in Tr_2 . Note that \diamond and \parallel are both associative operators (and the latter even commutative) on sets of traces, with $\{\bullet\}$ as neutral element.

Since a simple structural induction shows that $\llbracket b \rrbracket \neq \emptyset$ for all b , we trivially have

Lemma 4.2. *If $E \vdash P : b$ then $\llbracket b \rrbracket \neq \emptyset$.* □

EXAMPLE 4.3. Consider the behavior $b = \text{put}(\text{int}) \mid \text{get}(\text{int}).(\text{get}(\text{bool}) \mid \text{put}(\text{bool}))$. It is easy to see that $\llbracket b \rrbracket$ consists of the 8 traces depicted below:

```

put(int) get(int) put(bool) get(bool)
put(int) get(int) get(bool) put(bool)
get(int) put(int) put(bool) get(bool)
get(int) put(int) get(bool) put(bool)
  
```

```

get(int) put(bool) put(int) get(bool)
get(int) get(bool) put(int) put(bool)
get(int) put(bool) get(bool) put(int)
get(int) get(bool) put(bool) put(int)

```

□

Of the traces listed in the above example, only the first represents a “stand-alone” computation, since all other traces attempt to read something before it is written. This observation motivates the following definition:

Definition 4.4 (Comm). A trace tr belongs to Comm if tr takes the form $\text{put}(\sigma)\text{get}(\sigma')$. If in addition it holds that $\sigma = \sigma'$ we say that $tr \in \text{WtComm}$, the set of *well-typed communications*. □

EXAMPLE 4.5. Looking back on Example 4.3, only the first trace belongs to Comm^* , and even to WtComm^* . □

Note that also traces not in Comm^* are of interest, since interleaving them with other traces may produce traces in Comm^* .

4.2 Ordering on Types and Behaviors

We employ a relation $b_1 < b_2$, with the intuitive interpretation that b_2 is more “permissive” than b_1 . For example, $\text{put}(\text{int}) < \text{fromnow}(\text{int})$, since the former denotes exactly one output whereas the latter denotes an arbitrary number of inputs and outputs.

Definition 4.6 (Behavior subsumption). Let b_1 and b_2 be behaviors. Then, $b_1 < b_2$ iff $\llbracket b_1 \rrbracket \subseteq \llbracket b_2 \rrbracket$. □

Clearly, $<$ is a preorder, that is reflexive and transitive. Note that $<$ is not anti-symmetric, for instance $\text{put}(\text{int}).\text{put}(\text{int}) < \text{put}(\text{int}) \mid \text{put}(\text{int})$ and also $\text{put}(\text{int}) \mid \text{put}(\text{int}) < \text{put}(\text{int}).\text{put}(\text{int})$. Also note that $\text{put}(\sigma_1) < \text{put}(\sigma_2)$ holds only if σ_1 and σ_2 are syntactically equal; the reader is referred to [2, 3] for a presentation of a system in which a more advanced notion of subtyping is considered.

Lemma 4.7. The operators “ \mid ” and “ \cdot ” on behaviors respect the relation “ $<$ ”; thus the equivalence relation \equiv induced by $<$ is a congruence on behaviors wrt. these operators. Moreover, modulo \equiv it holds that “ \mid ” is associative and commutative and that “ \cdot ” is associative, both with ε as neutral element. □

The relation on behaviors induces a relation on behavior contexts:

Definition 4.8. $B_1 < B_2$ holds iff for all behaviors b we have $B_1[b] < B_2[b]$. □

Additionally, we also define a relation $\tau_1 < \tau_2$ over types; this relation is defined as the least one satisfying the clauses⁸ presented in Fig. 5. Notice that subtyping over ambient types is invariant, as the behaviors of the two ambient types are required to be subtypes of each other. Intuitively, this follows from the fact that ambient names can be used to (i) create new ambients as in $n[P]$ and (ii) dissolve existing ambients as in $\text{open } n.Q$. As the proof of Theorem 6.6 shows in the case for (Red Open), the former operation is contravariant by nature while the latter is covariant by nature. We could thus, as in [2], play the trick of [22] (akin to the “split types” of [5] for an object-oriented calculus) and split each of these arguments into two: one contravariant and the other covariant. But to keep things simple, we refrain from doing so.

5 The Type System

Figure 6 defines judgements $E \vdash M : \tau$ and $E \vdash P : b$, where E is an environment mapping names into types (we write $E, n : \sigma$ for the environment E' that behaves as E except that it maps n into σ). We employ

⁸Strictly speaking, the clause (Typ Trans) is superfluous.

$\tau <: \tau$	(Typ Refl)
If $\tau_1 <: \tau_2$ and $\tau_2 <: \tau_3$ then $\tau_1 <: \tau_3$	(Typ Trans)
If $b_1 <: b_2$ and $b_2 <: b_1$ then $\text{amb}[b_1] <: \text{amb}[b_2]$	(Typ Amb)
If $B_1 <: B_2$ then $\text{cap}[B_1] <: \text{cap}[B_2]$	(Typ Cap)

Figure 5. The Subtyping Relation.

Non-structural Rules

(Proc Subsumption)

$$\frac{E \vdash P : b}{E \vdash P : b'} \quad (b <: b')$$

(Exp Subsumption)

$$\frac{E \vdash M : \tau}{E \vdash M : \tau'} \quad (\tau <: \tau')$$

Expressions

(Exp n)

$$\frac{E(n) = \tau}{E \vdash n : \tau}$$

(Exp c)

$$\frac{\text{type}(c) = \tau}{E \vdash c : \tau}$$

(Exp ϵ)

$$E \vdash \epsilon : \text{cap}[\square]$$

(Exp In)

$$\frac{E \vdash M : \text{amb}[b]}{E \vdash \text{in } M : \text{cap}[\square]}$$

(Exp Out)

$$\frac{E \vdash M : \text{amb}[b]}{E \vdash \text{out } M : \text{cap}[\square]}$$

(Exp Open)

$$\frac{E \vdash M : \text{amb}[b]}{E \vdash \text{open } M : \text{cap}[b \mid \square]}$$

(Exp Coopen)

$$\frac{E \vdash M : \text{amb}[b]}{E \vdash \text{coopen } M : \text{cap}[\text{diss.}\square]}$$

(Exp Action)

$$\frac{E \vdash M_1 : \text{cap}[B_1] \quad E \vdash M_2 : \text{cap}[B_2]}{E \vdash M_1.M_2 : \text{cap}[B_1[B_2]]}$$

Processes

(Proc Zero)

$$E \vdash \mathbf{0} : \varepsilon$$

(Proc Par)

$$\frac{E \vdash P_1 : b_1 \quad E \vdash P_2 : b_2}{E \vdash P_1 \mid P_2 : b_1 \mid b_2}$$

(Proc Repl)

$$\frac{E \vdash P : b}{E \vdash !P : b} \quad (\text{if } (b \mid b) <: b)$$

(Proc Res)

$$\frac{E, n : \text{amb}[b] \vdash P : b_1}{E \vdash (\nu n : \text{amb}[b]).P : b_1}$$

(Proc Amb)

$$\frac{E \vdash M : \text{amb}[b_0] \quad E \vdash P : b}{E \vdash M[P] : \varepsilon} \quad (\text{if } b \text{ safe and } b \rightsquigarrow b_0)$$

(Proc Action)

$$\frac{E \vdash M : \text{cap}[B] \quad E \vdash P : b}{E \vdash M.P : B[b]}$$

(Proc Input)

$$\frac{E, n_1 : \tau_1, \dots, n_k : \tau_k \vdash P : b}{E \vdash (n_1 : \tau_1, \dots, n_k : \tau_k).P : \text{get}(\sigma).b}$$

(Proc Output)

$$\frac{E \vdash M_1 : \tau_1 \dots E \vdash M_k : \tau_k}{E \vdash \langle M_1, \dots, M_k \rangle : \text{put}(\sigma)}$$

In (Proc Input) and (Proc Output), $\sigma = \times(\tau_1, \dots, \tau_k)$ and $k \geq 0$.

Figure 6. Typing Rules.

a function $\text{type}()$ to assign types to the constants. Note that if $E \vdash M : \tau$ then there exists $\tau^- <: \tau$ such that $E \vdash M : \tau^-$ is derived by a structural inference rule, and if $E \vdash P : b$ then there exists $b^- <: b$ such that $E \vdash P : b^-$ is derived by a structural inference rule.

The side condition in (Proc Repl) prevents us from assigning $!\langle 7 \rangle$ the incorrect behavior $\text{put}(\text{int})$. The correct behavior for $!\langle 7 \rangle$ is $\text{fromnow}(\text{int})$, and can be obtained by applying the rule (Proc Subsumption). The side condition for (Proc Amb) employs a couple of notions which will be formally defined below.

First we capture the intuition that if P can be assigned a *safe* behavior then all communications performed by P will be well typed—at least until the ambient enclosing P is dissolved.

Definition 5.1 (Safe behavior). A behavior b is safe if no trace $tr \in \llbracket b \rrbracket$ can be written $tr = tr_0 \diamond tr_1 \diamond tr_2$ with $tr_0 \in \text{Comm}^*$ and $tr_1 \in \text{Comm} \setminus \text{WtComm}$. \square

Note that we might equivalently say that b is safe if $\llbracket b \rrbracket$ contains no trace of the form $tr_0 \diamond tr_1 \diamond tr_2$ with $tr_0 \in \text{WtComm}^*$ and $tr_1 \in \text{Comm} \setminus \text{WtComm}$.

EXAMPLE 5.2. The behavior $b = \text{put}(\text{int}) \mid \text{get}(\text{bool})$ is not safe. Intuitively, this is because a process that expects a boolean instead receives an integer. Formally, this is because $\llbracket b \rrbracket$ contains the trace $\text{put}(\text{int})\text{get}(\text{bool})$ which belongs to $\text{Comm} \setminus \text{WtComm}$.

On the other hand, perhaps surprisingly, the behavior $b = \text{diss}(\text{put}(\text{int}) \mid \text{get}(\text{bool}))$ is safe. Intuitively, this is because nothing bad happens as long as no one attempts to open the enclosing ambient (a process doing that would not be safe). Formally, this is because no trace in b has a prefix in Comm .

The behavior $b = \text{put}(\text{int}) \mid \text{get}(\text{int}).\text{get}(\text{bool}) \mid \text{put}(\text{int})$ is not safe. For $\llbracket b \rrbracket$ contains the trace

$$\text{put}(\text{int}) \text{get}(\text{int}) \text{put}(\text{int}) \text{get}(\text{bool})$$

which belongs to $\text{Comm} \diamond (\text{Comm} \setminus \text{WtComm})$. \square

EXAMPLE 5.3. Consider the program

$$m[\langle 7 \rangle \mid (x : \text{int}).\text{open } n.\langle x = 42 \rangle \mid n\{(y : \text{bool}).\mathbf{0}\}]$$

which is basically the example presented in Sect. 1.1, rewritten into our notation and with $P = \mathbf{0}$. It is easy to verify that the process enclosed within m has behavior

$$b = \text{put}(\text{int}) \mid \text{get}(\text{int}).(\text{get}(\text{bool}) \mid \text{put}(\text{bool}))$$

which is safe. This follows by a simple examination of the traces in $\llbracket b \rrbracket$, listed in Example 4.3: for the first trace in b belongs to WtComm^* ; the second trace can be written as $tr_0 \diamond tr$ with $tr_0 \in \text{WtComm}$ and tr not of the form $tr_1 \diamond tr_2$ for any $tr_1 \in \text{Comm}$; and none of the remaining traces are of the form $tr_1 \diamond tr_2$ with $tr_1 \in \text{Comm}$. \square

Concerning the relation $b \rightsquigarrow b_0$, the idea is that b_0 denotes “what remains” of b after its first occurrence of diss . For example, with $b = \text{get}(\text{int}).\text{diss} \mid \text{put}(\text{int})$ we have $b \rightsquigarrow \varepsilon$ since we can infer that $\text{put}(\text{int})$ is performed before diss . Similarly, for $b = \text{fromnow}(T) \mid \text{diss}$ we have $b \rightsquigarrow \text{fromnow}(T)$.

Definition 5.4 ($b \rightsquigarrow b'$). The relation $b \rightsquigarrow b'$ amounts to the following property: whenever there exists $tr_1 \in \text{Comm}^*$ and tr such that $tr_1 \diamond \text{diss} \diamond tr \in \llbracket b \rrbracket$, then $tr \in \llbracket b' \rrbracket$. \square

Note that if b contains no diss then $b \rightsquigarrow b'$ holds for all⁹ b' .

EXAMPLE 5.5 (PACKED ROUTING). Let us show how to type Example 2.1. Let the behavior b be defined as

$$b = \text{get}(\text{string}).(\text{get}(\text{cap}[\square]).\varepsilon \mid \text{put}(\text{cap}[\square]))$$

By assigning the behavior $\text{get}(\text{cap}[\square]).\varepsilon \mid \text{diss}$ to the body of *hop* (which can then be given the ambient type $\text{amb}[\text{get}(\text{cap}[\square]).\varepsilon]$), the safe behavior $b \mid \text{diss}$ to the body of *route* (which can then be given the ambient type $\text{amb}[b]$), and the safe behavior $b \mid \text{put}(\text{string})$ to the body of *packet*, we can construct a type derivation for

$$\begin{aligned} & \text{router}[\text{!route}\{\text{in } \text{packet}.\langle \text{dst} \rangle.\text{open } \text{hop}.\langle \text{lookup-route}(\text{dst}) \rangle\}] \mid \\ & \text{packet}[\text{in } \text{router}.\text{open } \text{route}.\langle \text{“bu”} \rangle \mid \text{hop}\{(x).x\}] \end{aligned}$$

We leave the construction of the derivation, as well as the verification of the appropriate side conditions, as an exercise for the reader. \square

⁹We might consider introducing a special “void” behavior $*$ such that $\llbracket * \rrbracket = \emptyset$. Then for b not containing diss we would have $b \rightsquigarrow *$, enabling us to assign the type $\text{amb}[*]$ to an ambient which does not allow itself to be opened. For subject reduction to carry through, however, we must ensure (by suitable side conditions) that Lemma 4.2 still holds (that is, if $E \vdash P : b$ then $\llbracket b \rrbracket \neq \emptyset$). To see why, consider the process $P = (x : \text{int}).(\text{open } n.\langle x + 7 \rangle) \mid \langle \text{true} \rangle$ where n has type $\text{amb}[*]$. If we do not put any restrictions on our type system we can derive $E \vdash P : *$ (since $\text{get}(\text{int}).(* \mid \text{put}(\text{int})) <: *$), even though with $\ell = \text{comm}(\text{int})$ it holds that $P \xrightarrow{\ell} Q$ where $Q = \text{open } n.\langle \text{true} + 7 \rangle$ which clearly cannot be typed. In order to type a process that attempts to open a locked ambient, we must thus assign it a non-void behavior such as, e.g., ε .

Below are some technical results, to be used later.

Lemma 5.6. *Suppose $b <: b^+$ with b^+ safe. Then also b is safe.* \square

Lemma 5.7. *Suppose that $\llbracket b' \rrbracket \cap \text{Comm}^* \neq \emptyset$. If $b'.b$ is safe then also b is safe.* \square

Proof. Assume b is not safe, that is there exists $tr \in \llbracket b \rrbracket$ such that we can write $tr = tr_0 \diamond tr_1 \diamond tr_2$ with $tr_0 \in \text{Comm}^*$ and $tr_1 \in \text{Comm} \setminus \text{WtComm}$. Let tr' belong to $\llbracket b' \rrbracket$ and to Comm^* . Then $tr' \diamond tr \in \llbracket b'.b \rrbracket$, and also $tr' \diamond tr = (tr' \diamond tr_0) \diamond tr_1 \diamond tr_2$ with $tr' \diamond tr_0 \in \text{Comm}^*$. But this conflicts with $b'.b$ being safe. \square

Lemma 5.8. *Assume that $b_1^- <: b_1$ and that $b_0 <: b_0^+$. If $b_1 \rightsquigarrow b_0$ then also $b_1^- \rightsquigarrow b_0^+$.* \square

Proof. Assume that $tr_1 \diamond \text{diss} \diamond tr \in \llbracket b_1^- \rrbracket$ with $tr_1 \in \text{Comm}^*$. Since $b_1^- <: b_1$, also $tr_1 \diamond \text{diss} \diamond tr \in \llbracket b_1 \rrbracket$, so since $b_1 \rightsquigarrow b_0$ we infer that $tr \in \llbracket b_0 \rrbracket$ which as $b_0 <: b_0^+$ implies the desired relation $tr \in \llbracket b_0^+ \rrbracket$. \square

Lemma 5.9. *Suppose that $\llbracket b_0 \rrbracket \cap \text{Comm}^* \neq \emptyset$. If $b_0.b \rightsquigarrow b'$ then also $b \rightsquigarrow b'$.* \square

Proof. Assume that $tr_1 \diamond \text{diss} \diamond tr \in \llbracket b \rrbracket$ with $tr_1 \in \text{Comm}^*$. Let tr_0 belong to $\llbracket b_0 \rrbracket$ and to Comm^* . Then $tr_0 \diamond tr_1 \diamond \text{diss} \diamond tr \in \llbracket b_0.b \rrbracket$ with $tr_0 \diamond tr_1 \in \text{Comm}^*$, so from $b_0.b \rightsquigarrow b'$ we infer the desired $tr \in \llbracket b' \rrbracket$. \square

6 Subject Reduction

In this section we shall show that our type system is semantically sound. This property is formulated as a subject reduction result (Theorem 6.6), intuitively stating that “well-typed processes communicate according to their behavior” and also stating that “well-typed safe processes never evolve into ill-typed processes”. The latter “safety” property demonstrates that the process $((n : \text{int}).\langle n + 7 \rangle) \mid \langle \text{true} \rangle$, though typeable (as safety is only enforced in (Proc Amb)), cannot be assigned a safe behavior, since it evolves into $\text{true} + 7$ which clearly cannot be typed. As a preparation for showing this result, we state a few standard lemmas.

Lemma 6.1 (Swapping). *Assume that $E_1, n_1 : \sigma_1, n_2 : \sigma_2, E_2 \vdash P : b$ with $n_1 \neq n_2$. Then it follows that $E_1, n_2 : \sigma_2, n_1 : \sigma_1, E_2 \vdash P : b$, with a derivation of the same shape. Similarly for $E_1, n_1 : \sigma_1, n_2 : \sigma_2, E_2 \vdash M : \tau$.* \square

Lemma 6.2 (Weakening). *Assume that $E \vdash P : b$, and that n is a name not in $\text{names}(P)$. Then for all τ it follows that $E, n : \tau \vdash P : b$, with a derivation of the same shape. Similarly for a judgment $E \vdash M : \tau_0$ (with $n \notin \text{names}(M)$).* \square

Lemma 6.3 (Strengthening). *Assume that $E, n : \tau \vdash P : b$ with n not in $\text{names}(P)$. Then also $E \vdash P : b$ holds, and with a derivation of the same shape. Similarly for $E, n : \tau \vdash M : \tau_0$ with $n \notin \text{names}(M)$.* \square

These lemmas are all proved by induction on derivations. In the proof of Lemma 6.2, we apply Lemma 6.1 to deal with the cases (Proc Res) and (Proc Input); similarly for the proof of Lemma 6.3.

Lemma 6.4 (Substitution). *Assume that $E, n_1 : \tau_1, \dots, n_k : \tau_k \vdash P : b$ (with $n_1 \dots n_k$ distinct), and that there exists $M_1 \dots M_k$ such that for all $i \in \{1 \dots k\}$ it holds that $E \vdash M_i : \tau_i$ and that P is non-conflicting with $\{n_i\} \cup \text{names}(M_i)$. Then $E \vdash P[n_i := M_i] : b$. Similarly, if $E, n_1 : \tau_1, \dots, n_k : \tau_k \vdash M : \tau$ and for all i it holds that $E \vdash M_i : \tau_i$ and that M is non-conflicting with $\{n_i\} \cup \text{names}(M_i)$ then $E \vdash M[n_i := M_i] : \tau$.* \square

Proof. The proof is by induction on the size of the typing derivation for P (resp. M). We do a case analysis on the inference rule applied, but only list two of the cases; the remaining follow by straightforward applications of the induction hypothesis, except for (Proc Input) which is handled as (Proc Res) below.

(Exp n). Here M is a name n . If $n = n_i$ for some $i \in \{1 \dots k\}$, we infer that $\tau = \tau_i$. The claim is then $E \vdash M_i : \tau_i$, which is among our assumptions. If $n \neq n_i$ for all $i \in \{1 \dots k\}$, we infer that $\tau = E(n)$. But then we have $E \vdash n : \tau$, as desired.

(Proc Res). Here P takes the form $(\nu n : \tau).P_0$, and $E, n_1 : \tau_1, \dots, n_k : \tau_k \vdash P : b$ holds because

$$E, n_1 : \tau_1, \dots, n_k : \tau_k, n : \tau \vdash P_0 : b. \quad (1)$$

Note that for all $i \in \{1 \dots k\}$ it holds (since P is non-conflicting with $\{n_i\} \cup \text{names}(M_i)$) that $n_i \neq n$ and $n \notin \text{names}(M_i)$ and that P_0 is non-conflicting with $\{n_i\} \cup \text{names}(M_i)$. We can thus apply Lemma 6.2 to infer that $E, n : \tau \vdash M_i : \tau_i$, and (repeatedly) apply Lemma 6.1 to infer that

$$E, n : \tau, n_1 : \tau_1, \dots, n_k : \tau_k \vdash P_0 : b \quad (2)$$

by a derivation of the same shape as the one for (1). We can thus apply the induction hypothesis on (2) to infer that $E, n : \tau \vdash P_0[n_i := M_i] : b$. But then an application of (Proc Res) yields $E \vdash (\nu n : \tau).P_0[n_i := M_i] : b$, which is as desired since $P[n_i := M_i] = (\nu n : \tau).P_0[n_i := M_i]$. \square

Lemma 6.5 (Subject Congruence). *Suppose that $P \equiv Q$. Then $E \vdash P : b$ if and only if $E \vdash Q : b$.* \square

The proof is given in Appendix A. We do not know if Lemma 6.5 still holds if the equivalence rule $!P \equiv P \mid !P$ is included in the definition of structural equivalence. This is why our operational semantics includes the reduction rule $!P \longrightarrow P \mid !P$ instead.

The formulation of subject reduction for processes states that if a process having behavior b performs a step labeled ℓ then the resulting process can be assigned a behavior that denotes “what remains of b after ℓ ”. To formalize this, we employ a function $\mathcal{B}(\ell)$ defined by stipulating that

$$\begin{aligned} \mathcal{B}(\epsilon) &= \epsilon \\ \mathcal{B}(\text{comm}(\sigma)) &= \text{put}(\sigma).\text{get}(\sigma) \end{aligned}$$

Note that $\llbracket \mathcal{B}(\ell) \rrbracket \cap \text{Comm}^* \neq \emptyset$ (preconditions for Lemmas 5.7 and 5.9).

Theorem 6.6 (Subject Reduction for Processes). *Suppose that $P \xrightarrow{\ell} Q$. If with b safe it holds that $E \vdash P : b$ then there exists safe b' such that $E \vdash Q : b'$ and $\mathcal{B}(\ell).b' \prec b$.* \square

Proof. The proof is by induction on the derivation of $P \xrightarrow{\ell} Q$, with a case analysis on the last rule used. By using Lemmas 5.6 and 5.7, we infer that if $\mathcal{B}(\ell).b' \prec b$ then b' will be safe.

(Red In). Our assumptions are $n[\text{in } m.P \mid Q] \mid m[R] \xrightarrow{\epsilon} m[n[P \mid Q] \mid R]$ and $E \vdash n[\text{in } m.P \mid Q] \mid m[R] : b$. The two top-level parallel processes have been typed using instantiations of the rule (Proc Amb) of the form

$$\frac{E \vdash n : \text{amb}[b_n] \quad E \vdash \text{in } m.P \mid Q : b_0}{E \vdash n[\text{in } m.P \mid Q] : \epsilon} \quad \text{and} \quad \frac{E \vdash m : \text{amb}[b_m] \quad E \vdash R : b_3}{E \vdash m[R] : \epsilon}$$

where b_0 is safe and satisfies $b_0 \rightsquigarrow b_n$, and where b_3 is safe and satisfies $b_3 \rightsquigarrow b_m$. It is easy to see that $\epsilon \prec b$ holds. From the rightmost premise of the leftmost inference we infer that there exists B_0, b_1, b_2 such that $E \vdash \text{in } m : \text{cap}[B_0]$, $E \vdash P : b_1$, and $E \vdash Q : b_2$, with $\text{cap}[\square] \prec \text{cap}[B_0]$ and with $B_0[b_1] \mid b_2 \prec b_0$. Thus $\square \prec B_0$ and therefore $b_1 \prec B_0[b_1]$, implying $b_1 \mid b_2 \prec b_0$.

From the above we can derive $E \vdash P \mid Q : b_0$, enabling us to apply (Proc Amb) to get $E \vdash n[P \mid Q] : \epsilon$. Then we can derive $E \vdash n[P \mid Q] \mid R : b_3$, enabling us to apply (Proc Amb) to get, after also applying (Proc Subsumption), $E \vdash m[n[P \mid Q] \mid R] : b$. Since $\mathcal{B}(\epsilon).b \prec b$, this yields the claim with $b' = b$.

(Red Out). Our assumptions are $m[n[\text{out } m.P \mid Q] \mid R] \xrightarrow{\epsilon} n[P \mid Q] \mid m[R]$ and $E \vdash m[n[\text{out } m.P \mid Q] \mid R] : b$. The topmost ambient has been typed using an instantiation of the rule (Proc Amb) of the form

$$\frac{E \vdash m : \text{amb}[b_m] \quad E \vdash n[\text{out } m.P \mid Q] \mid R : b_3}{E \vdash m[n[\text{out } m.P \mid Q] \mid R] : \epsilon}$$

where b_3 is safe and satisfies $b_3 \rightsquigarrow b_m$, and where $\epsilon \prec b$. We infer that the ambient in parallel with R has been typed using an instantiation of the rule (Proc Amb) of the form

$$\frac{E \vdash n : \text{amb}[b_n] \quad E \vdash \text{out } m.P \mid Q : b_0}{E \vdash n[\text{out } m.P \mid Q] : \epsilon}$$

where b_0 is safe and satisfies $b_0 \rightsquigarrow b_n$. From the rightmost premise of this inference we infer that there exists B_0, b_1, b_2 such that $E \vdash \text{out } m : \text{cap}[B_0]$, $E \vdash P : b_1$, and $E \vdash Q : b_2$, with $\text{cap}[\square] < \text{cap}[B_0]$ and with $B_0[b_1] \mid b_2 < b_0$. Thus $\square < B_0$ and therefore $b_1 < B_0[b_1]$, implying $b_1 \mid b_2 < b_0$. Additionally, there exists b'_3 such that $E \vdash R : b'_3$ and $\varepsilon \mid b'_3 < b_3$, implying $b'_3 < b_3$ and thus also $E \vdash R : b_3$.

From the above we can derive $E \vdash P \mid Q : b_0$, enabling us to apply (Proc Amb) to get $E \vdash n[P \mid Q] : \varepsilon$, and we can also apply (Proc Amb) to get $E \vdash m[R] : \varepsilon$. Thus we can arrive at $E \vdash n[P \mid Q] \mid m[R] : b$, which yields the claim with $b' = b$ since $\mathcal{B}(\varepsilon).b < b$.

(Red Open). Our assumptions are that the process $\text{open } n.P \mid n[\text{coopen } n.Q \mid R] \xrightarrow{\varepsilon} P \mid Q \mid R$, and that $E \vdash \text{open } n.P \mid n[\text{coopen } n.Q \mid R] : b$. Let $E(n) = \text{amb}[b_n]$. The rightmost parallel process has been typed using an instantiation of (Proc Amb) of the form

$$\frac{E \vdash n : \text{amb}[b'_n] \quad E \vdash \text{coopen } n.Q \mid R : b_0}{E \vdash n[\text{coopen } n.Q \mid R] : \varepsilon}$$

where $b_0 \rightsquigarrow b'_n$. Since the left premise has been derived using (Exp n) followed by zero or more applications of (Exp Subsumption), from the polarity rule for ambient types we have $b'_n < b_n$. Concerning the right premise, we deduce that there exists B_2, b_2, b_3 with

$$E \vdash Q : b_2 \text{ and } E \vdash R : b_3 \tag{3}$$

such that $E \vdash \text{coopen } n.Q : B_2[b_2]$ where $B_2[b_2] \mid b_3 < b_0$ and where $\text{diss}.\square < B_2$, implying $\text{diss}.b_2 < B_2[b_2]$ and therefore $\text{diss}.b_2 \mid b_3 < b_0$ which by Lemma 5.8 implies that

$$\text{diss}.b_2 \mid b_3 \rightsquigarrow b_n. \tag{4}$$

The leftmost parallel process $\text{open } n.P$ has been typed using an instantiation of (Proc Action) of the form

$$\frac{E \vdash \text{open } n : \text{cap}[B_1] \quad E \vdash P : b_1}{E \vdash \text{open } n.P : B_1[b_1]} \tag{5}$$

where $B_1[b_1] < b$. The derivation of the left premise contains an instance of (Exp Open) with premise $E \vdash n : \text{amb}[b''_n]$, where $b_n < b''_n$ (by the polarity rule for ambient types) and where $\text{cap}[b''_n \mid \square] < \text{cap}[B_1]$. Therefore $(b''_n \mid \square) < B_1$ implying $b''_n \mid b_1 < B_1[b_1]$, enabling us to derive

$$b_n \mid b_1 < b. \tag{6}$$

We want to prove that $E \vdash P \mid Q \mid R : b$, as then our claim will follow with $b' = b$, and due to (5) and (3) this can be accomplished by showing $b_1 \mid b_2 \mid b_3 < b$. So let $tr \in \llbracket b_1 \mid b_2 \mid b_3 \rrbracket$ be given. There must exist tr_1, tr_2, tr_3 such that $tr_i \in \llbracket b_i \rrbracket$ for $i \in \{1, 2, 3\}$ and such that $tr \in \{tr_1\} \parallel \{tr_{23}\}$ for some $tr_{23} \in \{tr_2\} \parallel \{tr_3\}$. Since $\text{diss} \diamond tr_{23}$ belongs to $\llbracket (\text{diss}.b_2) \mid b_3 \rrbracket$ we infer from (4) that $tr_{23} \in \llbracket b_n \rrbracket$. This shows that $tr \in \llbracket b_1 \rrbracket \parallel \llbracket b_n \rrbracket = \llbracket b_1 \mid b_n \rrbracket$, which by (6) implies the desired $tr \in \llbracket b \rrbracket$.

(Red Comm). Our assumptions are that $(n_1 : \tau_1, \dots, n_k : \tau_k).P \mid \langle M_1, \dots, M_k \rangle \xrightarrow{\text{comm}(\sigma)} P[n_i := M_i]$ with $\sigma = \times(\tau_1, \dots, \tau_k)$, and that $E \vdash (n_1 : \tau_1, \dots, n_k : \tau_k).P \mid \langle M_1, \dots, M_k \rangle : b$. The two top-level parallel processes have been typed using instantiations of the rules (Proc Input) and (Proc Output) of the form

$$\frac{E, n_1 : \tau_1, \dots, n_k : \tau_k \vdash P : b'}{E \vdash (n_1 : \tau_1, \dots, n_k : \tau_k).P : \text{get}(\sigma).b'} \text{ and } \frac{E \vdash M_1 : \tau'_1 \dots E \vdash M_k : \tau'_k}{E \vdash \langle M_1, \dots, M_k \rangle : \text{put}(\sigma')}$$

where $\sigma' = \times(\tau'_1, \dots, \tau'_k)$ and where $\text{get}(\sigma).b' \mid \text{put}(\sigma') < b$. By Lemma 4.2 there exists $tr \in \llbracket b' \rrbracket$, and thus $\text{put}(\sigma') \text{get}(\sigma) \diamond tr$ belongs to $\llbracket \text{get}(\sigma).b' \mid \text{put}(\sigma') \rrbracket \subseteq \llbracket b \rrbracket$. Since b is safe, this shows that $\sigma' = \sigma$, and thus for all $i \in \{1 \dots k\}$ we have $\tau'_i = \tau_i$. By assumption, $(n_1 : \tau_1, \dots, n_k : \tau_k).P$ is non-conflicting with $\text{names}(M_i)$ and therefore P is non-conflicting with $\{n_i\} \cup \text{names}(M_i)$. We can thus apply Lemma 6.4 to infer that $E \vdash P[n_i := M_i] : b'$. Since $\mathcal{B}(\text{comm}(\sigma)).b' = \text{put}(\sigma).\text{get}(\sigma).b' < \text{put}(\sigma) \mid (\text{get}(\sigma).b') < b$, this yields the claim.

(Red Repl). Our assumptions are that $!P \xrightarrow{\epsilon} P \mid !P$, and that $E \vdash !P : b$ which must have been derived by zero or more applications of (Proc Subsumption) with $b_0 < b$ from $E \vdash !P : b_0$ which in turn is derived by (Proc Repl) from $E \vdash P : b_0$ where $b_0 \mid b_0 < b_0$. By (Proc Par) we can thus derive $E \vdash P \mid !P : b_0 \mid b_0$ and therefore by (Proc Subsumption) also $E \vdash P \mid !P : b$, from which the claim follows with $b' = b$.

(Red Par). Our assumptions are that $P \mid R \xrightarrow{\ell} Q \mid R$ because $P \xrightarrow{\ell} Q$, and that $E \vdash P \mid R : b$ which must have been derived from $E \vdash P : b_1$ and $E \vdash R : b_3$ with $b_1 \mid b_3 < b$.

We now prove that b_1 is safe. If this is not the case, there exists $tr_0 \diamond tr_1 \diamond tr_2 \in \llbracket b_1 \rrbracket$ with $tr_0 \in \text{Comm}^*$ and $tr_1 \in \text{Comm} \setminus \text{WtComm}$. By Lemma 4.2 we can find $tr_3 \in \llbracket b_3 \rrbracket$, thus $tr_0 \diamond tr_1 \diamond tr_2 \diamond tr_3 \in \llbracket b_1 \mid b_3 \rrbracket \subseteq \llbracket b \rrbracket$ which is a contradiction since b is safe.

We can thus apply the induction hypothesis to find safe b_2 such that $\mathcal{B}(\ell).b_2 < b_1$ and $E \vdash Q : b_2$. Let $b' = b_2 \mid b_3$. Then $E \vdash Q \mid R : b'$, and as $\mathcal{B}(\ell).b' < (\mathcal{B}(\ell).b_2) \mid b_3 < b_1 \mid b_3 < b$ this is as desired.

(Red Res). Our assumptions are that $(\nu n : \tau).P \xrightarrow{\ell} (\nu n : \tau).Q$ because $P \xrightarrow{\ell} Q$, and that $E \vdash (\nu n : \tau).P : b$ which must have been derived from $E, n : \tau \vdash P : b_0$ with $b_0 < b$. As b_0 is safe (Lemma 5.6), inductively we can find b' such that $\mathcal{B}(\ell).b' < b_0$ and $E, n : \tau \vdash Q : b'$. But then also $E \vdash (\nu n : \tau).Q : b'$, and as $\mathcal{B}(\ell).b' < b$ this is as desired.

(Red Amb). Our assumptions are that $n[P] \xrightarrow{\epsilon} n[Q]$ because $P \xrightarrow{\ell} Q$, and that $E \vdash n[P] : b$ which must have been derived from $E \vdash n : \text{amb}[b_n]$ and $E \vdash P : b_1$ where b_1 is safe and $b_1 \rightsquigarrow b_n$ and $\epsilon < b$. We can thus apply the induction hypothesis to find safe b'_1 such that $\mathcal{B}(\ell).b'_1 < b_1$ and $E \vdash Q : b'_1$. By Lemmas 5.8 and 5.9 we infer that $b'_1 \rightsquigarrow b_n$, showing that we can apply (Proc Amb) and (Proc Subsumption) to derive $E \vdash n[Q] : b$. This yields the claim with $b' = b$ since $\mathcal{B}(\epsilon).b < b$.

(Red \equiv). Our assumptions are that $P' \xrightarrow{\ell} Q'$ because $P' \equiv P$ and $P \xrightarrow{\ell} Q$ and $Q \equiv Q'$, and that $E \vdash P' : b$. By Lemma 6.5 we infer that $E \vdash P : b$, and by applying the induction hypothesis we find b' such that $\mathcal{B}(\ell).b' < b$ and $E \vdash Q : b'$. By one more application of Lemma 6.5, this yields the desired judgment $E \vdash Q' : b'$. \square

7 Type Checking

In this section we show that given a complete type derivation for some process P (or an expression M), we can check its validity according to the rules from Fig. 6. For this purpose we need to show: (i) that we can decide the side-conditions for all the rules in Fig. 6, (ii) that we can determine if a certain behavior b (resp. behavior context B) can be obtained by filling in the hole of some behavior context B' with some behavior b' (resp. behavior context B'') and, (iii) that we can check if two behaviors (types) are syntactically identical. Clearly, conditions (ii) and (iii) are decidable. In what follows, we prove that the side conditions for the rules (Proc Subsumption), (Exp Subsumption), (Proc Repl) and (Proc Amb) are also decidable. This will follow from Lemma 7.9, Lemma 7.15, Corollary 7.10, Lemmas 7.18 and 7.19.

7.1 Automata

We shall use techniques from the theory of finite non-deterministic automata (some of the constructions are adaptations of standard results, but still we give the details). We have chosen to disallow ϵ -transitions, thus gearing our exposition towards an actual implementation. In fact, we believe that this choice makes certain correctness proofs easier (as a string can be uniquely decomposed into a sequence of automaton labels).

Definition 7.1 (Automata). An automaton G is a quadruple $G = (\mathcal{Q}, \iota, F, \delta)$ with \mathcal{Q} a finite set of states, $\iota \in \mathcal{Q}$ the initial state, $F \subseteq \mathcal{Q}$ the set of acceptance states, and δ a transition relation: a finite set of triples of the form (q_1, a, q_2) with $q_1, q_2 \in \mathcal{Q}$ and a an action. \square

We write δ^* for the reflexive and transitive closure of δ . Thus $(q, \bullet, q) \in \delta^*$ for all $q \in \mathcal{Q}$, and if $(q, tr_1, q_1) \in \delta^*$ and $(q_1, tr_2, q_2) \in \delta^*$ then $(q, tr_1 \diamond tr_2, q_2) \in \delta^*$.

Definition 7.2 (Acceptance). The set of strings accepted by G , to be denoted $\text{Acc}(G)$, is given by $\{tr \mid \exists q \in F : (\iota, tr, q) \in \delta^*\}$. \square

The following result is immediate:

Lemma 7.3. Given G , it is decidable whether $\text{Acc}(G) = \emptyset$. \square

Definition 7.4 (Implementation). An automaton G implements a behavior b if $\llbracket b \rrbracket = \text{Acc}(G)$. \square

In Appendix A we shall prove (for behaviors of the syntax listed explicitly in Fig. 4):

Lemma 7.5. Given behavior b , we can construct G implementing b . \square

Lemma 7.6. Let G_1 and G_2 be automata. Then we can construct an automaton $G_1 \setminus G_2$ such that $\text{Acc}(G_1 \setminus G_2) = \text{Acc}(G_1) \setminus \text{Acc}(G_2)$. \square

With $G_j = (\mathcal{Q}_j, \iota_j, F_j, \delta_j)$ for $j = 1, 2$, we construct $G_1 \setminus G_2 = (\mathcal{Q}, \iota, F, \delta)$ as follows:

$$\begin{aligned} \mathcal{Q} &= \mathcal{Q}_1 \times \mathcal{P}(\mathcal{Q}_2) \\ \iota &= (\iota_1, \{\iota_2\}) \\ F &= \{(q_1, Q_2) \mid q_1 \in F_1 \text{ and } Q_2 \cap F_2 = \emptyset\} \\ \delta &= \{((q_1, Q_2), a, (q'_1, Q'_2)) \mid (q_1, a, q'_1) \in \delta_1 \wedge Q'_2 = \{q'_2 \in \mathcal{Q}_2 \mid \exists q_2 \in Q_2 : (q_2, a, q'_2) \in \delta_2\}\} \end{aligned}$$

To prove Lemma 7.6 we need two additional lemmas, both easily provable by induction on the length of tr :

Lemma 7.7. Suppose that $((q_1, Q_2), tr, (q'_1, Q'_2)) \in \delta^*$. Then $(q_1, tr, q'_1) \in \delta_1^*$, and if $(q_2, tr, q'_2) \in \delta_2^*$ with $q_2 \in Q_2$ then $q'_2 \in Q'_2$. \square

Lemma 7.8. Suppose that $(q_1, tr, q'_1) \in \delta_1^*$, and that $Q'_2 = \{q'_2 \in \mathcal{Q}_2 \mid \exists q_2 \in \mathcal{Q}_2 : (q_2, tr, q'_2) \in \delta_2^*\}$. Then $((q_1, Q_2), tr, (q'_1, Q'_2)) \in \delta^*$. \square

We are now ready to prove Lemma 7.6. For “ \subseteq ”, our assumptions are that $((\iota_1, \{\iota_2\}), tr, (q_1, Q_2)) \in \delta^*$ with $q_1 \in F_1$ and $Q_2 \cap F_2 = \emptyset$. By Lemma 7.7 we infer that $(\iota_1, tr, q_1) \in \delta_1^*$, implying that $tr \in \text{Acc}(G_1)$, and that whenever $(\iota_2, tr, q_2) \in \delta_2^*$ then $q_2 \in Q_2$ and thus $q_2 \notin F_2$, implying that $tr \notin \text{Acc}(G_2)$.

For “ \supseteq ”, our assumptions imply that there exists $q_1 \in F_1$ such that $(\iota_1, tr, q_1) \in \delta_1^*$, and that with $Q_2 = \{q_2 \mid (\iota_2, tr, q_2) \in \delta_2^*\}$ we have $Q_2 \cap F_2 = \emptyset$. By Lemma 7.8, we infer that $((\iota_1, \{\iota_2\}), tr, (q_1, Q_2)) \in \delta^*$ which amounts to $tr \in \text{Acc}(G_1 \setminus G_2)$ since $(q_1, Q_2) \in F$. This concludes the proof of Lemma 7.6.

Lemma 7.9. Given behaviors b_1 and b_2 , it is decidable whether $b_1 < b_2$. \square

Proof. By Lemma 7.5, we can construct G_1 and G_2 implementing b_1 and b_2 . By Lemma 7.6, we can then construct $G_1 \setminus G_2$ such that $\text{Acc}(G_1 \setminus G_2) = \text{Acc}(G_1) \setminus \text{Acc}(G_2) = \llbracket b_1 \rrbracket \setminus \llbracket b_2 \rrbracket$.

Thus deciding $b_1 < b_2$, that is deciding whether $\llbracket b_1 \rrbracket \subseteq \llbracket b_2 \rrbracket$, amounts to deciding whether $\text{Acc}(G_1 \setminus G_2) = \emptyset$, a property which is decidable by Lemma 7.3. \square

Corollary 7.10. Given a behavior b , it is decidable whether $b \mid b < b$. \square

7.2 Deciding Subtyping

Next we consider the problem of deciding whether a given type is a subtype of another. Clearly the main difficulty lies in deciding whether $\text{cap}[B_1] < \text{cap}[B_2]$, that is whether $B_1 < B_2$, as this apparently involves testing $B_1[b] < B_2[b]$ for all possible b 's. Fortunately, Lemma 7.14 shows that it suffices to construct a single behavior, of the form test.test , and then check if $B_1[\text{test.test}] < B_2[\text{test.test}]$. Here test is an action (and thus also a behavior) that *tests* the behavior contexts in question:

Definition 7.11. We say that an action test tests a behavior b iff for all $tr \in \llbracket b \rrbracket$, test does not occur in tr .

We say that an action test tests a behavior context B iff for all behaviors b occurring in B it holds that test tests b . (Here b occurs in $b_0.B$ iff $b = b_0$ or b occurs in B , etc.) \square

Assume that test tests B . Then $\llbracket \text{test} \rrbracket = \{\text{test}\}$, and test also tests all subcontexts of B (that is, if $B = b'.B'$ then test tests B' , etc). Furthermore, it is easy to see by induction on B that all $tr \in \llbracket B[\text{test.test}] \rrbracket$ can be uniquely written on the form $tr_1 \diamond \text{test} \diamond tr_2 \diamond \text{test} \diamond tr_3$, where none of the traces tr_1, tr_2, tr_3 contain test.

As a preparation for Lemma 7.14, we show a couple of results:

Lemma 7.12. Let B be a behavior context and b a behavior, and assume that test tests B . Let $tr_1 \diamond \text{test} \diamond tr_2 \diamond \text{test} \diamond tr_3$ belong to $\llbracket B[\text{test.test}] \rrbracket$, and let tr_0 belong to $\{tr_2\} \parallel \llbracket b \rrbracket$. Then $tr_1 \diamond tr_0 \diamond tr_3$ belongs to $\llbracket B[b] \rrbracket$. \square

Proof. The proof is by induction on B , where we perform a case analysis on the form of B (omitting two symmetric cases).

$B = \square$. We infer that $tr_1 = tr_2 = tr_3 = \bullet$, and that $tr_0 \in \llbracket b \rrbracket$. Then the claim clearly holds.

$B = b'.B'$. Given the assumptions of the lemma, we can (since $B[\text{test.test}] = b'.B'[\text{test.test}]$) write $tr_1 = tr' \diamond tr'_1$ where $tr' \in \llbracket b' \rrbracket$ and where $tr'_1 \diamond \text{test} \diamond tr_2 \diamond \text{test} \diamond tr_3$ belongs to $\llbracket B'[\text{test.test}] \rrbracket$. Using the induction hypothesis, we infer that $tr'_1 \diamond tr_0 \diamond tr_3$ belongs to $\llbracket B'[b] \rrbracket$. But then $tr_1 \diamond tr_0 \diamond tr_3$ belongs to $\llbracket b' \rrbracket \diamond \llbracket B'[b] \rrbracket = \llbracket b'.B'[b] \rrbracket = \llbracket B[b] \rrbracket$, as desired.

$B = b' \mid B'$. Given the assumptions of the lemma, we infer (since $B[\text{test.test}] = b' \parallel B'[\text{test.test}]$) that there exists $tr'_1, tr'_2, tr'_3, tr''_1, tr''_2, tr''_3$ such that for $i \in \{1, 2, 3\}$ we have $tr_i \in \{tr'_i\} \parallel \{tr''_i\}$ and such that $tr'_1 \diamond tr'_2 \diamond tr'_3 \in \llbracket b' \rrbracket$ and such that $tr''_1 \diamond \text{test} \diamond tr''_2 \diamond \text{test} \diamond tr''_3$ belongs to $\llbracket B'[\text{test.test}] \rrbracket$. By associativity of interleaving, there exists $tr'_0 \in \{tr'_2\} \parallel \llbracket b \rrbracket$ such that $tr_0 \in \{tr'_2\} \parallel \{tr'_0\}$. Using the induction hypothesis on B' , we infer that $tr''_1 \diamond tr'_0 \diamond tr''_3 \in \llbracket B'[b] \rrbracket$. But then clearly $tr_1 \diamond tr_0 \diamond tr_3 \in \llbracket b' \rrbracket \parallel \llbracket B'[b] \rrbracket = \llbracket B[b] \rrbracket$, as desired. \square

Lemma 7.13. Let B be a behavior context and b a behavior, and assume that test tests B . Let tr belong to $\llbracket B[b] \rrbracket$. Then there exists $tr_1 \diamond \text{test} \diamond tr_2 \diamond \text{test} \diamond tr_3$ in $\llbracket B[\text{test.test}] \rrbracket$ such that we can write $tr = tr_1 \diamond tr_0 \diamond tr_3$ with $tr_0 \in \{tr_2\} \parallel \llbracket b \rrbracket$. \square

Proof. The proof is by induction on B , where we perform a case analysis on the form of B (omitting two symmetric cases).

$B = \square$. Now $tr \in \llbracket b \rrbracket$, so choosing $tr_0 = tr$ and $tr_1 = tr_2 = tr_3 = \bullet$ clearly yields the claim.

$B = b'.B'$. We can write $tr = tr' \diamond tr''$ with $tr' \in \llbracket b' \rrbracket$ and with $tr'' \in \llbracket B'[b] \rrbracket$. Inductively, there exists $tr'_1 \diamond \text{test} \diamond tr'_2 \diamond \text{test} \diamond tr'_3$ in $\llbracket B'[\text{test.test}] \rrbracket$ such that we can write $tr'' = tr'_1 \diamond tr_0 \diamond tr_3$ with $tr_0 \in \{tr_2\} \parallel \llbracket b \rrbracket$. Let $tr_1 = tr' \diamond tr'_1$, then we have the desired relations $tr_1 \diamond \text{test} \diamond tr_2 \diamond \text{test} \diamond tr_3 \in \llbracket b' \rrbracket \diamond \llbracket B'[\text{test.test}] \rrbracket = \llbracket B[\text{test.test}] \rrbracket$ and $tr = tr_1 \diamond tr_0 \diamond tr_3$.

$B = b' \mid B'$. There exists $tr' \in \llbracket b' \rrbracket$ and $tr'' \in \llbracket B'[b] \rrbracket$ such that $tr \in \{tr'\} \parallel \{tr''\}$. The induction hypothesis gives us $tr''_1 \diamond \text{test} \diamond tr''_2 \diamond \text{test} \diamond tr''_3$ in $\llbracket B'[\text{test.test}] \rrbracket$ such that $tr'' = tr''_1 \diamond tr''_0 \diamond tr''_3$ with $tr''_0 \in \{tr_2\} \parallel \llbracket b \rrbracket$. Since $tr \in \{tr'\} \parallel \{tr''_1 \diamond tr''_0 \diamond tr''_3\}$ there clearly exists $tr'_1, tr_1, tr'_3, tr_3, tr'_0, tr_0$ such that $tr' = tr'_1 \diamond tr'_0 \diamond tr'_3$, $tr = tr_1 \diamond tr_0 \diamond tr_3$, and for $i \in \{0, 1, 3\}$ also $tr_i \in \{tr'_i\} \parallel \{tr''_i\}$. Since tr_0 belongs to $\{tr'_0\} \parallel (\{tr''_2\} \parallel \llbracket b \rrbracket)$, associativity of interleaving enables us to find $tr_2 \in \{tr'_0\} \parallel \{tr''_2\}$ such that $tr_0 \in \{tr_2\} \parallel \llbracket b \rrbracket$. This establishes the desired relation $tr_1 \diamond \text{test} \diamond tr_2 \diamond \text{test} \diamond tr_3 \in \{tr'_1 \diamond tr'_0 \diamond tr'_3\} \parallel \{tr''_1 \diamond \text{test} \diamond tr''_2 \diamond \text{test} \diamond tr''_3\} \subseteq \llbracket b' \rrbracket \parallel \llbracket B'[\text{test.test}] \rrbracket = \llbracket B[\text{test.test}] \rrbracket$. \square

Lemma 7.14. Given B_1 and B_2 behavior contexts, we can construct a behavior test such that $B_1 < B_2$ holds if and only if $B_1[\text{test.test}] < B_2[\text{test.test}]$. \square

Proof. With m the maximal arity of a tuple occurring inside B_1 or B_2 , we choose¹⁰ test to be a behavior of the form $\text{put}(\times(\tau_1, \dots, \tau_{m+1}))$ where $\tau_i = \text{int}$ for $i \in \{1, \dots, m+1\}$. Then clearly test tests B_1 as well as B_2 .

¹⁰Alternatively, we might have introduced a special behavior test as part of our syntax.

The non-trivial point is to show “if”. For this purpose, assume that b has been given. Let tr belong to $\llbracket B_1[b] \rrbracket$, then our task is to show that $tr \in \llbracket B_2[b] \rrbracket$. By Lemma 7.13, there exists $tr_1 \diamond \text{test} \diamond tr_2 \diamond \text{test} \diamond tr_3$ in $\llbracket B_1[\text{test.test}] \rrbracket$ such that we can write $tr = tr_1 \diamond tr_0 \diamond tr_3$ with $tr_0 \in \{tr_2\} \parallel \llbracket b \rrbracket$. Using our assumption, $tr_1 \diamond \text{test} \diamond tr_2 \diamond \text{test} \diamond tr_3 \in \llbracket B_2[\text{test.test}] \rrbracket$. By Lemma 7.12 we now infer that tr belongs to $\llbracket B_2[b] \rrbracket$. \square

Lemma 7.15. *Given types τ_1 and τ_2 , it is decidable whether $\tau_1 < \tau_2$.* \square

Proof. An easy case analysis. The most interesting cases are:

$\tau_1 = \text{amb}[b_1]$ and $\tau_2 = \text{amb}[b_2]$. Here $\tau_1 < \tau_2$ holds iff $b_1 < b_2$ and $b_2 < b_1$. But by Lemma 7.9, this is decidable.

$\tau_1 = \text{cap}[B_1]$ and $\tau_2 = \text{cap}[B_2]$. Here $\tau_1 < \tau_2$ holds iff $B_1 < B_2$. By Lemma 7.14 we can construct behavior test such that deciding $\tau_1 < \tau_2$ amounts to deciding $B_1[\text{test.test}] < B_2[\text{test.test}]$, which is decidable by Lemma 7.9. \square

7.3 Deciding the Auxiliary Relations

We are still left with deciding whether a given b is safe and whether $b \rightsquigarrow b_0$ holds for given b and b_0 . For that purpose, we introduce a couple of auxiliary concepts.

Definition 7.16. Let \mathcal{L} be a language, that is a member of $\mathcal{P}(\text{Trace})$, and let $G = (\mathcal{Q}, \iota, F, \delta)$ be an automaton. We then define

$$G\#\mathcal{L} = \{q \in \mathcal{Q} \mid \exists tr \in \mathcal{L} : (\iota, tr, q) \in \delta^*\}.$$

We say that \mathcal{L} is *testable* if for every automaton G there exists a procedure for computing $G\#\mathcal{L}$. \square

Lemma 7.17. *Let \mathcal{L} be testable, and let G be an automaton. Then we can construct an automaton $G - \mathcal{L}$ such that for all traces tr the following property holds: tr belongs to $\text{Acc}(G - \mathcal{L})$ if and only if there exists $tr_0 \in \mathcal{L}$ such that $tr_0 \diamond tr \in \text{Acc}(G)$.* \square

Proof. With $G = (\mathcal{Q}, \iota, F, \delta)$ and with ι_0 a symbol not in \mathcal{Q} , we construct $G - \mathcal{L} = (\mathcal{Q}_0, \iota_0, F_0, \delta_0)$ as follows:

$$\begin{aligned} \mathcal{Q}_0 &= \mathcal{Q} \cup \{\iota_0\} \\ F_0 &= F \cup (\text{if } (G\#\mathcal{L}) \cap F = \emptyset \text{ then } \emptyset \text{ else } \{\iota_0\}) \\ \delta_0 &= \delta \cup \{(\iota_0, a, q) \mid \exists q_0 \in G\#\mathcal{L} \text{ with } (q_0, a, q) \in \delta\}. \end{aligned}$$

Concerning the claim of the lemma, we first address the “only if” part. The assumption is that $tr \in \text{Acc}(G - \mathcal{L})$, that is there exists $q_0 \in F_0$ such that $(\iota_0, tr, q_0) \in \delta_0^*$. We must consider two cases:

- If $tr = \bullet$ then $\iota_0 \in F_0$, so by the construction of F_0 there exists q_1 in $(G\#\mathcal{L}) \cap F$. There thus exists $tr_0 \in \mathcal{L}$ with $(\iota, tr_0, q_1) \in \delta^*$ where $q_1 \in F$, implying that $tr_0 \diamond tr = tr_0 \in \text{Acc}(G)$.
- If tr takes the form $a \diamond tr_1$, we infer that there exists $q \in \mathcal{Q}_0$ such that $(\iota_0, a, q) \in \delta_0$ and such that $(q, tr_1, q_0) \in \delta_0^*$. The former relation implies the existence of $q_1 \in G\#\mathcal{L}$ with $(q_1, a, q) \in \delta$, and thus the existence of $tr_0 \in \mathcal{L}$ with $(\iota, tr_0, q_1) \in \delta^*$; the latter relation implies (since clearly $q \neq \iota_0$) that $(q, tr_1, q_0) \in \delta^*$ and $q_0 \neq \iota_0$ and thus (as $q_0 \in F_0$) also $q_0 \in F$. Since combining the abovementioned properties of δ^* yields $(\iota, tr_0 \diamond a \diamond tr_1, q_0) \in \delta^*$, this demonstrates that $tr_0 \diamond tr \in \text{Acc}(G)$, as desired.

Next we address the “if” part, where our assumptions are that there exists $tr_0 \in \mathcal{L}$ such that $tr_0 \diamond tr \in \text{Acc}(G)$, that is there exists $q \in F$ such that $(\iota, tr_0 \diamond tr, q) \in \delta^*$. We must consider two cases:

- If $tr = \bullet$ then $q \in G\#\mathcal{L}$, which by the construction of F_0 implies that $\iota_0 \in F_0$ and therefore $\bullet \in \text{Acc}(G - \mathcal{L})$, as desired.

- If tr takes the form $a \diamond tr_1$, there exists q_1 and q_2 such that $(\iota, tr_0, q_1) \in \delta^*$, implying $q_1 \in G\#L$, such that $(q_1, a, q_2) \in \delta$, thus implying $(\iota_0, a, q_2) \in \delta_0$, and such that $(q_2, tr_1, q) \in \delta^*$, implying that also $(q_2, tr_1, q) \in \delta_0^*$. Therefore $(\iota_0, tr, q) \in \delta_0^*$, and since $q \in F_0$ this demonstrates the desired relation $tr \in \text{Acc}(G - L)$. \square

Lemma 7.18. *Given a behavior b , it is decidable whether b is safe.* \square

Proof. The language $\mathcal{L}_0 = \text{Comm}^*$ is clearly testable, and since membership in WtComm can be decided by a simple syntactic check, so is the language $\mathcal{L}_1 = \text{Comm} \setminus \text{WtComm}$.

By Lemma 7.5 there exists an automaton G implementing b . Using Lemma 7.17 twice, we can construct the automaton $G' = (G - \mathcal{L}_0) - \mathcal{L}_1$. We shall prove that b is safe if and only if $\text{Acc}(G') = \emptyset$, a property which is decidable (Lemma 7.3).

First assume that b is not safe. Then there exists $tr_0 \in \mathcal{L}_0$ and $tr_1 \in \mathcal{L}_1$ and tr_2 such that $tr_0 \diamond tr_1 \diamond tr_2 \in \llbracket b \rrbracket = \text{Acc}(G)$. By Lemma 7.17 we infer first that $tr_1 \diamond tr_2 \in \text{Acc}(G - \mathcal{L}_0)$ and then that $tr_2 \in \text{Acc}(G')$. Hence, $\text{Acc}(G') \neq \emptyset$.

Next assume that $\text{Acc}(G') \neq \emptyset$. Let $tr \in \text{Acc}(G')$. By Lemma 7.17 we infer first that there exists $tr_1 \in \mathcal{L}_1$ such that $tr_1 \diamond tr \in \text{Acc}(G - \mathcal{L}_0)$, and next that there exists $tr_0 \in \mathcal{L}_0$ such that $tr_0 \diamond tr_1 \diamond tr \in \text{Acc}(G) = \llbracket b \rrbracket$. Hence, b is not safe. \square

Lemma 7.19. *Given b_1 and b_2 , it is decidable whether $b_1 \rightsquigarrow b_2$.* \square

Proof. By Lemma 7.5 there exists automata G_1 and G_2 implementing b_1 and b_2 . The language $\mathcal{L} = \text{Comm}^* \diamond \text{diss}$ is clearly testable, so using Lemma 7.17 we can construct an automaton $G_0 = G_1 - \mathcal{L}$. By Lemma 7.6 we can construct an automaton $G = G_0 \setminus G_2$. We shall prove that $b_1 \rightsquigarrow b_2$ holds if and only if $\text{Acc}(G) = \emptyset$, a property which is decidable (Lemma 7.3).

For “if”, we must establish $b_1 \rightsquigarrow b_2$ and therefore consider the situation where there exists $tr_1 \in \text{Comm}^*$ and tr such that $tr_1 \diamond \text{diss} \diamond tr \in \llbracket b_1 \rrbracket = \text{Acc}(G_1)$. Since $tr_1 \diamond \text{diss} \in \mathcal{L}$, we infer by Lemma 7.17 that $tr \in \text{Acc}(G_0)$. Since by assumption $tr \notin \text{Acc}(G)$, Lemma 7.6 tells us $tr \in \text{Acc}(G_2) = \llbracket b_2 \rrbracket$, which is just as desired.

For “only if”, we assume in order to arrive at a contradiction that there exists $tr \in \text{Acc}(G)$. By Lemma 7.6 we infer that $tr \in \text{Acc}(G_0)$ and that tr does not belong to $\text{Acc}(G_2) = \llbracket b_2 \rrbracket$. By Lemma 7.17 we infer that there exists $tr_1 \in \text{Comm}^* \diamond \text{diss}$ such that $tr_1 \diamond tr \in \text{Acc}(G_1) = \llbracket b_1 \rrbracket$. But since $b_1 \rightsquigarrow b_2$, this yields the desired contradiction. \square

7.4 Type Derivations

We are now ready to state the main result of this section. Namely, that given a complete derivation for an expression M or a process P , we can verify that the derivation is valid according to the rules from Fig. 6.

Theorem 7.20 (Decidability of type checking). *Given a purported derivation of $E \vdash M : \tau$ or $E \vdash P : b$, we can check its validity.* \square

It is clear, however, that the worst-case complexity of type checking is at least exponential. This follows since the number of states in the automaton for $b_1 \mid b_2$ is the product of the number of states in the automata for b_1 and b_2 , cf. the proof of Lemma 7.5. The issue of *inferring* types, perhaps with some guidance from the user, is left for future work.

8 Comparison with Other Systems

8.1 Relation to the Original Type System for the Ambient Calculus

There is a natural embedding of the type system for **AC** presented in [10, Sect. 3] into our type system. Towards this end we use a function \mathcal{G} translating entities in the former system into entities in our system:

expressions M^C into expressions, processes P^C into processes, “message types” W^C into types, “exchange types” T^C into behaviors, and environments E^C into environments. \mathcal{G} is defined recursively on the structure of its argument; most clauses are straightforward homomorphisms except for

$$\begin{aligned}\mathcal{G}(M^C[P^C]) &= \mathcal{G}(M^C)[\mathcal{G}(P^C) \mid \text{coopen } \mathcal{G}(M^C).\mathbf{0}] \\ \mathcal{G}(\text{cap}[T^C]) &= \text{cap}[\mathcal{G}(T^C) \mid \square] \\ \mathcal{G}(Shh) &= \varepsilon \\ \mathcal{G}(W_1^C \times \dots \times W_n^C) &= \text{fromnow}(\times(\mathcal{G}(W_1^C), \dots, \mathcal{G}(W_n^C)))\end{aligned}$$

We then have the following result, proved in Appendix A:

Theorem 8.1. *Suppose that $E^C \vdash P^C : T^C$, respectively $E^C \vdash M^C : W^C$, is derivable in the system of [10, Sect. 3]. Then $\mathcal{G}(E^C) \vdash \mathcal{G}(P^C) : \mathcal{G}(T^C)$, respectively $\mathcal{G}(E^C) \vdash \mathcal{G}(M^C) : \mathcal{G}(W^C)$, is derivable in our system. \square*

One can also show (as in [2]) that if $P_1^C \longrightarrow P_2^C$ holds in the system of [10], after replacing the rule $!P \equiv P \mid !P$ by the rule $!P \longrightarrow P \mid !P$, then $\mathcal{G}(P_1^C) \longrightarrow \mathcal{G}(P_2^C)$ holds in our system.

8.2 Type Systems for Ambient Interferences

In [1], the present type system is modified and augmented so as to record not only communications but also ambient movements, enabling one to reason about which ambients can occur inside which ambients and also enabling one to express, as in [7], that a given ambient is immobile. This is achieved by introducing actions enter and exit; additionally the syntax and semantics of \mathbf{AC}^* have to be extended in order to allow a statement of semantic soundness. In a sense, the type system of [22] can be viewed as an abstraction over traces containing such actions: for example, $\llbracket \forall O \rightsquigarrow I \rrbracket$ is the set of traces containing actions $\text{put}(\tau)$ with τ described by O , actions $\text{get}(\tau)$ with τ described by I , but no enter or exit actions.

8.3 Type Systems for the Pi-Calculus

There are many points of contact between the π -calculus and \mathbf{AC} . Other researchers have already related the two calculi and discussed their respective merits. Here, we restrict attention to *session types* in the π -calculus, because they were motivated by considerations similar to ours in relation to *orderly communication* in \mathbf{AC}^* .

Session types originated with the work of Honda and his collaborators, who proposed a variant of the π -calculus where some channels are designated to be *session channels*, in [20] and [15]. Such a channel is allowed to carry a sequence of different message types over time, by contrast to a channel that is restricted to a single type of message throughout its lifetime, as in a system of simple types for the π -calculus. More recently, Gay and Hole in [13] have developed a type system for the π -calculus which combines *session types*, subtyping and recursive types. Whereas session channels form a distinct syntactic category in [20] and [15], Gay and Hole enforce this distinction by means of their type system.

Despite the many similarities, there are also many differences between sessions types in the π -calculus and orderly communication in \mathbf{AC}^* . Technical issues regarding the former do not apply to the latter and vice-versa; this is best illustrated by some of the problems we have solved in relation to orderly communication which have no counterpart (or have not been raised) in relation to session types. However, a final assessment of their respective merits awaits a more systematic comparison, probably to be based on a translation from the π -calculus to \mathbf{AC}^* , or vice-versa, which is also type-preserving. By “type-preserving” we mean that if P is a process of the π -calculus and Q is its translation in \mathbf{AC}^* , then P is typable in the system with session types if and only if Q is typable in our type system for \mathbf{AC}^* with orderly communication. It is a question whether such a type-preserving translation, in either direction, is possible at all; we have not tried to carry it out.

References

- [1] Torben Amtoft. Causal type system for ambient movements. Submitted for publication, 2002.

- [2] Torben Amtoft, Assaf J. Kfoury, and Santiago M. Pericas-Geertsen. What are polymorphically-typed ambients? Technical Report BUCS-TR-2000-021, Comp. Sci. Dept., Boston Univ., December 2000.
- [3] Torben Amtoft, Assaf J. Kfoury, and Santiago M. Pericas-Geertsen. What are polymorphically-typed ambients? In David Sands, editor, *ESOP 2001, Genova*, volume 2028 of *LNCS*, pages 206–220. Springer-Verlag, April 2001. An extended version appears as [2].
- [4] Michele Bugliesi and Giuseppe Castagna. Secure safe ambients. In *Conf. Rec. POPL '01: 28th ACM Symp. Princ. of Prog. Langs.*, pages 222–235, 2001.
- [5] Michele Bugliesi and Santiago M. Pericas-Geertsen. Depth subtyping and type inference for object calculi. In *Proc. Seventh Workshop on Foundations of Object-Oriented Languages*, Boston, Mass., U.S.A., 2000.
- [6] Luca Cardelli. Abstractions for mobile computation. In Jan Vitek and Christian Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *LNCS*, pages 51–94. Springer-Verlag, 1999.
- [7] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Mobility types for mobile ambients. In Jiri Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP'99*, volume 1644 of *LNCS*, pages 230–239. Springer-Verlag, July 1999. Extended version appears as Microsoft Research Technical Report MSR-TR-99-32, 1999.
- [8] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Ambient groups and mobility types. In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, Tohoku University, Sendai, Japan, volume 1872 of *LNCS*, pages 333–347. Springer-Verlag, August 2000.
- [9] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In Maurice Nivat, editor, *FoSSaCS'98*, volume 1378 of *LNCS*, pages 140–155. Springer-Verlag, 1998.
- [10] Luca Cardelli and Andrew D. Gordon. Types for mobile ambients. In *POPL'99, San Antonio, Texas*, pages 79–92. ACM Press, January 1999.
- [11] M. Dezani-Ciancaglini and I. Salvo. Security types for mobile safe ambients. In *ASIAN Computing Sciece Conference - ASIAN'00*, volume 1961 of *LNCS*, pages 215–236. Springer, 2000.
- [12] Cedric Fournet, Georges Gonthier, Jean-Jacques Levy, Luc Maranget, and Didier Remy. A calculus of mobile agents. In *CONCUR 1996*, volume 1119 of *LNCS*, pages 406–421. Springer-Verlag, 1996.
- [13] Simon Gay and Malcolm Hole. Types and subtypes for client-server interactions. In *Proc. European Symp. on Programming*, volume 1576 of *LNCS*, pages 74–90. Springer-Verlag, 1999.
- [14] Xudong Guan, Yiling Yang, and Jinyuan You. Typing evolving ambients. *Information Processing Letters*, 80(5):265–270, November 2001.
- [15] Kohei Honda, Vasco Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 122–138. Springer-Verlag, 1998.
- [16] Francesca Levi and Davide Sangiorgi. Controlling interference in ambients. In *POPL'00, Boston, Massachusetts*, pages 352–364. ACM Press, January 2000.
- [17] James Riely and Matthew Hennessy. A typed language for distributed mobile processes. In *POPL 1998*, pages 378–390. ACM Press, 1998.
- [18] James Riely and Matthew Hennessy. Trust and partial typing in open systems of mobile agents. In *POPL'99, San Antonio, Texas*, pages 93–104. ACM Press, 1999.

- [19] Peter Sewell and Jan Vitek. Secure composition of insecure components. In *12th IEEE Computer Security Foundations Workshop (CSFW-12)*, Mordano, Italy, June 1999.
- [20] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In *PARLE'94*, volume 817 of *LNCS*, pages 398–413. Springer-Verlag, 1994.
- [21] Jan Vitek and Giuseppe Castagna. Seal: A framework for secure mobile computations. In *Internet Programming Languages*, volume 1686 of *LNCS*. Springer-Verlag, 1999.
- [22] Pascal Zimmer. Subtyping and typing algorithms for mobile ambients. In *FOSSACS 2000, Berlin*, volume 1784 of *LNCS*, pages 375–390. Springer-Verlag, 2000.

A Proofs of Results in Main Text

Lemma 6.5 Suppose that $P \equiv Q$. Then $E \vdash P : b$ if and only if $E \vdash Q : b$.

Proof. The proof is by induction on the derivation of $P \equiv Q$, much as the similar result in [10]. Many cases are trivial, such as (Struct ParComm) (which follows from commutativity of “ $|$ ”) and (Struct ResRes) (which follows easily from Lemma 6.1). Below we list the most interesting cases.

(Struct ParAssoc). Assume that $E \vdash (P | Q) | R : b$ (the other direction is similar). Then there exists b_{12} and b_3 with $b_{12} | b_3 < b$ such that $E \vdash P | Q : b_{12}$ and $E \vdash R : b_3$. Therefore there exists b_1 and b_2 with $b_1 | b_2 < b_{12}$ such that $E \vdash P : b_1$ and $E \vdash Q : b_2$. Now $E \vdash P | (Q | R) : b_1 | (b_2 | b_3)$, and by Lemma 4.7 we infer that $b_1 | (b_2 | b_3) \equiv (b_1 | b_2) | b_3 < b_{12} | b_3 < b$. This implies the desired judgment $E \vdash P | (Q | R) : b$.

(Struct ResPar). Assume that $E \vdash (\nu n : \tau).(P | Q) : b$. There exists b_1 and b_2 with $b_1 | b_2 < b$ such that $E, n : \tau \vdash P : b_1$ and $E, n : \tau \vdash Q : b_2$. Since $n \notin \text{fn}(P)$, we can α -rename P into a P' such that $n \notin \text{names}(P')$. Clearly $E, n : \tau \vdash P' : b_1$ (this claim amounts to the case (Struct α -rename)), so by Lemma 6.3 we have $E \vdash P' : b_1$ and therefore also $E \vdash P : b_1$. We can thus infer first $E \vdash (\nu n : \tau).Q : b_2$ and then (since $b_1 | b_2 < b$) the desired judgment $E \vdash P | (\nu n : \tau).Q : b$.

The other direction is quite similar, employing Lemma 6.2 rather than Lemma 6.3.

(Struct ResAmb). Let $E(m) = (E, n : \tau)(m) = \text{amb}[b_1]$. Assume that $E \vdash (\nu n : \tau).m[P] : b$ (the other direction is similar). We infer that $\varepsilon < b$, and that there exists safe b_0 such that $E, n : \tau \vdash P : b_0$ and $b_0 \rightsquigarrow b_1$. This implies that $E \vdash (\nu n : \tau).P : b_0$, clearly enabling us to infer the desired judgment $E \vdash m[(\nu n : \tau).P] : b$.

(Struct ZeroPar). Assume that $E \vdash P | \mathbf{0} : b$. There exists b_1 and b_2 with $b_1 | b_2 < b$ such that $E \vdash P : b_1$ and $E \vdash \mathbf{0} : b_2$. We infer that $\varepsilon < b_2$ and by Lemma 4.7 therefore $b_1 \equiv b_1 | \varepsilon < b_1 | b_2 < b$. This implies the desired judgment $E \vdash P : b$. The other direction is trivial.

(Struct ZeroRepl). First assume that $E \vdash !\mathbf{0} : b$. We infer that there exists $b_0 < b$ such that $E \vdash \mathbf{0} : b_0$. But then also $E \vdash \mathbf{0} : b$, as desired.

Conversely, assume that $E \vdash \mathbf{0} : b$ from which we infer that $\varepsilon < b$. Since $\varepsilon | \varepsilon < \varepsilon$ we can apply (Proc Zero) and (Proc Repl) to derive $E \vdash !\mathbf{0} : \varepsilon$, and therefore also the desired $E \vdash !\mathbf{0} : b$.

(Struct ε). Assume that $E \vdash P : b$. Since $E \vdash \varepsilon : \text{cap}[\square]$, we can by (Proc Action) infer $E \vdash \varepsilon.P : b$.

Conversely, assume that $E \vdash \varepsilon.P : b$. Then there exists B and b_0 with $B[b_0] < b$ such that $E \vdash \varepsilon : \text{cap}[B]$ and $E \vdash P : b_0$. As $\text{cap}[\square] < \text{cap}[B]$ we deduce that $\square < B$, implying that $b_0 = \square[b_0] < B[b_0] < b$. Thus we can derive the desired judgment $E \vdash P : b$.

(Struct \cdot). Assume that $E \vdash (M_1.M_2).P : b$. There exists B_0 and b_0 with $B_0[b_0] < b$ such that $E \vdash M_1.M_2 : \text{cap}[B_0]$ and $E \vdash P : b_0$. There thus exists B_1 and B_2 with $\text{cap}[B_1[B_2]] < \text{cap}[B_0]$, implying $B_1[B_2] < B_0$, such that $E \vdash M_1 : \text{cap}[B_1]$ and $E \vdash M_2 : \text{cap}[B_2]$. We can thus infer first $E \vdash M_2.P : B_2[b_0]$ and then $E \vdash M_1.(M_2.P) : B_1[B_2[b_0]]$. Moreover, $B_1[B_2[b_0]] = (B_1[B_2])[b_0] < B_0[b_0] < b$. This shows that we can derive the desired judgment $E \vdash M_1.(M_2.P) : b$.

Conversely, assume that $E \vdash M_1.(M_2.P) : b$. There exists B_1 and b_1 with $B_1[b_1] \prec b$ such that $E \vdash M_1 : \text{cap}[B_1]$ and $E \vdash M_2.P : b_1$, and therefore there exists B_2 and b_0 with $B_2[b_0] \prec b_1$ such that $E \vdash M_2 : \text{cap}[B_2]$ and $E \vdash P : b_0$. We can thus infer first $E \vdash M_1.M_2 : \text{cap}[B_1[B_2]]$ and then $E \vdash (M_1.M_2).P : (B_1[B_2])[b_0]$. But by employing the fact that “|” and “.” respect \prec : (Lemma 4.7), we deduce that $(B_1[B_2])[b_0] = B_1[B_2[b_0]] \prec B_1[b_1] \prec b$. This shows that we can derive the desired judgment $E \vdash (M_1.M_2).P : b$. \square

Lemma 7.5 Given behavior b , we can construct G implementing b .

Proof. The proof is by structural induction on b , where we perform a case analysis:

$b = \varepsilon$. Choose some ι and let $G = (\{\iota\}, \iota, \{\iota\}, \emptyset)$; then $\text{Acc}(G) = \{\bullet\} = \llbracket \varepsilon \rrbracket$.

$b = \text{put}(\sigma)$. (The cases $b = \text{get}(\sigma)$ and $b = \text{diss}$ are similar.) Choose distinct ι and q and let

$$G = (\{\iota, q\}, \iota, \{q\}, \{(\iota, \text{put}(\sigma), q)\}).$$

Then $\text{Acc}(G) = \{\text{put}(\sigma)\} = \llbracket b \rrbracket$.

$b = \text{fromnow}(\sigma)$. Choose some ι and let $G = (\{\iota\}, \iota, \{\iota\}, \delta)$ where $\delta = \{(\iota, \text{put}(\sigma), \iota), (\iota, \text{get}(\sigma), \iota)\}$. This clearly does the job.

$b = b_1.b_2$. Inductively we can construct automata G_1 implementing b_1 and G_2 implementing b_2 . Let $G_j = (\mathcal{Q}_j, \iota_j, F_j, \delta_j)$ for $j = 1, 2$; wlog. we can assume that \mathcal{Q}_1 and \mathcal{Q}_2 are disjoint. We now construct $G = (\mathcal{Q}, \iota, F, \delta)$ as follows:

$$\begin{aligned} \mathcal{Q} &= \mathcal{Q}_1 \cup \mathcal{Q}_2 \\ \iota &= \iota_1 \\ F &= F_2 \cup (\text{if } \iota_2 \in F_2 \text{ then } F_1 \text{ else } \emptyset) \\ \delta &= \delta_1 \cup \delta_2 \cup \{(q_1, a, q_2) \mid q_1 \in F_1 \text{ and } (\iota_2, a, q_2) \in \delta_2\} \end{aligned}$$

We first prove that $\llbracket b \rrbracket$ is a subset of $\text{Acc}(G)$. So let $tr \in \llbracket b \rrbracket$ be given. We can write $tr = tr_1 \diamond tr_2$ with $tr_1 \in \llbracket b_1 \rrbracket$ and $tr_2 \in \llbracket b_2 \rrbracket$. Therefore $tr_1 \in \text{Acc}(G_1)$ and $tr_2 \in \text{Acc}(G_2)$, implying that there exists $q_1 \in F_1$ and $q_2 \in F_2$ such that $(\iota_1, tr_1, q_1) \in \delta_1^*$ and $(\iota_2, tr_2, q_2) \in \delta_2^*$. We do a case analysis on tr_2 :

- If $tr_2 = \bullet$, then $\iota_2 = q_2 \in F_2$ implying $F_1 \subseteq F$ and therefore $q_1 \in F$. This shows the desired relation $tr \in \text{Acc}(G)$, since from $\iota = \iota_1$ and $tr = tr_1$ and $\delta_1 \subseteq \delta$ we infer that $(\iota, tr, q_1) \in \delta^*$.
- If tr_2 takes the form $a \diamond tr'_2$, there exists q'_2 such that $(\iota_2, a, q'_2) \in \delta_2$ and $(q'_2, tr'_2, q_2) \in \delta_2^*$. By construction of δ , the former relation implies that $(q_1, a, q'_2) \in \delta$. Moreover, it clearly holds that $(\iota_1, tr_1, q_1) \in \delta^*$ and that $(q'_2, tr'_2, q_2) \in \delta^*$. Therefore $(\iota_1, tr_1 \diamond (a \diamond tr'_2), q_2) = (\iota, tr, q_2)$ belongs to δ^* , which since $q_2 \in F_2 \subseteq F$ implies the desired relation $tr \in \text{Acc}(G)$.

Next we prove that $\text{Acc}(G)$ is a subset of $\llbracket b \rrbracket$. So let $tr \in \text{Acc}(G)$ be given. That is, there exists $q \in F$ such that $(\iota, tr, q) \in \delta^*$. There are now two cases to consider:

- If $q \in F_1$ then it is easy to see that $(\iota, tr, q) \in \delta_1^*$. Thus $tr \in \text{Acc}(G_1) = \llbracket b_1 \rrbracket$. Moreover, from the construction of F we infer that $\iota_2 \in F_2$, showing that $\bullet \in \text{Acc}(G_2) = \llbracket b_2 \rrbracket$. Therefore $tr = tr \diamond \bullet$ belongs to $\llbracket b \rrbracket$, as desired.
- If $q \in F_2$ then it is easy to see that there exists $q_1 \in \mathcal{Q}_1$ and $q_2 \in \mathcal{Q}_2$ such that we can write $tr = tr_1 \diamond (a \diamond tr_2)$ with $(\iota_1, tr_1, q_1) \in \delta_1^*$ and $(q_2, tr_2, q) \in \delta_2^*$ and $(q_1, a, q_2) \in \delta$, implying that $q_1 \in F_1$ and that $(\iota_2, a, q_2) \in \delta_2$. This shows that $tr_1 \in \text{Acc}(G_1) = \llbracket b_1 \rrbracket$, and that $(\iota_2, a \diamond tr_2, q) \in \delta_2^*$ which amounts to $a \diamond tr_2 \in \text{Acc}(G_2) = \llbracket b_2 \rrbracket$. This establishes the desired relation $tr \in \llbracket b \rrbracket$.

$b = b_1 \mid b_2$. Inductively we can construct automata G_1 implementing b_1 and G_2 implementing b_2 . Let $G_j = (\mathcal{Q}_j, \iota_j, F_j, \delta_j)$ for $j = 1, 2$. We now construct $G = (\mathcal{Q}, \iota, F, \delta)$ as follows:

$$\begin{aligned} \mathcal{Q} &= \mathcal{Q}_1 \times \mathcal{Q}_2 \\ \iota &= (\iota_1, \iota_2) \\ F &= F_1 \times F_2 \\ \delta &= \{((q_1, q_2), a, (q'_1, q'_2)) \mid (q_1, a, q'_1) \in \delta_1\} \\ &\cup \{((q_1, q_2), a, (q_1, q'_2)) \mid (q_2, a, q'_2) \in \delta_2\} \end{aligned}$$

To reason about G , we first establish

$$(q_1, tr_1, q'_1) \in \delta_1^* \wedge (q_2, tr_2, q'_2) \in \delta_2^* \wedge tr \in \{tr_1\} \parallel \{tr_2\} \implies ((q_1, q_2), tr, (q'_1, q'_2)) \in \delta^*. \quad (1)$$

We prove (1) by induction on the length of tr . If $tr = \bullet$, then also $tr_1 = tr_2 = \bullet$ so $q'_1 = q_1$ and $q'_2 = q_2$ and the claim is trivial.

If tr takes the form $a \diamond tr'$, we can wlog. assume that there exists tr'_1 such that $tr_1 = a \diamond tr'_1$ and $tr' \in \{tr'_1\} \parallel \{tr_2\}$. Thus there exists $q'_1 \in \mathcal{Q}_1$ such that $(q_1, a, q'_1) \in \delta_1$, implying $((q_1, q_2), a, (q'_1, q'_2)) \in \delta$, and $(q'_1, tr'_1, q'_1) \in \delta_1^*$. The induction hypothesis tells us that $((q'_1, q_2), tr', (q'_1, q'_2)) \in \delta^*$, enabling us to arrive at the desired relation $((q_1, q_2), tr, (q'_1, q'_2)) \in \delta^*$.

Next we establish, also by induction on the length of tr , that

$$((q_1, q_2), tr, (q'_1, q'_2)) \in \delta^* \implies \exists tr_1, tr_2: tr \in \{tr_1\} \parallel \{tr_2\} \wedge (q_1, tr_1, q'_1) \in \delta_1^* \wedge (q_2, tr_2, q'_2) \in \delta_2^*. \quad (2)$$

If $tr = \bullet$, then $q'_1 = q_1$ and $q'_2 = q_2$. We can thus choose $tr_1 = tr_2 = \bullet$.

If tr takes the form $a \diamond tr'$, then there exists q such that $((q_1, q_2), a, q) \in \delta$ and $(q, tr', (q'_1, q'_2)) \in \delta^*$. Wlog. we can assume that there exists q'_2 such that $q = (q_1, q'_2)$ and $(q_2, a, q'_2) \in \delta_2$. The induction hypothesis tells us that there exists tr_1 and tr'_2 with $tr' \in \{tr_1\} \parallel \{tr'_2\}$ such that $(q_1, tr_1, q'_1) \in \delta_1^*$ and $(q'_2, tr'_2, q'_2) \in \delta_2^*$. Let $tr_2 = a \diamond tr'_2$. Then clearly $tr \in \{tr_1\} \parallel \{tr_2\}$ and $(q_2, tr_2, q'_2) \in \delta_2^*$, as desired.

We are now ready to embark on the proof that $\llbracket b \rrbracket = \text{Acc}(G)$. If $tr \in \llbracket b \rrbracket$ there exists tr_1 and tr_2 with $tr \in \{tr_1\} \parallel \{tr_2\}$ such that $tr_1 \in \llbracket b_1 \rrbracket = \text{Acc}(G_1)$ and $tr_2 \in \llbracket b_2 \rrbracket = \text{Acc}(G_2)$. There thus exists $q_1 \in F_1$ and $q_2 \in F_2$ such that $(\iota_1, tr_1, q_1) \in \delta_1^*$ and $(\iota_2, tr_2, q_2) \in \delta_2^*$. By (1) we infer that $(\iota, tr, (q_1, q_2)) \in \delta^*$. Since $(q_1, q_2) \in F$, this shows that $tr \in \text{Acc}(G)$. Conversely, assume that $tr \in \text{Acc}(G)$. Thus there exists $(q_1, q_2) \in F$ such that $((\iota_1, \iota_2), tr, (q_1, q_2)) \in \delta^*$. By (2) we infer that there exists tr_1, tr_2 with $tr \in \{tr_1\} \parallel \{tr_2\}$ such that $(\iota_1, tr_1, q_1) \in \delta_1^*$ and $(\iota_2, tr_2, q_2) \in \delta_2^*$. Since $q_1 \in F_1$ and $q_2 \in F_2$, this shows that $tr_1 \in \text{Acc}(G_1) = \llbracket b_1 \rrbracket$ and that $tr_2 \in \text{Acc}(G_2) = \llbracket b_2 \rrbracket$, implying $tr \in \llbracket b \rrbracket$ as desired. \square

Theorem 8.1 Suppose that $E^C \vdash P^C : T^C$, respectively $E^C \vdash M^C : W^C$, is derivable in the system of [10, Sect. 3]. Then $\mathcal{G}(E^C) \vdash \mathcal{G}(P^C) : \mathcal{G}(T^C)$, respectively $\mathcal{G}(E^C) \vdash \mathcal{G}(M^C) : \mathcal{G}(W^C)$, is derivable in our system.

Proof. The proof is by induction on the derivation (in the system of [10]), where we perform a case analysis on the last rule applied and where we employ the following simple observations:

$$\forall T^C : \varepsilon <: \mathcal{G}(T^C) \quad (3)$$

$$\forall T^C : \mathcal{G}(T^C) \mid \mathcal{G}(T^C) <: \mathcal{G}(T^C) \quad (4)$$

$$\forall T^C : \mathcal{G}(T^C) \mid \text{diss is safe} \quad (5)$$

$$\forall T^C : (\mathcal{G}(T^C) \mid \text{diss}) \rightsquigarrow \mathcal{G}(T^C). \quad (6)$$

(Exp n). Our assumption is that $E^C \vdash n : W^C$ because $E^C(n) = W^C$. But then $(\mathcal{G}(E^C))(n) = \mathcal{G}(W^C)$, implying the desired judgment $\mathcal{G}(E^C) \vdash \mathcal{G}(n) : \mathcal{G}(W^C)$.

(Exp ε). Our assumption is that $E^C \vdash \varepsilon : \text{cap}[T^C]$. We must prove that $\mathcal{G}(E^C) \vdash \varepsilon : \text{cap}[\mathcal{G}(T^C) \mid \square]$, and, since by (Exp ε) we have $\mathcal{G}(E^C) \vdash \varepsilon : \text{cap}[\square]$, it is sufficient to show that $\text{cap}[\square] <: \text{cap}[\mathcal{G}(T^C) \mid \square]$ which

amounts to establishing $\square \prec: \mathcal{G}(T^C) \mid \square$. But this follows since for all b we have, employing (3), that $\square[b] = b \equiv \varepsilon \mid b \prec: \mathcal{G}(T^C) \mid b = (\mathcal{G}(T^C) \mid \square)[b]$.

(Exp .). Our assumption is that $E^C \vdash M_1^C.M_2^C : \text{cap}[T^C]$ because $E^C \vdash M_1^C : \text{cap}[T^C]$ and $E^C \vdash M_2^C : \text{cap}[T^C]$. By applying the induction hypothesis we infer that $\mathcal{G}(E^C) \vdash \mathcal{G}(M_1^C) : \text{cap}[\mathcal{G}(T^C) \mid \square]$ and $\mathcal{G}(E^C) \vdash \mathcal{G}(M_2^C) : \text{cap}[\mathcal{G}(T^C) \mid \square]$, which by (Exp Action) yields $\mathcal{G}(E^C) \vdash \mathcal{G}(M_1^C).\mathcal{G}(M_2^C) : \text{cap}[\mathcal{G}(T^C) \mid (\mathcal{G}(T^C) \mid \square)]$. An application of (Exp Subsumption) yields the desired judgment $\mathcal{G}(E^C) \vdash \mathcal{G}(M_1^C.M_2^C) : \text{cap}[\mathcal{G}(T^C) \mid \square]$ provided we can establish $\mathcal{G}(T^C) \mid (\mathcal{G}(T^C) \mid \square) \prec: \mathcal{G}(T^C) \mid \square$ which amounts to showing that for all b we have $\mathcal{G}(T^C) \mid \mathcal{G}(T^C) \mid b \prec: \mathcal{G}(T^C) \mid b$. But this follows from (4).

(Exp In). Our assumption is that $E^C \vdash \text{in } M^C : \text{cap}[T^C]$ because $E^C \vdash M^C : \text{amb}[S^C]$. By applying the induction hypothesis we infer that $\mathcal{G}(E^C) \vdash \mathcal{G}(M^C) : \text{amb}[\mathcal{G}(S^C)]$ which by (Exp In) yields $\mathcal{G}(E^C) \vdash \text{in } \mathcal{G}(M^C) : \text{cap}[\square]$. An application of (Exp Subsumption) yields the desired judgment $\mathcal{G}(E^C) \vdash \mathcal{G}(\text{in } M^C) : \text{cap}[\mathcal{G}(T^C) \mid \square]$, provided we can show that $\square \prec: \mathcal{G}(T^C) \mid \square$. But this follows as in the case (Exp ε).

(Exp Out). Similar to the case (Exp In).

(Exp Open). Our assumption is that $E^C \vdash \text{open } M^C : \text{cap}[T^C]$ because $E^C \vdash M^C : \text{amb}[T^C]$. By applying the induction hypothesis we infer that $\mathcal{G}(E^C) \vdash \mathcal{G}(M^C) : \text{amb}[\mathcal{G}(T^C)]$ which by (Exp Open) yields the desired judgment $\mathcal{G}(E^C) \vdash \mathcal{G}(\text{open } M^C) : \text{cap}[\mathcal{G}(T^C) \mid \square]$.

(Proc Action). Our assumption is that $E^C \vdash M^C.P^C : T^C$ because $E^C \vdash M^C : \text{cap}[T^C]$ and $E^C \vdash P^C : T^C$. By applying the induction hypothesis we infer that $\mathcal{G}(E^C) \vdash \mathcal{G}(M^C) : \text{cap}[\mathcal{G}(T^C) \mid \square]$ and $\mathcal{G}(E^C) \vdash \mathcal{G}(P^C) : \mathcal{G}(T^C)$, which by (Proc Action) yields $\mathcal{G}(E^C) \vdash \mathcal{G}(M^C).\mathcal{G}(P^C) : \mathcal{G}(T^C) \mid \mathcal{G}(T^C)$. Using (4) and the rule (Proc Subsumption), this yields the desired judgment $\mathcal{G}(E^C) \vdash \mathcal{G}(M^C.P^C) : \mathcal{G}(T^C)$.

(Proc Amb). Our assumption is that $E^C \vdash M^C[P^C] : S^C$ because $E^C \vdash M^C : \text{amb}[T^C]$ and $E^C \vdash P^C : T^C$. By applying the induction hypothesis we infer that $\mathcal{G}(E^C) \vdash \mathcal{G}(M^C) : \text{amb}[\mathcal{G}(T^C)]$ and $\mathcal{G}(E^C) \vdash \mathcal{G}(P^C) : \mathcal{G}(T^C)$, and therefore also $\mathcal{G}(E^C) \vdash \mathcal{G}(P^C) \mid \text{coopen } \mathcal{G}(M^C) : \mathcal{G}(T^C) \mid \text{diss}$. Using (5) and (6) we can thus apply (Proc Amb) to get the judgement $\mathcal{G}(E^C) \vdash \mathcal{G}(M^C)[\mathcal{G}(P^C) \mid \text{coopen } \mathcal{G}(M^C)] : \varepsilon$ which by (3) yields $\mathcal{G}(E^C) \vdash \mathcal{G}(M^C[P^C]) : \mathcal{G}(S^C)$.

(Proc Res). Our assumption is that $E^C \vdash (\nu n : \text{amb}[T^C]).P^C : S^C$ because $E^C, n : \text{amb}[T^C] \vdash P^C : S^C$. By applying the induction hypothesis we infer that $\mathcal{G}(E^C), n : \text{amb}[\mathcal{G}(T^C)] \vdash \mathcal{G}(P^C) : \mathcal{G}(S^C)$ which by (Proc Res) yields $\mathcal{G}(E^C) \vdash (\nu n : \text{amb}[\mathcal{G}(T^C)])\mathcal{G}(P^C) : \mathcal{G}(S^C)$.

(Proc Zero). Our assumption is that $E^C \vdash \mathbf{0} : T^C$. We must prove that $\mathcal{G}(E^C) \vdash \mathcal{G}(\mathbf{0}) : \mathcal{G}(T^C)$, but since $\mathcal{G}(E^C) \vdash \mathbf{0} : \varepsilon$ this follows from (3).

(Proc Par). Our assumption is that $E^C \vdash P^C \mid Q^C : T^C$ because $E^C \vdash P^C : T^C$ and $E^C \vdash Q^C : T^C$. By applying the induction hypothesis we infer that $\mathcal{G}(E^C) \vdash \mathcal{G}(P^C) : \mathcal{G}(T^C)$ and that $\mathcal{G}(E^C) \vdash \mathcal{G}(Q^C) : \mathcal{G}(T^C)$, which by (Proc Par) yields $\mathcal{G}(E^C) \vdash \mathcal{G}(P^C) \mid \mathcal{G}(Q^C) : \mathcal{G}(T^C) \mid \mathcal{G}(T^C)$. Using (4) and the rule (Proc Subsumption), this yields the judgment $\mathcal{G}(E^C) \vdash \mathcal{G}(P^C \mid Q^C) : \mathcal{G}(T^C)$.

(Proc Repl). Our assumption is that $E^C \vdash !P^C : T^C$ because $E^C \vdash P^C : T^C$. By applying the induction hypothesis we infer that $\mathcal{G}(E^C) \vdash \mathcal{G}(P^C) : \mathcal{G}(T^C)$, and due to (4) we can apply (Proc Repl) to infer the desired judgment $\mathcal{G}(E^C) \vdash !\mathcal{G}(P^C) : \mathcal{G}(T^C)$.

(Proc Input). Our assumption is that $E^C \vdash (n_1 : W_1^C, \dots, n_k : W_k^C).P^C : \times(W_1^C, \dots, W_k^C)$ because

$$E^C, n_1 : W_1^C, \dots, n_k : W_k^C \vdash P^C : \times(W_1^C, \dots, W_k^C).$$

Let $\sigma = \times(\mathcal{G}(W_1^C), \dots, \mathcal{G}(W_k^C))$, thus $\mathcal{G}(\times(W_1^C, \dots, W_k^C)) = \text{fromnow}(\sigma)$. By applying the induction hypothesis we infer that $\mathcal{G}(E^C), n_1 : \mathcal{G}(W_1^C), \dots, n_k : \mathcal{G}(W_k^C) \vdash \mathcal{G}(P^C) : \text{fromnow}(\sigma)$ which by (Proc Input) yields the judgement $\mathcal{G}(E^C) \vdash (n_1 : \mathcal{G}(W_1^C), \dots, n_k : \mathcal{G}(W_k^C)).\mathcal{G}(P^C) : \text{get}(\sigma).\text{fromnow}(\sigma)$. Since clearly

$\text{get}(\sigma).\text{fromnow}(\sigma) \prec \text{fromnow}(\sigma)$, we can apply (Proc Subsumption) to arrive at the desired judgment

$$\mathcal{G}(E^C) \vdash \mathcal{G}((n_1 : W_1^C, \dots, n_k : W_k^C).P^C) : \mathcal{G}(\times(W_1^C, \dots, W_k^C)).$$

(Proc Output). Our assumption is that $E^C \vdash \langle M_1^C, \dots, M_k^C \rangle : \times(W_1^C, \dots, W_k^C)$ because for all $i \in \{1 \dots k\}$ we have $E^C \vdash M_i^C : W_i^C$. Let $\sigma = \times(\mathcal{G}(W_1^C), \dots, \mathcal{G}(W_k^C))$, so that $\mathcal{G}(\times(W_1^C, \dots, W_k^C)) = \text{fromnow}(\sigma)$. By applying the induction hypothesis we infer that for all $i \in \{1 \dots k\}$ it holds that $\mathcal{G}(E^C) \vdash \mathcal{G}(M_i^C) : \mathcal{G}(W_i^C)$ which by (Proc Output) yields the judgement

$$\mathcal{G}(E^C) \vdash \langle \mathcal{G}(M_1^C), \dots, \mathcal{G}(M_k^C) \rangle : \text{put}(\sigma).$$

Since clearly $\text{put}(\sigma) \prec \text{fromnow}(\sigma)$ we can apply (Proc Subsumption) to arrive at the desired judgment $\mathcal{G}(E^C) \vdash \mathcal{G}(\langle M_1^C, \dots, M_k^C \rangle) : \mathcal{G}(\times(W_1^C, \dots, W_k^C))$. □