# From CML to Process Algebras

Flemming Nielson, Hanne Riis Nielson
Computer Science Department, Aarhus University
Ny Munkegade, DK-8000 Aarhus C, Denmark

e-mail:{fnielson,hrnielson}@daimi.aau.dk

March 1993

**Abstract**

Reppy's language CML extends Standard ML of Milner et al. with primitives for communication. It thus inherits a notion of strong polymorphic typing and may be equipped with a structural operational semantics. We formulate an effect system for statically expressing the communication behaviours of CML programs as these are not otherwise reflected in the types.

We then show how types and behaviours evolve in the course of computation: types may decrease and behaviours may loose alternatives as well as decrease. It will turn out that the syntax of behaviours is rather similar to that of a process algebra; our main results may therefore be viewed as regarding the semantics of a process algebra as an *abstraction* of the semantics of an underlying programming language. This establishes a new kind of connection between "realistic" concurrent programming languages and "theoretical" process algebras.

# 1 Introduction

One trend in the research on process algebras is to extend them with "higher-order" features somewhat analogous to the "higher-order" role that functions play in functional languages. Some approaches allow passing labels or ports, e.g. [13], whereas others allow passing processes, e.g. [21, 23]. Sometimes this leads to hybrid calculi that contain the syntax of a process algebra as well as that of the $\lambda$-calculus, e.g. [5, 14, 9]. Putting more emphasis on the functional features, another approach is to extend a "realistic" functional language with primitives for communication. Good examples include CML [18, 19, 13] and Facile [7] but also Concurrent Clean [16] may be viewed in this way. We refer to [10] for a much more detailed survey of some of these issues.

We follow the latter approach and base ourselves on Reppy's language CML. It is an extension of Standard ML with primitives for communication; among other things this allows channels to be created and processes to be forked and then processes may send and receive values over channels. Since CML is an extension of Standard ML it inherits a notion of strong typing unlike some other approaches. However, the types are very close to those of Standard ML and therefore do not contain much information about the communication that takes place during computation. Following [14, 9] we believe that it is desirable with some type-like "formula" that gives a concise summary of the possible communication behaviours. Our approach deviates from [14, 9] in separating the type and communication information by using the notion of effect system previously developed for functional languages, e.g. [11, 20]. Section 2 gives a presentation of this system.

Both [19] and [1] give a structural operational semantics for CML. As is usual the types do not influence the semantics but for the purpose of proofs it may be desirable to label the transition relation with additional book-keeping details (and to retain some type information in the expression). The main difference between [19] and [1] is that the latter is a traditional operational semantics whereas the former uses the notion of "evaluation context" in order to present the rules more concisely and in order to facilitate proofs. In Section 3 we present a definition close to that of [19] but with additional book-keeping details; in keeping with tradition the types and behaviours do not influence the semantics.

The impact of the operational semantics on types and behaviours emerges when showing "subject reduction" and related results. Actually, types may *decrease* in the course of computation and this phenomenon also arose in [3] in the context of modelling object-oriented programming. In a similar way the behaviours may decrease in the course of computation but additionally certain alternatives may *disappear* due to the choices made during computation. It is instructive to regard this combined decreasing and disappearance of behaviours as an operational semantics for behaviours. Since behaviours syntactically resemble process algebras (e.g. the one in [8]) this suggests the viewpoint that the semantics of a process algebra is an *abstraction* of the semantics of an underlying programming language. This is quite unlike previous attempts to relate languages like CML to process algebras where programs from CML are directly translated into primitives of some given process algebra or vice versa. Section 4 provides the precise formulations of the results we have to offer as well as overviews of the proofs.

We finish with prospects for future research and concluding remarks in Section 5. In Appendix A we briefly discuss variations on the system presented here and in Appendix B we provide full details of the proofs.

# 2 CML with Behaviours of Communication

We follow [19, 1] in embedding the essential features of CML into a small fragment of Standard ML. For simplicity we restrict the attention to a monomorphic fragment and we take care to structure the syntax in a way that facilitates adding new constructs as the need arises.

The syntax of expressions $e \in \mathbf{Exp}$ and weakly[1] evaluated expressions $w \in \mathbf{Exp}$ is given by:

$$\begin{array}{rl} e ::= & w \mid e_1\ e_2 \mid \texttt{let}\ i = e_1\ \texttt{in}\ e_2 \mid \texttt{rec}\ i_0(i_1) : t \Rightarrow e \\ & \mid\ \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : t \\ w ::= & c : t \mid i \mid \texttt{fn}\ i : t \Rightarrow e \mid \cdots \end{array}$$

They are defined by mutual recursion and include constants with an explicit

[1]This terminology is consistent with the weak normal forms of [17].

monotype, identifiers[2] $i \in \mathbf{Ident}$ (unspecified), function abstraction, application, `let`-abstraction but without any polymorphism, recursive definitions, and conditional with an explicit monotype indicating the type of the recursive function. In the next section we shall need to introduce additional weakly evaluated expressions corresponding to the intermediate results that arise during computation.

The syntax of constants $c \in \mathbf{Const}$ is given by:

$$\begin{array}{rl} c ::= & () \mid \texttt{true} \mid \texttt{false} \mid n \\ \mid & \texttt{pair} \mid \texttt{fst} \mid \texttt{snd} \\ \mid & \texttt{nil} \mid \texttt{cons} \mid \texttt{hd} \mid \texttt{tl} \mid \texttt{isnil} \\ \mid & \texttt{send} \mid \texttt{receive} \mid \texttt{choose} \mid \texttt{noevent} \\ \mid & \texttt{wrap} \mid \texttt{sync} \mid \texttt{fork} \mid \texttt{channel} \end{array}$$

This includes the element () of the unit type, the booleans `true` and `false`, and numerals $n \in \mathbf{Num}$ (unspecified). For products we write $\texttt{pair}\ e_1\ e_2$ for $(e_1, e_2)$ in order to reduce the amount of syntactic sugar and we then use `fst` and `snd` to select components. Similarly for lists we write $\texttt{cons}\ e_1\ (\cdots(\texttt{cons}\ e_n\ \texttt{nil})\cdots)$ for $[e_1, \cdots, e_n]$ and we select components using `hd` and `tl` and test for emptiness using `isnil`.

Turning to the concurrency primitives we may send values over channels, receive values over channels, choose between a list of computations (and writing `noevent` for $\texttt{choose}\ \texttt{nil}$), and modify a value that is communicated by applying a function to it. Actually, these primitives construct "delayed" communications that may be enacted using synchronization. Finally, we may fork a new process to the pool of processes and we may allocate a new free channel for communication.

For types $t \in \mathbf{Type}$ we take:

$$\begin{array}{rl} t ::= & \texttt{unit} \mid \texttt{bool} \mid \texttt{int} \mid tv \mid t_1 \times t_2 \mid t\ \texttt{list} \\ \mid & t_1 \rightarrow^b t_2 \mid t\ \texttt{chan}\ r \mid t\ \texttt{com}\ b \end{array}$$

[2]It is customary to take $w ::= i$ rather than $e ::= i$ but for the purposes of this section this does not matter. Since we only rewrite closed expressions in the next section it also does not matter for the remainder of the paper.

As in Standard ML we have three base types, type variables $tv \in \mathbf{TyVar}$ (e.g. $\tau, \tau', \tau_1$) and products and list. Concerning functions we use a superscript behaviour $b \in \mathbf{Beh}$ for indicating the communication that will take place when the function is executed; the precise details follow shortly. Much as in CML we have a type for channels over which values of a given type may be communicated; to allow some separation among the identity of channels we indicate the specific region where the channel is allocated. For regions $r \in \mathbf{Reg}$ we take:

$$r := i \mid r_1 + r_2 \mid rv$$

A region will describe a non-empty set of "program points" and we shall occasionally need region variables $rv \in \mathbf{RegVar}$ (e.g. $\rho, \rho', \rho_1$) However, it would be possible to dispense with regions throughout without invalidating the results of the paper. Also as in CML we have a type for a "delayed" communication yielding a result of a certain type; unlike CML we have added a behaviour for indicating the communication that will take place when the "delayed" commununication is enacted.

Finally, behaviours $b \in \mathbf{Beh}$ are given by:

$$\begin{array}{rcl} b ::= & \epsilon \mid r!t \mid r?t \mid t \text{ CHAN } r \mid t \text{ FORK } b \\ \mid & b_1; b_2 \mid b_1 + b_2 \mid \text{REC } bv.\ b \mid bv \end{array}$$

The behaviours include primitive constructs for describing "no communication", sending a value of some type over a channel allocated in a certain region, receiving a value, allocating a channel, and forking a new process of a given type and with a given behaviour when executed. We use semi-colon to express that one behaviour takes place before another and we use plus to express that either the first behaviour takes place or the second does. For recursive functions we need a behaviour REC $bv.\ b$ for expressing a behaviour that is as given by $b$ provided that recursive calls are as given by $bv \in \mathbf{BehVar}$ (e.g. $\beta, \beta'.\beta_1$).

**Example 2.1:** The `map` function for mapping a function down a list of elements may be defined by:

```
rec map f ⇒ fn xs ⇒ if isnil xs then nil
                    else cons (f (hd xs)) (map f (tl xs))
```

where we have dispensed with the ": $t$" at a number of places. Its overall type is

$$(\texttt{int} \rightarrow^{\epsilon} \texttt{bool}) \rightarrow^{\epsilon} \texttt{intlist} \rightarrow^{\epsilon} \texttt{boollist}$$

A parallel version may be defined by:

```
rec mappar f ⇒ fn xs ⇒ if isnil xs then nil
                        else let ch = channel ()
                             in fork (fn d ⇒ sync
                                          (send (ch, f (hd xs))));
                             let ys = mappar f (tl xs)
                             in sync (wrap (receive ch,
                                            fn y ⇒ cons y ys))
```

where we write $(e_1, e_2)$ for `pair` $e_1$ $e_2$ and $e_1; e_2$ for `snd`$(e_1, e_2)$ . Its overall type is

$$(\texttt{int} \rightarrow^{\epsilon} \texttt{bool}) \rightarrow^{\epsilon} \texttt{int list} \rightarrow^{b} \texttt{bool list}$$

where $b = \text{REC } \beta.\ \epsilon + ((\texttt{bool CHAN m});\ (\texttt{bool FORK m!bool}); \beta; (\texttt{m?bool}))$ and where we assume that the region corresponding to the channel is `m`. □

## Well-typing

We shall say that an expression $e$ has type $t$ and behaviour $b$, written

$$tenv \vdash e \mid t \ \&\ b,$$

whenever the type of $e$ is $t$ in the usual sense and evaluation of $e$ gives rise to the communication behaviour $b$. As usual *tenv* is a type environment, i.e. a finite list of pairs of identifiers and types, giving the types of free variables; since CML is an eager language there is no effect associated with accessing an identifier and therefore the type environment does not contain any behaviour component (except embedded within the types). For constants our syntax

prescribes an explicit monotype to be given; we use the polytypes of Figure 1 to restrict the choice of monotypes. Only three primitives involve functions with a non-trivial behaviour: `sync` for enacting a "delayed" computation, `fork` for forking a new process and `channel` for allocating a new channel.

The details of the type inference for expressions are given by the axioms and rules of Figure 2. We already explained the axioms for identifiers and constants. For function abstraction the resulting type and behaviour indicate that no communication takes place when constructing the function abstraction but only when the function is executed. For application the overall behaviour expresses eager left-to-right evaluation: first the expression in function position in evaluated to a function abstraction, then the argument is evaluated and finally the function is applied to the argument. We do not require equality between the type of the actual parameter and the type of the formal parameter but merely that the type of the actual parameter is a sub-type of the type of the formal parameter. As illustrated in Appendix A this is useful for allowing a function expressing mild restrictions on the argument, e.g. that it only communicates over channels in certain regions, to be applied to a concrete argument with a very specific communication behaviour. The definition of the sub-typing relation is given below. The rule for `let`-abstraction is rather straightforward. The rule for recursive functions is much as the rule for function abstraction except that we need to extend the type environment with assumptions about the recursive function and we only require the type and behaviour of the body to be sub-types and sub-behaviours of the corresponding parts of the assumptions. The above example illustrates that REC-behaviours may be "deeply" nested within the type of the recursive function[3]. Finally, the rule for conditional allows the types of the branches to be dissimilar and only requires them to be sub-types of a common type. To require equality would invalidate the subject reduction property proved in Section 4.

**Fact 2.2:** (Unique Typing) If $tenv \vdash e \mid t_1 \;\&\; b_1$ and $tenv \vdash e \mid t_2 \;\&\; b_2$ then $t_1 = t_2$ and $b_1 = b_2$ □

This is a consequence of not having an explicit subsumption rule but instead integrating it with the other rules.

[3]In the notation of Figure 2 there need not be any occurrences of REC in $b$ even though there may be occurrences in $t$.

| $c$ | TypeOf($c$) |
|---|---|
| `()` | $\texttt{unit}$ |
| `true` | $\texttt{bool}$ |
| `false` | $\texttt{bool}$ |
| $n$ | $\texttt{int}$ |
| `pair` | $\forall\tau_1,\tau_2.\ \tau_1 \rightarrow^{\epsilon} \tau_2 \rightarrow^{\epsilon} \tau_1 \times \tau_2$ |
| `fst` | $\forall\tau_1,\tau_2.\ \tau_1 \times \tau_2 \rightarrow^{\epsilon} \tau_1$ |
| `snd` | $\forall\tau_1,\tau_2.\ \tau_1 \times \tau_2 \rightarrow^{\epsilon} \tau_2$ |
| `nil` | $\forall\tau.\ \tau\ \texttt{list}$ |
| `cons` | $\forall\tau.\ \tau \rightarrow^{\epsilon} \tau\ \texttt{list} \rightarrow^{\epsilon} \tau\ \texttt{list}$ |
| `hd` | $\forall\tau.\ \tau\ \texttt{list} \rightarrow^{\epsilon} \tau$ |
| `tl` | $\forall\tau.\ \tau\ \texttt{list} \rightarrow^{\epsilon} \tau\ \texttt{list}$ |
| `isnil` | $\forall\tau.\ \tau\ \texttt{list} \rightarrow^{\epsilon} \texttt{bool}$ |
| `send` | $\forall\rho,\tau.\ (\tau\ \texttt{chan}\ \rho) \times \tau \rightarrow^{\epsilon} (\tau\ \texttt{com}\ (\rho!\tau))$ |
| `receive` | $\forall\rho,\tau.\ (\tau\ \texttt{chan}\ \rho) \rightarrow^{\epsilon} (\tau\ \texttt{com}\ (\rho?\tau))$ |
| `choose` | $\forall\beta,\tau.\ (\tau\ \texttt{com}\ \beta)\ \texttt{list} \rightarrow^{\epsilon} (\tau\ \texttt{com}\ \beta)$ |
| `noevent` | $\forall\beta,\tau.\ (\tau\ \texttt{com}\ \beta)$ |
| `wrap` | $\forall\beta_1,\beta_2,\tau_1,\tau_2.\ (\tau_1\ \texttt{com}\ \beta_1) \times (\tau_1 \rightarrow^{\beta_2} \tau_2) \rightarrow^{\epsilon} (\tau_2\ \texttt{com}\ (\beta_1;\beta_2))$ |
| `sync` | $\forall\beta,\tau.\ (\tau\ \texttt{com}\ \beta) \rightarrow^{\beta} \tau$ |
| `fork` | $\forall\beta,\tau.\ (\texttt{unit} \rightarrow^{\beta} \tau) \rightarrow^{\tau\ \textsc{fork}\ \beta} \texttt{unit}$ |
| `channel` | $\forall\iota,\tau.\ \texttt{unit} \rightarrow^{\tau\ \textsc{chan}\ \iota} (\tau\ \texttt{chan}\ \iota)$ |

Figure 1: Types of Primitives

From a pragmatic point of view it might be better to add the "rearrangement" rule

$$\frac{tenv \vdash e \mid t_1\ \&\ b_1}{tenv \vdash e \mid t_2\ \&\ b_2} \quad \text{If } t_1 \equiv t_2 \text{ and } b_1 \equiv b_2$$

where $\equiv$ are the equivalences introduced shortly. Then Fact 2.2 should be changed to use $\equiv$ instead of $=$. This would allow to prove that in the sequential subset of CML all behaviours may be taken to be $\epsilon$.

## Sub-typing

Since types involve regions as well as behaviours the sub-typing relation must involve a sub-region relation and a sub-behaviour relation. These relations

may be defined by axioms and inference rules and have some important similarities (as well as important differences). To save repetition and to help demonstrating that they constitute the "right" collection we shall organize their presentation with diligence.

$$tenv \vdash c : t \mid t \;\&\; \epsilon \qquad \text{if } \mathrm{TypeOf}(c) \succ t$$

$$tenv \vdash i \mid t \;\&\; \epsilon \qquad \text{if } tenv(i) = t$$

$$\frac{tenv[i \mapsto t] \vdash e \mid t' \;\&\; b}{tenv \vdash \mathtt{fn}\ i : t \ \Rightarrow\ e \mid t \to^b t' \;\&\; \epsilon}$$

$$\frac{tenv \vdash e_1 \mid t \to^b t' \;\&\; b_1 \quad tenv \vdash e_2 \mid t^- \;\&\; b_2}{tenv \vdash e_1\ e_2 \mid t' \;\&\; b_1; b_2; b} \qquad \text{if } t^- \leq t$$

$$\frac{tenv \vdash e_1 \mid t_1 \;\&\; b_1 \quad tenv[i \mapsto t_1] \vdash e_2 \mid t_2 \;\&\; b_2}{tenv \vdash \mathtt{let}\ i = e_1\ \mathtt{in}\ e_2 \mid t_2 \;\&\; b_1; b_2}$$

$$\frac{tenv[i_0 \mapsto t_1 \to^b t][i_1 \mapsto t_1] \vdash e \mid t^- \;\&\; b^-}{tenv \vdash \mathtt{rec}\ i_0(i_1) : t_1 \to^b t \Rightarrow e \mid t_1 \to^b t \;\&\; \epsilon} \qquad \text{if } t^- \leq t \text{ and } b^- \leq b$$

$$\frac{tenv \vdash e \mid \mathtt{bool} \;\&\; b \quad tenv \vdash e_1 \mid t_1 \;\&\; b_1 \quad tenv \vdash e_2 \mid t_2 \;\&\; b_2}{tenv \vdash \mathtt{if}\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : t \mid t \;\&\; b; (b_1 + b_2)} \qquad \text{if } t_1 \leq t \text{ and } t_2 \leq t$$

Figure 2: Type Inference

We begin with regions. Intuitively, $r_1 \leq r_2$ is to mean that the set of identifiers listed in $r_1$ is a subset of those listed in $r_2$. Formally, this may be axiomatized as shown in Figure 3. The first 5 axioms and rules simply state that $\leq$ is a preorder and that $\equiv$ is the associated equivalence. The last 4 axioms and rules state that $+$ is a least upper bound operator (modulo the equivalence). The two axioms involving $\leq$ are standard but the inference rule and the axiom involving $\equiv$ are usually replaced by a rule that allows one to infer $r_1 + r_2 \leq r$ from $r_1 \leq r$ and $r_2 \leq r$. Luckily, the two formulations are equivalent in the presence of the other rules and axioms but we prefer the choice made since the structural rule is typical of the rules we shall need for behaviours and types. The notion of polarity is explained below.

Turning to types we once more need to state that $\leq$ is a preorder and $\equiv$ is the associated equivalence. The details of this are as for regions and are therefore not repeated in Figure 4. Next comes a structural rule for each type constructor. To summarize these succinctly we use the notion of polarity. There are three polarities: $\oplus$ for a covariant or monotonic position,

$\ominus$ for a contravariant or antimonotonic position and $\odot$ for a mixed co- and contravariant position. The examples given in Figures 3 and 4 should make the intention clear[4]. The definitions are in good accord with the literature on sub-typing.

Many of the rules and axioms for behaviours in Figure 5 follow the pattern seen already. On top of this we have distribution laws for ' $+$ , and for '; we have an associative law and two axioms stating that $\epsilon$ is a neutral element. For recursion we have axioms for $\alpha$-conversion, one-level unfolding and a simple structural rule.

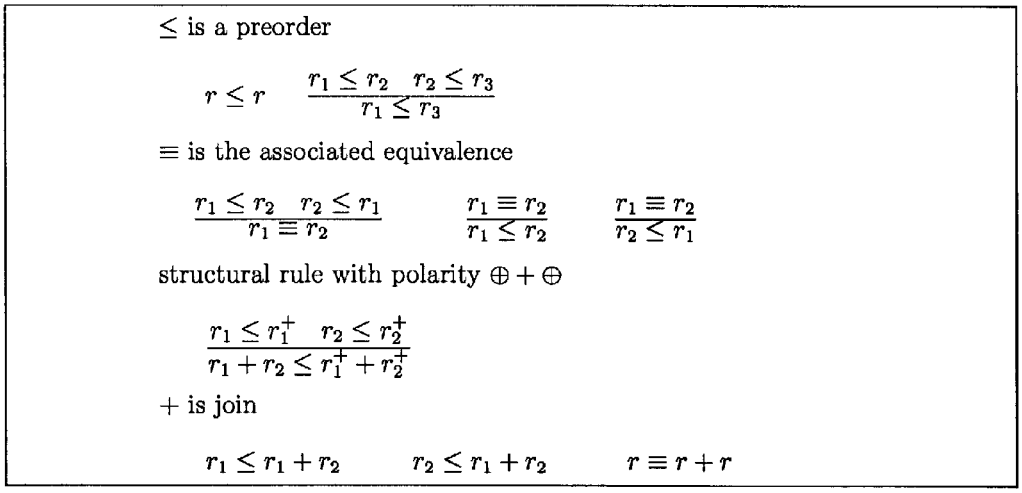


Figure 3: Coercion Rules for Regions

# 3 Dynamic Semantics of CML

We now present a structural operational semantics for the eager left-to-right evaluation of CML. The formulation is close in spirit to [19] and amounts to the definition of three transition relations: one for *sequential* evaluation, one for *concurrent* evaluation and to handle the `sync` operator we also need

[4]One can be more formal as follows. Write $t[\oplus]t'$ for $t \leq t'$, $t[\ominus]t'$ for $t' \leq t$, and $t[\odot]t'$ for $t \equiv t'$. A type constructor $\varphi$ has polarity $\varphi(p_1, \ldots, p_n)$ if and only if the structural inference rule says that $\varphi(t_1, \ldots, t_n)[\oplus]\varphi(t'_1, \ldots, t'_n)$ follows from $t_1[p_1]t_1, \ldots, t_n[p_n]t'_n$.

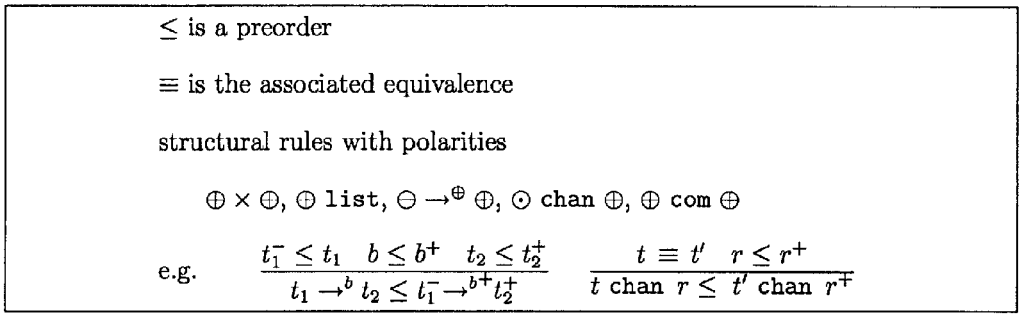


Figure 4: Coercion Rules for Types

a transition system for *matching* the communications against one another. Some differences include the treatment of $\delta$-reduction and the choice of book-keeping details[5] that label the transition relations and the expressions.

## Sequential evaluation

We begin with the sequential evaluation of expressions. This encompasses all features of CML *except* the `channel, fork` and `sync` primitives; these were the primitives listed in Figure 1 that did not have an $\epsilon$-behaviour associated with the function space. The definition of the transition relation is given in Figure 6 and makes use of a number of auxiliary concepts. A central concept is that of an *evaluation context* $E$. It may be defined inductively by:

$$E ::= [] \mid E\ e \mid w\ E \mid \texttt{let}\ i = E\ \texttt{in}\ e \mid \texttt{if}\ E\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : t$$

Here [] is an empty context or a "hole"; so in general $E$ describes an expression with precisely one hole in it. We then write $E[e]$ for the expression that is like $E$ except that the hole is replaced by $e$. The definition of $E$ is responsible for enforcing the eager left-to-right evaluation. As an example consider application, i.e. $e_1\ e_2$. The presence of $E\ e$ means that the function part, i.e. $e_1$, may always be evaluated whereas the presence of $w\ E$ means

[5]For the purposes of this section it would be straightforward to simplify the amount of book-keeping details, but in the next section we would then have a rather cumbersome task of re-introducing them or else inventing other mediating concepts.

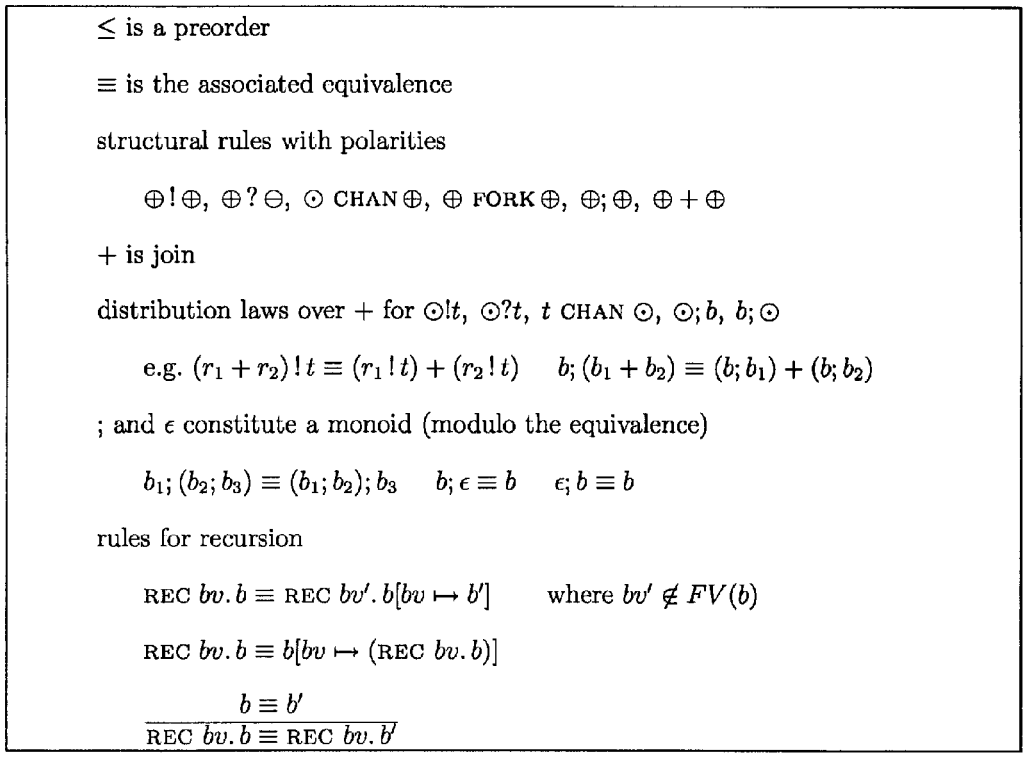


Figure 5: Coercion Rules for Behaviours

that the argument part, i.e. $e_2$, may only be evaluated after the function part has been (weakly) evaluated (e.g. to a function abstraction).

Most of the axioms of Figure 6 are now straightforward. The first axiom expresses the one-level unfolding of a (type correct) recursive definition. For this we make use of the standard notation $e_1[e_2/i]$ for substituting $e_2$ for all free occurrences of $i$ in $e_1$; when doing so care must be taken to rename bound identifiers in $e_1$ so as to avoid the capture of free identifiers in $e_2$. Comparing with the $E[e]$ notation we could thus write $E[e/[]]$ for $E[e]$ but in this case the definition of $E$ ensures that there is no risk of capturing free identifiers in $e$. The second axiom is $\beta$-reduction and the use of $w$, rather than $e$, in the argument position ensures that we obtain call-by-value semantics. The third axiom is consistent with the view that `let` $i = e_1$ `in` $e_2$ is semantically equivalent to (`fn` $i \Rightarrow e_2$) $e_1$. The fourth axiom is actually an abbreviation for two axioms describing the evaluation of the conditional depending on the

outcome of the text.

The fifth axiom describes the $\delta$-reductions for the primitive constructs of CML. The details are listed in Figure 7 and once again make use of a number of auxiliary concepts. To record the piecemeal evaluation of constants, as in the intended reduction sequence

$$\begin{array}{l}\texttt{pair}\ (1+2)(3+4) \rightarrow^{+} \texttt{pair}\ 3\ (3+4) \rightarrow \\ \langle\texttt{pair}\ 3\rangle(3+4) \rightarrow^{+} \langle\texttt{pair}\ 3\rangle\ 7 \rightarrow \\ \langle\texttt{pair}\ 3\ 7\rangle\end{array}$$

we need to extend the syntax of weakly evaluated expressions with new "constants" $\langle\texttt{pair}\ 3\rangle$ and $\langle\texttt{pair}\ 3\ 7\rangle$. Formally, we proceed as follows. Let $c$ be one of the constants of Figure 1 and let $n$ be maximal such that TypeOf($c$) may be written as $t'_1 \rightarrow^{\epsilon} \ldots \rightarrow^{\epsilon} t'_{n+1}$ where we have dispensed with the universal quantifiers. For each monotype instance $t = t_1 \rightarrow^{\epsilon} \ldots \rightarrow^{\epsilon} t_{n+1}$ of TypeOf($c$) we then add the weakly evaluated constants $\langle c : tw_1\rangle, \ldots, \langle c : tw_1 \ldots w_n\rangle$ to the syntax of weakly evaluated expressions, i.e.

$$w \ ::= \ldots \mid \langle c : tw_1\rangle \mid \ldots \mid \langle c : tw_1 \ldots w_n\rangle$$

We also add a new typing rule:

$$\frac{tenv \vdash c \mid t_1 \rightarrow^{\epsilon} \cdots \rightarrow^{\epsilon} t_{n+1}\&\epsilon\ tenv \vdash w_1 \mid t_1^{-}\&\epsilon\ \cdots\ tenv \vdash w_i \mid t_i^{-}\&\epsilon}{tenv \vdash \langle c : t_1 \rightarrow^{\epsilon} \cdots \rightarrow^{\epsilon} t_{n+1} w_1 \cdots w_i\rangle \mid t_{i+1} \rightarrow^{\epsilon} \cdots \rightarrow^{\epsilon} t_{n+1}\&\epsilon}$$

where $i \leq n$ and $t_j^{-} \leq t_j$ for $j \leq i$

Returning to Figure 7 most of the $\delta$-"rules" are rather straightforward. A small notational point is that it might have been preferable to use curried constants as this would allow writing e.g. $\langle\texttt{wrap} : tw_1w_2\rangle$ instead of the more cumbersome $\langle\texttt{wrap} : t'\langle\texttt{pair} : t''w_1w_2\rangle\rangle$; in examples we shall sometimes use this more readable notation and also dispense with the ' $: t'$. It is worth pointing out that we deviate from [19] in not making a *meta-syntactic* distinction between weakly evaluated expressions of type $t$ com $b$ and those not of a type on this form; we simply use the meta-variable $w$ whereas [19] uses $ev$ and $v$. More importantly we deviate form [19] in not requiring $\delta_{\rightarrow}$

to be total, e.g. we allow that we have no $\delta$-"rule" for `hd nil`. We regard it overly restrictive to exclude this situation and instead introduce a new set $\delta_{\not\rightarrow}$ for characterizing the dynamically stuck configurations. It may be defined by

$(\texttt{hd}\ :t, \texttt{nil}) \in \delta_{\not\rightarrow}$
$(\texttt{tl}\ :t, \texttt{nil}) \in \delta_{\not\rightarrow}$

$$E\,[\texttt{rec}\ i_0(i_1) : t_1 \rightarrow^b t_2 \Rightarrow e] \\ \quad \rightarrow E[(\texttt{fn}\ i_1 : t_1 \Rightarrow e)[(\texttt{rec}\ i_0(i_1) : t_1 \rightarrow^b t_2 \Rightarrow e)/i_0]]$$

$$E\,[(\texttt{fn}\ i : t \Rightarrow e)\ w] \rightarrow E[e[w/i]]$$

$$E\,[\texttt{let}\ i = w\ \texttt{in}\ e] \rightarrow E[e[w/i]]$$

$$E\,[\texttt{if}\ w\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : t] \rightarrow \begin{cases} E\,[e_1] & \text{if } w = \texttt{true} \\ E\,[e_2] & \text{if } w = \texttt{false} \end{cases}$$

$$E\,[w_1\ w_2] \rightarrow E\,[w_3] \qquad \text{if } (w_1, w_2, w_3) \in \delta_{\rightarrow}$$

Figure 6: Sequential Evaluation

and so allows to distinguish between the situations $(3 + \texttt{true}) \not\rightarrow$ that should have been caught by the type system and $(\texttt{hdnil}) \not\rightarrow$ that cannot be expected to be caught by any decidable type system. — Alternatively, one could mask the dynamically stuck configurartions using non-termination, e.g. to impose $(\texttt{hd}:\ t,\ \texttt{nil}, \texttt{hd}:\ t\ \texttt{nil}) \in \delta_{\rightarrow}$; this is essentially the approach of [15, Chapter 6].

## Concurrent evaluation

The transition relation for concurrent evaluation is given in Figure 8. Configurations have the form

$$cenv, PP$$

where *cenv* is a *channel environment* and $PP$ is a *process pool*. More precisely, a process pool $PP$ is a *partial function* from process identifiers $pi \in$

| | | |
|---|---|---|
| $\mathtt{pair}\colon t$ | $w_1$ | $\langle \mathtt{pair}\colon t\, w_1 \rangle$ |
| $\langle \mathtt{pair}\colon t\, w_1 \rangle$ | $w_2$ | $\langle \mathtt{pair}\ \colon t\, w_1\, w_2 \rangle$ |
| $\mathtt{fst}\colon t$ | $\langle \mathtt{pair}\colon t'\, w_1\, w_2 \rangle$ | $w_1$ |
| $\mathtt{snd}\colon t$ | $\langle \mathtt{pair}\colon t'\, w_1\, w_2 \rangle$ | $w_2$ |
| $\mathtt{cons}\colon t$ | $w_1$ | $\langle \mathtt{cons}\colon t\, w_1 \rangle$ |
| $\langle \mathtt{cons}\colon t\, w_1 \rangle$ | $w_2$ | $\langle \mathtt{cons}\colon t\, w_1\, w_2 \rangle$ |
| $\mathtt{hd}\colon t$ | $\langle \mathtt{cons}\colon t'\, w_1\, w_2 \rangle$ | $w_1$ |
| $\mathtt{tl}\colon t$ | $\langle \mathtt{cons}\colon t'\, w_1\, w_2 \rangle$ | $w_2$ |
| $\mathtt{isnil}\colon t$ | $\mathtt{nil}$ | $\mathtt{true}$ |
| $\mathtt{isnil}\colon t$ | $\langle \mathtt{cons}\colon t'\, w_1\, w_2 \rangle$ | $\mathtt{false}$ |
| $\mathtt{send}\colon t$ | $w$ | $\langle \mathtt{send}\colon t\, w \rangle$ |
| $\mathtt{receive}\colon t$ | $w$ | $\langle \mathtt{receive}\colon t\, w \rangle$ |
| $\mathtt{choose}\colon t$ | $w$ | $\langle \mathtt{choose}\colon t\, w \rangle$ |
| $\mathtt{wrap}\colon t$ | $w$ | $\langle \mathtt{wrap}\colon t\, w \rangle$ |

Figure 7: Tabulation of $(e_1, e_2, e_3) \in \delta_{\rightarrow}$

**PIdent** (e.g. $\mathtt{p}-\mathtt{0}, \mathtt{p}-\mathtt{1}, \ldots$) to the expression residing there. When writing a process pool $PP'$ in the form $PP[pi_1 \mapsto e_1] \ldots [pi_n \mapsto e_n]$ we take it for granted that all of $\mathrm{dom}(PP)$, $\{pi_1\}, \ldots, \{pi_n\}$ are mutually disjoint. The channel environment *cenv* is much like the type environment and so associates channel identifiers $ci \in$ **CIdent** (e.g. $\mathtt{c}-\mathtt{0}, \mathtt{c}-\mathtt{1}, \ldots$) with the type of values that may be communicated over the channel. We assume that the sets **Ident, PIdent** and **CIdent** are mutually disjoint. Also we formally regard a channel environment as a list of pairs of identifiers and types; as for the type environments we may then extract a partial function by mapping an identifier to the type of its rightmost occurrence. The advantage of this view will only show up in later proofs. The fact that we use a channel environment rather than just a set of previously allocated channels, is an example of the

book-keeping details present in the semantics. (If we were to use a set we would have to regard channel identifiers as constants, i.e. having an explicit type attached to each occurrence, and we would then need to formulate that all occurrences have the same type attached; this would turn out to be a more clumsy variation on the approach taken.)

The first axiom embeds sequential evaluation within concurrent evaluation. There is no explicit mentioning of the evaluation context since this is all taken care of in Figure 6. For book-keeping purposes the transition relation is labelled with the process executing and an *indication* of the communication behaviour; this will be useful in formulating and proving the results of the next section. Next we have axioms for those primitives of Figure 1 that were not dealt with in the definition of sequential evaluation. For channel allocation we use the channel environment to make sure that we do not re-allocate an already allocated channel. To record the allocation the channel environment is extended; for book-keeping purposes it turns out to be helpful for the next section that also the type is recorded and we do this by means of the channel environment. The behaviour labelling the arrow will be a monotype instance of $\forall\tau.\forall\imath.\tau$ CHAN $\imath$. The third axiom deals with process creation and is rather similar in spirit to the axiom for channel creation. The behaviour labelling the arrow will be a monotype instance of $\forall\tau.\forall\beta.\tau$ FORK $\beta$. The fourth axiom takes care of communication among different processes. (That they are indeed different follows from the syntactic conventions mentioned above.) The formulation makes use of a transition system for expressing when two "delayed" communications match and for calculating the respective outcomes as well as indications of the communication behaviour. One of the behaviours labelling the arrow will be a monotype instance of $\forall\rho.\forall\tau.\rho!\tau$ and the other will be a monotype instance of $\forall\rho.\forall\tau.\rho?\tau$.

## Matching

The transition system for matching is given in Figure 9. The first axiom collects the values communicated between primitive `send` and `receive` constructs. The next two rules take care of the situation where the communication taking place in the first position amounts to choosing between several possibilities. The subsequent rule allows modifying the local version of the value communicated; it does not affect the value communicated as can be

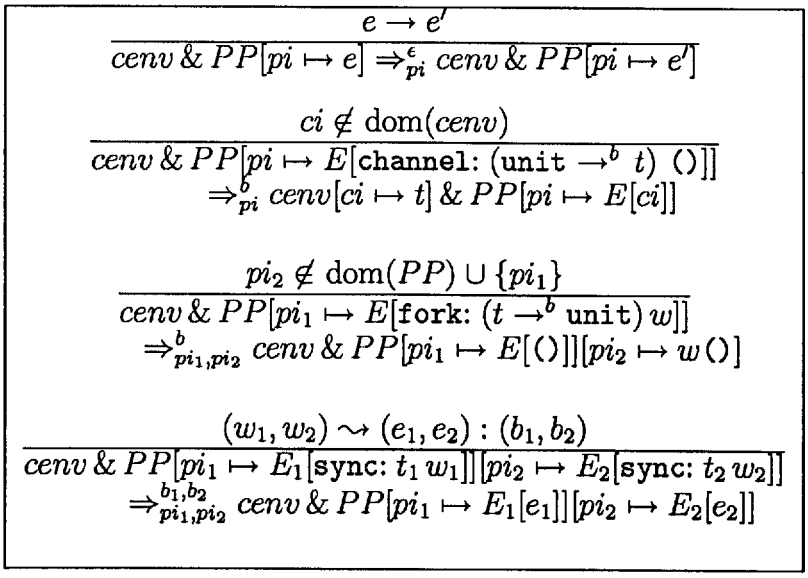


Figure 8: Concurrent Evaluation

seen from the fact that only the value in one of the components is being modified. On top of this we would need the symmetric system of one axiom and three rules but to conserve space we follow [19] in "cheating" by adding the final "restructuring rule"; this has the moderately undesirable effect of adding "non-structural" rules.

Finally, we note that it would be possible to add additional rules to the transition system for concurrent evaluation. Assuming that there is some distinguished start process `p-0` then the axiom

$$cenv \;\&\; PP[pi \mapsto w] \Rightarrow cenv \;\&\; PP \qquad\qquad \text{if } pi \mapsto \texttt{p-0}.$$

would describe the garbage collection of processes that have finished execution. In a similar way one could add an axiom for reclaiming channels no longer in use.

# 4 Deriving a Process Algebra from CML

We now show to which extent the types and behaviours are preserved or modified in the course of computation.

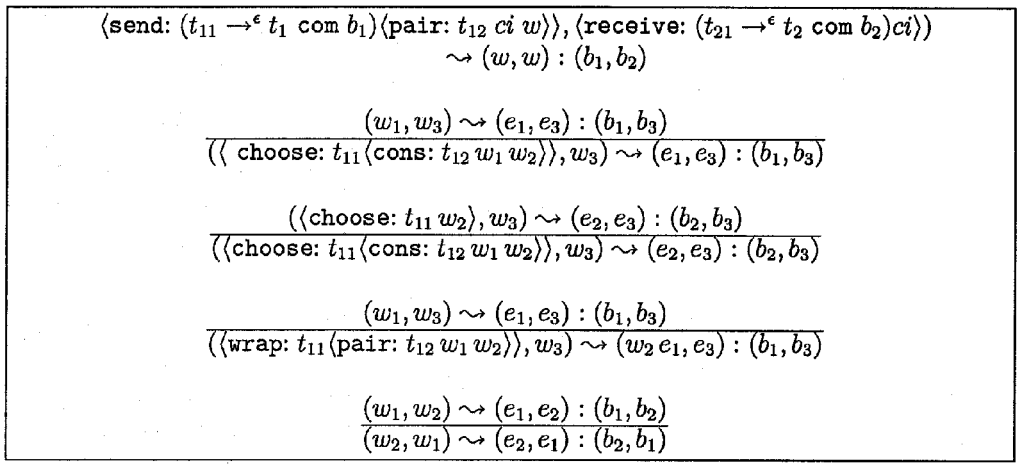


Figure 9: Matching

## Sequential Correctness

It is natural to restrict the attention to closed expressions, i.e. expressions with no free identifiers, because the definition of evaluation context is such that we never pass inside the scope of any defining occurrence for identifiers. However, we will have to allow that the expressions include channel identifiers that have been allocated in previous concurrent transitions. So we shall regard an expression $e$ as being closed when $cenv \vdash e \mid t\&b$ for some $cenv, t$ and $b$.

**Proposition 4.1**

If $cenv \vdash e \mid t \ \& \ b$ and $e \to e'$ then there exists $t^- \leq t$ and $b^- \leq b$ such that $cenv \vdash e' \mid t^- \ \& \ b^-$. □

Before approaching the proof it may be instructive to demonstrate why it would be too demanding to require that $t^- = t$ or $b^- = b$. For types suppose

that $t^- < t$ is given and that $c : t^-$ is a constant; then $(\texttt{fn}\ x : t \Rightarrow x)(c : t^-)$ has type $t$ but it evaluates to $c : t^-$ that has type $t^-$. For behaviours simply note that $\texttt{if true then}\ e_1\ \texttt{else}\ e_2$ has behaviour $\epsilon; (b_1 + b_2)$ and that it evaluates to $e_1$ that has behaviour $b_1$.

To conduct the proof we need several auxiliary results. The first lemma relates substitution to the use of the type environment. For the formulation recall that we regard type environments as lists of pairs (of identifiers and types) from which a partial function (from identifiers to types) can readily be recovered.

**Lemma 4.2**

If $i \notin \mathrm{dom}(tenv_2)$, $cenv \vdash e_0 \mid t_0 \,\&\, \epsilon$ and $cenv, tenv_1, [i \mapsto t_0], tenv_2 \vdash e \mid t \,\&\, b$ then $cenv, tenv_1, tenv_2 \vdash e[e_0/i] \mid t \,\&\, b$, i.e. the overall type and behaviour is unchanged under the substitution. □

The proof is by induction on the typing inference for $e$ and may be found in Appendix B. A simple consequence of this lemma is that if an identifier is not free in the expression then it may be removed from the type environment (as then substitution with respect to that identifier has no effect).

The second lemma may be read as saying that type and behaviour inference acts monotonically in the type environment as well as in the type and behaviour of subexpressions. To obtain a concise formulation we write $tenv_1 \leq tenv_2$ whenever $tenv_1$ and $tenv_2$ have equal length and pairs $(i_1, t_1)$ and $(i_2, t_2)$ in corresponding positions satisfy $i_1 = i_2$ and $t_1 \leq t_2$.

**Lemma 4.3**

If $cenv \vdash e_0 \mid t_0 \,\&\, b_0$, $cenv \vdash e_0' \mid t_0^- \,\&\, b_0^-$ and $cenv, tenv \vdash e[e_0/i] \mid t \,\&\, b$ and also $t_0^- \leq t_0, b_0^- \leq b_0$ and $tenv^- \leq tenv$; then there exists $t^- \leq t$ and $b^- \leq b$ such that $cenv, tenv^- \vdash e[e_0'/i] \mid t^- \,\&\, b^-$. □

The proof is by induction on the syntax of the expression $e$ and may be found in Appendix B. In a sense the lemma is two results in one, but to be economical in the proof effort it is advantageous to prove them jointly. So we shall sometimes feel free to use the lemma without any substitution. (To make this formally correct simply use a trivial substitution where $i$ is a "fresh" identifier that is not free in $e$.) The lemma has also the following important consequence:

**Corollary 4.4**

If $cenv \vdash e_0 \mid t_0 \;\&\; b_0, cenv \vdash e_0' \mid t_0^- \;\&\; b_0^-$ and $cenv \vdash E[e_0] \mid t \;\&\; b$ and also $t_0^- \leq t_0$ and $b_0^- \leq b_0$; then there exists $t^- \leq t$ and $b^- \leq b$ such that $cenv \vdash E[e_0'] \mid t^- \;\&\; b^-$. □

**Proof** Simply use the fact that $E[e]$ equals $(E[i])[e/i]$ when $i$ does not occur in $E$. □

One can now prove Proposition 4.1 by induction, i.e. case analysis, on the inference $e \rightarrow e'$; we refer to Appendix B for the details.

## Matching Correctness

The transition relation for concurrent evaluation utilizes the transition relations for sequential evaluation and for matching. It is therefore convenient to formulate the correctness of matching before considering the correctness of concurrent evaluation.

**Proposition 4.5**

If $cenv \vdash w_1 \mid (t_{01}\ \texttt{com}\ b_{01}) \;\&\; \epsilon,\ cenv \vdash w_2 \mid (t_{02}\ \texttt{com}\ b_{02}) \;\&\; \epsilon$ and

$$(w_1, w_2) \rightsquigarrow (e_1, e_2) : (b_1, b_2)$$

then there exists $t_{01}^-, t_{02}^-, b_1'$ and $b_2'$ such that

$$cenv \vdash e_1 \mid t_{01}^- \;\&\; b_1' \text{ with } b_1; b_1' \leq b_{01}$$
$$cenv \vdash e_2 \mid t_{02}^- \;\&\; b_2' \text{ with } b_2; b_2' \leq b_{02}$$

and where $t_{01}^- \leq t_{01}$ and $t_{02}^- \leq t_{02}$.

Furthermore, one of $b_1$ and $b_2$ may be written $r_1!t_1$ and the other $r_2?t_2$ where $t_1 \equiv t_2$ and $r_1$ and $r_2$ have a lower bound, i.e. $\exists r_0 : r_0 \leq r_1 \wedge r_0 \leq r_2$. □

The proof is by induction on the transition relation for matching and may be found in Appendix B.

## Concurrent Correctness

So far we have not extended the notion of well-typing to the configurations of the concurrent transition relation and our first task is to remedy this. To this end we shall need a partial function $PT$ of *process types*: it maps process identifiers $pi \in \mathbf{PIdent}$ to types. Similarly, we shall need a partial function $PB$ of *process behaviours*: it maps process identifiers to behaviours. Intuitively, a process pool $PP$ is correct with respect to $PT$ and $PB$ if each process, $PP(pi)$, has type and behaviour given by $PT(pi)$ and $PB(pi)$, respectively. It may be instructive to think of one of the process identifiers, e.g. `p-0`, as indicating the initial process whose final result is the overall result presented to the user, but this will have little impact on the development.

Formally, the correctness of $PP$ with respect to $PT$ and $PB$ is written

$$\vdash cenv, PP \mid PT \;\&\; PB$$

and is given by

$$\begin{array}{l}\mathrm{dom}(PP) = \mathrm{dom}(PT) = \mathrm{dom}(PB) \wedge \\ \forall pi \in \mathrm{dom}(PP) : cenv \vdash PP(pi) \mid PT(pi) \;\&\; PB(pi)\end{array}$$

If we were to admit axioms for reclaiming unused processes we should instead require that $\mathrm{dom}(PP) \subseteq \mathrm{dom}(PT) = \mathrm{dom}(PB)$. Also it would be possible to add the condition that all of $\mathrm{dom}(PP)$, $\mathrm{dom}(PT)$ and $\mathrm{dom}(PB)$ are finite and non-empty sets, but again this would have little impact on the development.

Our main result about concurrent evaluation is the following proposition that gives information about the evolution of types and behaviours. A concise formulation requires some additional notation. We allow writing $\vec{b}$ for $b$ as well as $b_1, b_2$ and similarly $\vec{pi}$ for $pi$ as well as $pi_1, pi_2$. When writing $\{\vec{pi}\}$ this then stands for $\{pi\}$ or $\{pi_1, pi_2\}$, respectively. When $\mathcal{P}$ is a partial function from process identifiers we write $\mathcal{P}\backslash\{\vec{pi}\}$ for the restriction $\mathcal{P}\lceil(\mathrm{dom}(\mathcal{P})\backslash\{\vec{pi}\})$ of $\mathcal{P}$ to the subset $\mathrm{dom}(\mathcal{P})\backslash\{\vec{pi}\}$ of $\mathrm{dom}(\mathcal{P})$. This notation applies to process pools, process types and process behaviours. For process types $PT$ and $PT'$ we write

$$PT'[\vec{pi}] \leq PT[\vec{pi}]$$

for

$$\{\vec{pi}\} \subseteq \mathrm{dom}(PT') \wedge \forall pi \in \{\vec{pi}\} \cap \mathrm{dom}(PT) : PT'(pi) \leq PT(pi)$$

This takes care of the situation where new processes are created.

**Proposition 4.6**

If $\vdash cenv,\ PP \mid PT\ \&\ PB$ and $cenv,\ PP \Rightarrow^{\vec{b}}_{\vec{pi}} cenv',\ PP'$ then there exists $PT'$ and $PB'$ such that

- if $\vec{b} = t_0$ CHAN $i_0$ then $cenv' = cenv[ci \mapsto t_0$ `chan` $i_0]$ for some $ci \notin \mathrm{dom}(cenv)$; otherwise $cenv' = cenv$,
- if $\vec{b} = t_0$ FORK $b_0$ and $\vec{pi} = pi_1, pi_2$ then $PT'(pi_2) \leq t_0$,
- $PP' \setminus \{\vec{pi}\} = PP \setminus \{\vec{pi}\}$,
- $PT' \setminus \{\vec{pi}\} = PT \setminus \{\vec{pi}\}$ and $PT'[\vec{pi}] \leq PT[\vec{pi}]$,
- $PB' \setminus \{\vec{pi}\} = PB \setminus \{\vec{pi}\}$

as well as $\vdash cenv', PP' \mid PT' \& PB'$. □

The proof is by cases on the rule used for the concurrent transition and may be found in Appendix B. It makes use of the following generalization of Corollary 4.4:

**Lemma 4.7**

If $cenv \vdash e_0 \mid t_0\ \&\ b_0,\ cenv \vdash e_0 \mid t_0'\ \&\ b_0',\ cenv \vdash E[e_0] \mid t\ \&\ b$ and also $t_0' \leq t_0$ and $b^\bullet; b_0' \leq b_0$; then there exists $t'$ and $b'$ such that $cenv \vdash E[e_0'] \mid t'\ \&\ b'$ and also $t' \leq t$ and $b^\bullet; b_0 \leq b$. □

The proof is by induction on the structure of $E$ and may be found in Appendix B.

## Process Algebras

The statement of Proposition 4.6 (as opposed to its proof) does not convey much information about the relationship between $PB[\vec{pi}], \vec{b}$ and $PB'[\vec{pi}]$. This

will be rectified now and our main tools will be two transition relations: one for the evolution of individual behaviours and one for the evolution of process behaviours.

The transition relation for individual behaviours takes the form

$$b_1 \longmapsto^a b_2$$

and says that the behaviour $b_1 \in \mathbf{Beh}$ evolves to $b_2 \in \mathbf{Beh}$ while performing the action $a$. It is possible to identify actions and behaviours, i.e. to use $a \in \mathbf{Beh}$, but it may be more informative to be more restrictive. To this end we define actions $a \in \mathbf{Act}$ by:

$$a \ ::= \ \epsilon \mid r!t \mid r?t \mid t \ \textsc{chan} \ r \mid t \ \textsc{fork} \ b$$

The details of the transition system are given in Figure 10. The first axiom simply records the effect of performing an individual action. Then we have a rule that allows evolution of actions to take place in more elaborate contexts. The next rule is patterned after a structural rule

$$\frac{b_1 \equiv b_1 \ b_1' \longmapsto^a b_2' \ b_2' \equiv b_2}{b_1 \longmapsto^a b_2}$$

as might be found in the $\pi$-calculus [13]. However, because of our use of sub-typing we find that we need a stronger rule and to obtain this we replace $\equiv$ by $\longmapsto^\epsilon$ and add three more axioms. The first says that $\equiv$ is contained in $\longmapsto^\epsilon$ and the final two allow to discard possible behaviours. (Actually $b_1 + b_2 \longmapsto^\epsilon b_2$ is derivable from the remaining axioms and rules.)

It is important that we have:

**Lemma 4.8**

The statement $b_1 \longmapsto^a b_2$ is equivalent to the statement $a; b_2 \leq b_1$. □

**Proof:** The implication from left to right may be established by induction on the inference tree for $b_1 \longmapsto^a b_2$: that $a; b_2 \leq b_1$ holds is clear for the axioms and is maintained by the rules. For the converse implication assume that $a; b_2 \leq b_1$. It follows that $a; b_2 + b_1 \equiv b_1$ and hence

$$b_1 \longmapsto^{\epsilon} a; b_2$$

From $a, \longmapsto^{a} \epsilon$ we next get

$$a; b_2 \longmapsto^{a} \epsilon; b_2$$

and since $\epsilon; b_2 \equiv b_2$ we then have

$$\epsilon; b_2 \longmapsto^{\epsilon} b_2$$

Putting this together we have

$$b_1 \longmapsto^{a} b_2$$

as desired. □

$$a \longmapsto^{a} \epsilon$$

$$\frac{b_1 \longmapsto^{a} b_2}{b_1; b \longmapsto^{a} b_2; b}$$

$$\frac{b_1 \longmapsto^{\epsilon} b_1' \quad b_1' \longmapsto^{a} b_2' \quad b_2' \longmapsto^{\epsilon} b_2}{b_1 \longmapsto^{a} b_2}$$

$$b \longmapsto^{\epsilon} b' \quad \text{if } b' \equiv b$$

$$b_1 + b_2 \longmapsto^{\epsilon} b_1$$

$$b_1 + b_2 \longmapsto^{\epsilon} b_2$$

Figure 10: Evolution of behaviours

The transition relation for process behaviours takes the form

$$PB \Longrightarrow^{\vec{b}}_{\vec{pi}} PB'$$

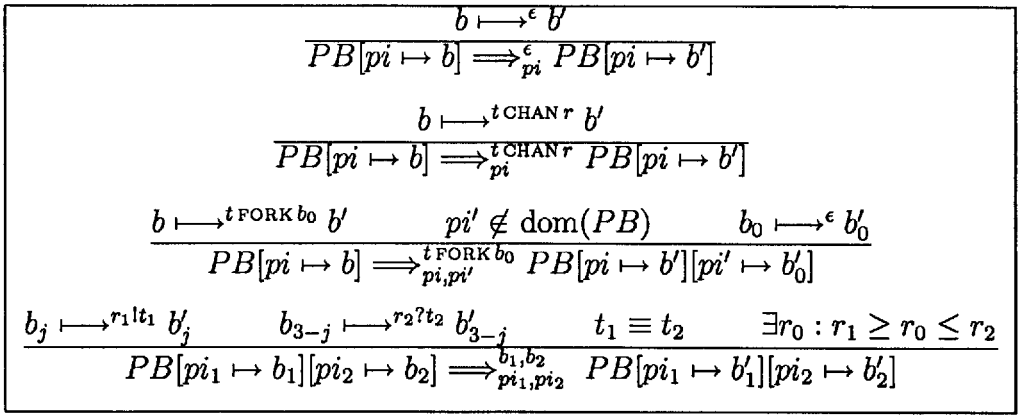

Figure 11: Evolution of process behaviours

and says that the process behaviour $PB$ evolves to the process behaviour $PB'$. Regarding process behaviours as a process algebra this transition relation then gives the operational semantics of terms in the process algebra. The details of the transition system are given in Figure 11 and make use of the transition relation for individual behaviours.

Our main result linking CML with Behaviours to a process algebra is the following:

**Proposition 4.9**

The statement of Proposition 4.6 may be extended with the following condition:

- $PB \Longrightarrow_{\vec{pi}}^{\vec{b}} PB'$

(using the notation of Proposition 4.6). □

The proof simply amounts to inspecting the proof of Proposition 4.6 and checking that the process behaviour $PB'$ constructed there satisfies the new claim.

# 5 Conclusion

We started our work with an existing programming language. The first step was to "extend the type system" with additional information about some of the phenomena that take place during execution; our approach was to define the syntax of behaviours based on the notion of effect systems [11]. The second step was to (re-)define an operational semantics in such a way that the newly added information does not influence the semantics, yet enough information is retained that it meaningfully describes the result of one step of evaluation. The third step was to prove this formally in the form of "subject reduction" and related results. In the course of this development the behaviours took on a life of their own: they were equipped with an operational semantics designed to make "subject reduction" both informative and provable, and the operational semantics was very close to the semantics of process algebras.

We believe that the main impact of this approach is not confined to the study of CML or similar languages. Rather one may ask in general for a programming language (with communication): how does the associated process algebra look. And conversely for an existing process algebra one may ask: for what kind of languages is this an appropriate process algebra. Questions like this may provide further insights into the role of operators in process algebras, e.g. the possibility of implementing an operator like '+' of CCS, because the perspective is beyond that of merely translating between the syntax of a programming language and the syntax of a process algebra. Also studies of the process algebra may provide valuable information when reasoning about programs in the language. In particular "negative" information can be carried over: we may for example conclude that a program definitely deadlocks whenever its behaviour has this property.

In our future work we hope to perform a deeper study of the relationship between the semantics of the programming language and the semantics (and syntax) of the process algebra. Our work will be guided by the following slogans:

- a *process algebra* is an *abstract interpretation* of (the effect system of) a *programming language* with communication;
- the *"propositions as types"* correspondence generalizes to a *"processes*

*as behaviours"* correspondence.

We believe that we have already demonstrated that a process algebra is an abstraction of a programming language; whether this is describable as an abstract interpretation remains to be seen.

## Acknowledgements

This work is part of the DART-project supported by The Danish Research Councils. Discussions with Uffe Engberg and Mads Tofte have been most helpful.

# References

[1] D. Berry, R. Milner, D.N. Turner: A semantics for ML concurrency primitives. Proceedings of POPL'92, ACM Press, 1992, pages 119-129.

[2] L. Cardelli, G. Longo: A Semantic Basis for Quest. *Journal of Functional Programming* **1** *4*, 1991, pages 417-458.

[3] G. Castagna, G. Ghelli, G. Longo: A Calculus for Overloaded Functions with Subtyping. Proceedings of the 1992 ACM Conference on Lisp and Functional Programming, ACM Press, 1992, pages 182-192.

[4] M. Coppo, M. Dezani: A new type assignment for $\lambda$-terms. *Archive. Math. Logik* **19**, 1978, pages 139-156.

[5] C. Crasemann: $\pi\lambda$-Kalküle für Prozesse und Funktionen. Ph.D.-Thesis, Christian-Albrechts-Universität zu Kiel, 1992.

[6] B.A. Davey, H.A. Priestly: *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[7] A. Giacalone, P. Mishra, S. Prasad: Operational and Algebraic Semantics for Facile: A Symmetric Integration of Concurrent and Functional Programming. Proceedings of ICALP '90, SLNCS 443, pages 765-780, 1990.

[8] K. Havelund, K.G. Larsen: The Fork Calculus. To appear in Proceedings of ICALP '93, 1993.

[9] H. Hüttel, K.G. Larsen: A Dynamic Type System for Higher-Order Processes. Manuscript, 1992.

[10] J.J.-Levy, B. Thomsen, L. Leth, A. Giacalone: Esprit Basic Research Action 6454— CONFER: CONcurrency and Functions: Evaluation and Reduction. EATCS Bulletin No. 48, 1992, pages 88-106.

[11] J.M. Lucassen, D.K. Gifford: Polymorphic Effect Systems. Proceedings of POPL'88, ACM Press, 1988, pages 47-57.

[12] R. Milner, M. Tofte, R. Harper: *The Definition of Standard ML.* MIT Press, 1990.

[13] R. Milner: The Polyadic $\pi$-Calculus: A Tutorial. Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, University of Edinburgh, 1991.

[14] F. Nielson: The Typed Lambda-Calculus with First-Class Processes. Proceedings of PARLE'89, Springer Lecture Notes in Computer Science **366**, 1989, pages 357-373.

[15] F. Nielson, H.R. Nielson: *Two-Level Functional Languages.* Cambridge University Press, 1992.

[16] E.G.J.M.H. Nocker, J.E.W. Smetsers, M.C.J.D. van Eekelen, M.J. Plasmeijer: Concurrent Clean. Proceedings of PARLE'91, SLNCS 506, pages 202-219, 1991.

[17] C. Reade: *Elements of Functional Programming.* Addison-Wesley, 1989.

[18] J.H. Reppy: CML: A Higher-order Concurrent Language. Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation, ACM Press, 1991, pages 293-305.

[19] J.H. Reppy: Higher-Order Concurrency. Ph.D.-Thesis, Report 92-1285, Department of Computer Science, Cornell University, 1992.

[20] J.-P. Talpin, P. Jouvelot: The Type and Effect Discipline. Proceedings of LICS'92, 1992, pages 162-173.

[21] B. Thomsen: A Calculus of Higher Order Communicating Systems. Proceedings of POPL'89, ACM Press, 1989, pages 143-154.

[22] B. Thomsen: Plain CHOCS. Report DOC 89/4, Imperial College, University of London, 1989.

[23] B. Thomsen: Calculi for Higher Order Communicating Systems. Ph.D.-Thesis, Imperial College, University of London, 1990.

# A Variations on the subtyping

The main idea behind the ordering $t_1 \leq t_2$ on types is that $t_2$ is more permissive than $t_1$ in the communications being allowed but that the "underlying" types $t_1$ and $t_2$ must be equal. This is illustrated in the following example.

**Example A.1**

Consider the following fragment of a program:

```
(fn f :  unit →^{b_f} unit ⇒ ···)
      (rec g x :  unit →^{b_f} unit ⇒
      let y = sync (receive ach)
      in let z = sync (send (pair bch y))
         in if ··· then g () else ())
```

We shall assume that `ach` has type `int chan a` and `bch` has type `int chan b` and then the remaining type information should be clear from the context. Here `g` is a concrete program that a number of times will receive an integer over ach and then retransmit it over bch. Its type is `g:  unit` $\rightarrow^{b_g}$ `unit` where

$$b_g = \text{REC}\beta.\ \texttt{a?int;}\ \texttt{b!int;}\ (\beta + \epsilon)$$

Similarly `(fn f :  unit` $\rightarrow^{b_f}$ `unit` $\Rightarrow \cdots$`)` is some module that requires that the argument obeys a certain protocol. This protocol says that the only communications allowed are the input of integers over some channel in region `a` and the output of integers over some channel in region `b`. This may be described more formally by

$$b_f = \text{REC}\beta.\ (\texttt{a?int;}\beta) + (\texttt{b!int;}\beta) + \epsilon$$

We would then like to show that

$$\texttt{unit} \rightarrow^{b_g} \texttt{unit} \leq \texttt{unit} \rightarrow^{b_f} \texttt{unit}$$

corresponding to the fact that g obeys the protocol of `f`. Using the rules of Figure 4 this amounts to showing $b_g \leq b_f$. □

The axioms and rules of Figure 5 do not suffice for proving this. This suggests adding a "contraction rule"

$$\frac{b_1 \leq b[bv \mapsto b_1] \qquad b[bv \mapsto b_2] \leq b_2}{b_1 \leq b}$$

if $b$ guards $bv$ and $bv$ is positive in $b$.

This generalizes a rule from [2] and is explained in the sequel. A behaviour variable is *positive* in a behaviour if all occurrences have positive polarity in the sense of Section 3. This is a standard concept and we shall not go deeper into it here although some care is called for due to presence of recursion. Intuitively, a behaviour $b$ *guards* a behaviour variable $bv$ if each "path" from the "beginning" of $b$ to $bv$ must pass through a non-empty behaviour. The precise details are more subtle than in CCS because $bv$ does not occur guarded in $b = b'; bv$ if we have $b'; b' \equiv b'$ (as when $b' = \epsilon$). We shall not go further into this here but only show that with the above rule we can solve the problem of Example A.1. We should add that the theoretical development of this paper is fairly robust under the addition of new axioms and rules.

**Example A.2**

We now show $b_g \leq b_f$ where $b_g$ and $b_f$ are as in Example A.1. For this define

$$b = \texttt{a?int}; \texttt{b!int}; (\beta + \epsilon).$$

Since $b_g = \text{REC}\beta.b$ it is evident that

$$b_g \leq [\beta \mapsto b_g]$$

using the axiom for the one-level onfolding of REC. Although we have not defined "positive" and "guards" formally it should be immediate that $\beta$ is "positive" in $b$ and that $b$ "guards" $\beta$. Using the contraction rule above it then suffices to show

$$b[\beta \mapsto b_f] \leq b_f$$

For this we abbreviate `a?int` to $A$ and `b!int` to $B$. We then have

$$
\begin{aligned}
b[\beta \mapsto b_f] &\equiv A; B; (b_f + \epsilon) \\
&\leq A; B; b_f + A; B; (\epsilon + A; b_f + B; b_f) \\
&\equiv A; B; b_f + A; B; b_f \\
&\equiv A; B; b_f \\
&\leq A; (A; b_f + B; b_f + \epsilon) \\
&\equiv A; b_f \\
&\leq A; b_f + B; b_f + \epsilon \\
&\equiv b_f
\end{aligned}
$$

and the result follows.

By contrast, if $b'_g = \texttt{a?int;b!bool}$ and we were to show $b'_g \leq b_f$ then no contraction rule would be needed: just use the axiom for the one-level unfolding of REC twice and then some simple axioms. □

## Coarsening the structure

Early on we said that we intended to deviate from [14] in keeping the dependencies between individual communications. However, suppose that now we want to coarsen the structure of behaviours such that these distinctions no longer can be made. One possibility is to add the axioms

$$
\begin{aligned}
&b_1 + b_2 \equiv b_1; b_2 \\
&\text{REC } bv.b \equiv b \\
&bv \equiv \epsilon
\end{aligned}
$$

The first expresses that we no longer distinguish between choice and sequencing. The next two axioms have the effect of removing the "REC $bv$." binder as well as behaviour variables; for closed behaviours this would be equivalent to the axiom REC $bv$. $b \equiv b[bv \mapsto \epsilon]$.

An alternative presentation of the same idea is to translate the behaviours $b \in$ **Beh** to a simpler structure of behaviours $b' \in$ **Beh'** given by:

$$
b' \ ::= \ \epsilon \mid r!t \mid r?t \mid t \text{ CHAN} \mid t \text{ FORK } b' \mid b'_1 \cup b'_2
$$

Here $(b_1 + b_2)' = (b_1; b_2)' = b'_1 \cup b'_2$, (REC $bv$. $b)' = b'$ and $bv' = \epsilon$. Comparing with the approach of [14] we have now lost the dependency between commu-

nications and $b'_1 \cup b'_2$ expresses that each of $b'_1$ and $b'_2$ may be performed zero, one or many times and in arbitrary order. This is the same interpretation of the union operator $\cup$ as in [14].

However, we still deviate from [14] in keeping behaviours and effects separate. To mimic the development in [14] more closely we may translate types $t \in$ **Type** and behaviours $b \in$ **Beh** into so-called behaviour types $[\![t]\!], [\![b]\!],\ bt \in$ **BehTyp** given by

$$\begin{array}{rcl} bt & ::= & \texttt{unit} \mid \texttt{bool} \mid \texttt{int} \mid bt_1 \times bt_2 \mid bt\ \texttt{list} \\ & \mid & bt_1 \to bt_2 \mid bt\ \texttt{chan}\ r \mid \epsilon \mid r!bt \mid r?bt \\ & \mid & \texttt{fork}\ bt \mid bt_1 \cup bt_2 \end{array}$$

Most translations are fairly simple structural definitions. Some of the more interesting ones are:

$$\begin{array}{rcl} [\![t_1 \times t_2]\!] & \equiv & [\![t_1]\!] \times [\![t_2]\!] \\ [\![t_1 \to^b t_2]\!] & \equiv & [\![b]\!] \cup ([\![t_1]\!] \to [\![t_2]\!]) \\ [\![t\ \texttt{chan}\ r]\!] & \equiv & [\![t]\!]\ \texttt{chan}\ r \\ [\![t\ \texttt{com}\ b]\!] & \equiv & [\![t]\!] \cup [\![b]\!] \\ [\![r!t]\!] & \equiv & r![\![t]\!] \\ [\![t\ \textsc{chan}\ r]\!] & \equiv & [\![t]\!]\ \texttt{chan}\ r \\ [\![t\ \textsc{fork}\ b]\!] & \equiv & \texttt{fork}\ ([\![t]\!] \cup [\![b]\!]) \\ [\![b_1; b_2]\!] & \equiv & [\![b_1]\!] \cup [\![b_2]\!] \end{array}$$

The resulting system is pretty close in spirit to [14] and the remaining differences are due to the differences between the underlying languages (CML versus TPL of [14]).

# B Proofs of main results

## Proof of Lemma 4.2

We proceed by induction on the typing inference for $e$. So we must consider each axiom and rule of Figure 2 as well as the rule added in Section 3.

*Constants.* Then $e$ is a constant and $e[e_0/i]$ equals $e$. The result is then immediate.

*Identifiers.* Then we have two cases. If $e$ is an identifier different from $i$ the result follows as for constants. If $e$ is identical to $i$ then $t = t_0$ and $b = \epsilon$. But $e[e_0/i]$ equals $e_0$ and by assumption $cenv \vdash e_0 \mid t \;\&\; b$. To obtain the desired result we modify the proof tree for $cenv \vdash e_0 \mid t \;\&\; b$ as follows: each node must be of the form[6]

$$cenv, tenv \vdash e_1 \mid t_1 \;\&\; b_1$$

and we replace it by

$$cenv, tenv_1, tenv_2, tenv \vdash e_1 \mid t_1 \;\&\; b_1$$

obtaining the desired proof tree for $cenv, tenv_1, tenv_2 \vdash e_0 \mid t \;\&\; b$.

*Abstraction.* Then $e$ must be of the form $\texttt{fn}\ i_1 : t_1 \Rightarrow e_1$ and $t = t_1 \rightarrow^{b_1} t_2$ and $b = \epsilon$. We have two cases. If $i$ is different from $i_1$ then the induction hypothesis is applicable to

$$cenv, tenv_1[i \mapsto t_0], tenv_2[i_1 \mapsto t_1] \vdash e_1 \mid t_2 \;\&\; b_1$$

and yields

$$cenv, tenv_1, tenv_2[i_1 \mapsto t_1] \vdash e_1[e_0/i] \mid t_2 \;\&\; b_1$$

from which the desired

$$cenv, tenv_1, tenv_2 \vdash e[e_0/i] \mid t \;\&\; b$$

follows because $e[e_0/i]$ equals $\texttt{fn}\ i_1 : t_1 \Rightarrow (e_1[e_0/i])$.

The second case is when $i$ is identical to $i_1$. Then $e[e_0/i]$ equals $i$ and from

$$cenv, tenv_1[i \mapsto t_0], tenv_2 \vdash e \mid t \;\&\; b \qquad (\star)$$

[6]Here we make use of the fact that type environments are lists rather than functions.

we must infer

$$cenv, tenv_1, tenv_2 \vdash e \mid t \;\&\; b.$$

We do this by modifying the proof tree for $(\star)$ as follows: each node must be of the form

$$cenv, tenv_1[i \mapsto t_0], tenv_2, tenv \vdash e_2 \mid t_3 \;\&\; b_2$$

and we replace it by

$$cenv, tenv_1, tenv_2, tenv \vdash e_2 \mid t_3 \;\&\; b_2$$

This is valid since $i \in \mathrm{dom}(tenv)$ whenever $i$ is free in $e_2$.

*Application.* Then $e$ must be of the form $e_1 e_2$. Inspecting the proof tree we must have premisses of the form

$$cenv, tenv_1[i \mapsto t_0], tenv_2 \vdash e_1 \mid t_1 \rightarrow^{b_3} t \;\&\; b_1$$
$$cenv, tenv_1[i \mapsto t_0], tenv_2 \vdash e_2 \mid t_1^- \;\&\; b_2$$

with $t_1^- \leq t_1$ and $b = b_1; b_2; b_3$. The induction hypothesis is applicable and yields

$$cenv, tenv_1, tenv_2 \vdash e_1[t_0/i] \mid t_1 \rightarrow^{b_3} t \;\&\; b_1$$
$$cenv, tenv_1, tenv_2 \vdash e_2[t_0/i] \mid t_1^- \;\&\; b_2$$

from which the desired result follows.

*Let-abstraction.* This case follows using combinations of the techniques from abstraction and application; this should not be surprising because in the absence of polymorphism the expression $(\texttt{fn}\ i_1 : t_1 \Rightarrow e_1)e_2$ is rather equivalent to $\texttt{let}\ i_1 = e_2\ \texttt{in}\ e_1$.

*Recursion.* Then $e$ must be of the form $\texttt{rec}\ i_0(i_1) : t \Rightarrow e_1$ and $t = t_1 \rightarrow^{b_1} t_2$ and $b = \epsilon$. If $i_0, i_1$ and $i$ are all different we proceed as follows. Inspection of the proof tree reveals a premiss of the form

$$cenv,\ tenv_1[i \mapsto t_0], tenv_2[i_0 \mapsto t][i_1 \mapsto t_1] \vdash e_1 \mid t_2^- \ \&\ b_1^-$$

where $t_2^- \leq t_2$ and $b_1^- \leq b_1$. Applying the induction hypothesis we obtain

$$cenv, tenv_1, tenv_2[i_0 \mapsto t][i_1 \mapsto t_1] \vdash e_1[e_0/i] \mid t_2^- \ \&\ b_1^-$$

and this yields

$$cenv, tenv_1, tenv_2 \vdash \texttt{rec}\ i_0(i_1) : t \Rightarrow e_1[e_0/i] \mid t \ \&\ b$$

which is indeed the desired result.

If $i$ equals one or more of $i_0$ and $i_1$ then $e[e_0/i]$ equals $e$. We then prove the desired result as in the similar case for abstraction.

*Conditional.* This case follows using the techniques from application; this should not be surprising because for the purposes of type inference the expression `if` $e'$ `then` $e_1$ `else` $e_2 : t$ behaves as the nested application (`cond:` $t'$) $e'$ $e_1$ $e_2$.

*Weakly evaluated constants.* Given our decision to take $w ::= i$ rather than $e ::= i$ in the syntax this is not a trivial case because identifiers may occur in weakly evaluated expressions of the form $\langle c\ w_1 \ldots w_m \rangle$. However, the proof may be conducted using the techniques from application. □

## Proof of Lemma 4.3

We proceed by induction on the syntax of the expression $e$; this includes the syntactic category of weakly evaluated expressions.

*Constants.* Then $e$ is of the form $c : t$ and $b = \epsilon$. It follows that $e[e_0/i]$ and $e[e_0'/i]$ both equal $c : t$ and the result is then straightforward: let $t^- = t$ and $b^- = b$.

*Identifiers.* We have two cases. If the expression $e$ is different from the identifier $i$ then $e[e_0/i]$ and $e[e_0'/i]$ both equal $e$ and we have $t = tenv(i)$ and $b = \epsilon$. By taking $t^- = tenv^-(i)$ and $b^- = b$ the result is then straightforward.

The other case is when $e$ is identical to $i$. Then $e[e_0/i]$ equals $e_0$. From the assumption

$$cenv \vdash e_0 \mid t_0 \;\&\; b_0$$

we may obtain

$$cenv, tenv \vdash e_0 \mid t_0 \;\&\; b_0$$

by modifying the proof tree in the manner demonstrated in the proof of Lemma 4.2. Since we also have

$$cenv, tenv \vdash e_0 \mid t \;\&\; b$$

it follows from Fact 2.2 that $t = t_0$ and $b = b_0$. Since $e[e'_0/i]$ equals $e'_0$ we obtain

$$cenv, tenv \vdash e[e'_0/i] \mid t_0^- \;\&\; b_0^-$$

by performing a simple modification of the proof tree. Taking $t^- = t_0^-$ and $b^- = b_0^-$ then gives the desired result.

*Abstraction.* Then $e$ must be of the form $\texttt{fn}\ i_1 : t_1 \Rightarrow e_1$ and $t = t_1 \rightarrow^{b_1} t_2$ and $b = \epsilon$. We have two cases. If $i$ is identical to $i_1$ then both $e[e_0/i]$ and $e[e'_0/i]$ equal $e$. Inspection of the proof tree for $cenv, tenv \vdash e \mid t \;\&\; b$ then reveals a premiss

$$cenv, tenv[i_1 \mapsto t_1] \vdash e_1 \mid t_2 \;\&\; b_1$$

The induction hypothesis is applicable because $e_1$ equals $e_1[e_0/j]$ for a fresh identifier $j$ and yields

$$cenv, tenv^-[i_1 \mapsto t_1] \vdash e_1 \mid t_2^- \;\&\; b_1^-$$

for $t_2^- \leq t_2$ and $b_1^- \leq b_1$. It follows that

$$cenv, tenv^- \vdash e \mid t^- \;\&\; b^-$$

with $t^- = t_1 \rightarrow^{b_1^-} t_2^-$ and $b^- = \epsilon$.

The other case is when $i$ is different from $i_1$. Inspection of the proof tree for $cenv, tenv \vdash e[e_0/i] \mid t \;\&\; b$ then reveals a premiss

$$cenv, tenv[i_1 \mapsto t_1] \vdash e_1[e_0/i] \mid t_2 \;\&\; b_1$$

The induction hypothesis is applicable and yields

$$cenv, tenv^-[i_1 \mapsto t_1] \vdash e_1[e_0'/i] \mid t_2^- \;\&\; b_1^-$$

for $t_2^- \leq t_2$ and $b_1^- \leq b_1$. It follows that

$$cenv, tenv^- \vdash e[e_0'/i] \mid t^- \;\&\; b^-$$

with $t^- = t_1 \rightarrow^{b_1^-} t_2^-$ and $b^- = \epsilon$.

*Application.* Then $e$ must be of the form $e_1\ e_2$. Inspecting the proof tree for $e[e_0/i]$ we find premisses of the form

$$cenv, tenv \vdash e_1[e_0/i] \mid t_1 \rightarrow^{b_3} t \;\&\; b_1$$
$$cenv, tenv \vdash e_2[e_0/i] \mid t_1^- \;\&\; b_2$$

with $t_1^- \leq t_1$ and $b = b_1; b_2; b_3$. The induction hypothesis is applicable and yields

$$cenv, tenv^- \vdash e_1[e_0'/i] \mid t_1^+ \rightarrow^{b_3^-} t^- \;\&\; b_1^-$$
$$cenv, tenv^- \vdash e_2[e_0'/i] \mid t_1^{--} \;\&\; b_2^-$$

for $t_1^+ \geq t_1, b_3^- \leq b_3, t^- \leq t, b_1^- \leq b_1, t_1^{--} \leq t_1^-$ and $b_2^- \leq b_2$. Since $t_1^{--} \leq t_1^+$ it follows that

$$cenv, tenv^- \mid e[e_0'/i] \mid t^- \;\&\; b^-$$

with $b^- = b_1^-; b_2^-; b_3^-$ . This shows the desired result.

*Let-abstraction.* Then $e$ must be of the form $\texttt{let}\ j = e_1\ \texttt{in}\ e_2$. As in the proof of Lemma 4.2 this case utilizes combinations of the techniques

from abstraction and application. The additional complication is that the substitution into $e_1$ may present a new modification of the type environment for $e_2$. We have two cases. If the identifiers $i$ and $j$ are distinct then inspection of the proof tree for $e[e_0/i]$ reveals premisses of the form

$$cenv, tenv \vdash e_1[e_0/i] \mid t_1 \;\&\; b_1$$
$$cenv, tenv[i_1 \mapsto t_1] \vdash e_2[e_0/i] \mid t \;\&\; b_2$$

with $b = b_1; b_2$. Applying the induction hypothesis to the first inference yields

$$cenv, tenv^- \vdash e_1[e_0'/i] \mid t_1^- \& b_1^-$$

where $t_1^- \leq t_1$ and $b_1^- \leq b_1$. Applying the induction hypothesis to the second inference yields

$$cenv, tenv^-[i_1 \mapsto t_1^-] \vdash e_2[e_0'/i] \mid t^- \;\&\; b_2^-$$

with $t^- \leq t$. Taking $b^- = b_1^-; b_2^-$ this yields

$$cenv, tenv^- \vdash e[e_0'/i] \mid t^- \;\&\; b^-$$

which is the desired result.

The second case is when $i$ and $j$ are identical. As was illustrated for abstraction it is straightforward to modify the above proof so as to apply in this case: we still have to substitute in $e_1$ but should not do so in $e_2$.

*Recursion.* Then $e$ must be of the form $\texttt{rec}\ i_0(i_1) : t \Rightarrow e_1$ and $t = t_1 \rightarrow^{b_1} t_2$ and $b = \epsilon$. We have two cases. If $i$ equals one or more of $i_0$ and $i_1$, then $e[e_0/i]$ and $e[e_0'/i]$ both equal $e$. Inspection of the proof tree for $e[e_0/i]$ then reveals a premiss

$$cenv, tenv[i_0 \mapsto t][i_1 \mapsto t_1] \vdash e_1 \mid t_2^- \;\&\; b_1^-$$

where $t_2^- \leq t_2$ and $b_1^- \leq b_1$. The induction hypothesis is applicable (because $e_1$ equals $e_1[e_0/j]$ for a fresh identifier $j$) and yields

$$cenv, tenv^-[i_0 \mapsto t][i_1 \mapsto t_1] \vdash e_1 \mid t_2^{--} \;\&\; b_1^{--}$$

for $t_2^{--} \leq t_2^-$ and $b_1^{--} \leq b_1^-$. It follows that

$$cenv, tenv^- \vdash e \mid t \;\&\; b$$

since $t_2^{--} \leq t_2$ and $b_1^{--} \leq b_1$.

The other case is when $i_0, i_1$ and $i$ are all different. Inspection of the proof tree for $e[e_0/i]$ then reveals a premiss

$$cenv, tenv[i_0 \mapsto t][i_1 \mapsto t_1] \vdash e_1[e_0/i] \mid t_2^{--} \;\&\; b_1^{--}$$

where $t_2^- \leq t_2$ and $b_1^- \leq b_1$. The induction hypothesis is applicable and yields

$$cenv, tenv^-[i_0 \mapsto t][i_1 \mapsto t_1] \vdash e_1[e_0'/i] \mid t_2^{--} \;\&\; b_1^{--}$$

for $t_2^{--} \leq t_2^-$ and $b_1^{--} \leq b_1^-$. It follows that

$$cenv, tenv^- \vdash e[e_0'/i] \mid t \;\&\; b$$

because $e[e_0/i]$ equals $\texttt{rec}\ i_0(i_1) : t \Rightarrow (e_1[e_0/i])$.

*Conditional.* As in the proof of Lemma 4.2 this case follows using the techniques from application.

*Weakly evaluated constants.* As in the proof of Lemma 4.2 this case follows using the techniques from application. □

## Proof of Proposition 4.1

We proceed by induction on the inference $e \to e'$, i.e. by case analysis according to Figure 6.

*Recursion.* The inference $e \to e'$ must have the form

$$E[e_1] \to E[e_2[e_1/i_0]]$$

where $e_1$ and $e_2$ are given by

$$e_1 = \texttt{rec}\ i_0(i_1) : t_1 \rightarrow^{b_1} t_2 \Rightarrow e_0$$
$$e_2 = \texttt{fn}\ i_1 : t_1 \Rightarrow e_0$$

Since the proof tree for $cenv \vdash E[e_1] \mid t\ \&\ b$ follows the syntax of $E[e_1]$ it is possible to identify the node corresponding to the hole in $E$. It must have the form

$$cenv, tenv \vdash e_1 \mid t_1'\ \&\ b_1'$$

with $tenv$ being empty because the definition of the evaluation context is such that the hole is never in the scope of any binding occurrence of an identifier. The premiss of this node must be

$$cenv[i_0 \mapsto t_1 \rightarrow^{b_1} t_2][i_1 \mapsto t_1] \vdash e_0 \mid t_2^-\ \&\ b_1^-$$

for $t_2^- \leq t_2$ and $b_1^- \leq b_1$. This then shows that $t_1' = t_1 \rightarrow^{b_1} t_2$ and $b_1' = \epsilon$.

In the case where $i_0$ is distinct from $i_1$ we may use Lemma 4.2 to obtain a proof

$$cenv[i_1 \mapsto t_1] \vdash e_0[e_1/i_0] \mid t_2^-\ \&\ b_1^-$$

from which

$$cenv \vdash e_2[e_1/i_0] \mid t_1 \rightarrow^{b_1^-} t_2^-\ \&\ \epsilon$$

is immediate. But by Corollary 4.4 and $t_1 \rightarrow^{b_1^-} t_2^- \leq t_1 \rightarrow^{b_1} t_2$ and $\epsilon \leq b_1'$ there exist $t^- \leq t$ and $b^- \leq b$ such that

$$cenv \vdash E[e_2[e_1/i_0]] \mid t^-\ \&\ b^-$$

and this is the result.

In the case where $i_0$ is identical to $i_1$ we have that $e_2[e_1/i_0]$ equals $e_2$. From the typing of $e_0$ we may then obtain

$$cenv[i_0 \mapsto t_1 \rightarrow^{b_1} t_2] \vdash e_2 \mid t_1 \rightarrow^{b_1} t_2^- \;\&\; \epsilon$$

and by straightforward modification of the proof tree we obtain

$$cenv \vdash e_2 \mid t_1 \rightarrow^{b_1^-} t_2^- \;\&\; \epsilon$$

As before Corollary 4.4 then gives the desired result.

*Application ($\beta$-reduction).* The inference $e \rightarrow e'$ must have the form

$$E[(\texttt{fn}\ i : t_1 \Rightarrow e_0)w] \rightarrow E[e_0[w/i]]$$

Inspection of the proof tree for $cenv \vdash e \mid t \;\&\; b$ once more identifies a node

$$cenv \vdash (\texttt{fn}\ i : t_1 \Rightarrow e_0)w \mid t_2 \;\&\; b_1$$

corresponding to the hole in $E$. It must have premisses

$$\begin{array}{ll} cenv \vdash (\texttt{fn}\ i : t_1 \Rightarrow e_0) \mid t_1 \rightarrow^{b_1} t_2 \;\&\; \epsilon & (\star) \\ cenv \vdash w \mid t_1^- \;\&\; b_2 & \end{array}$$

where $t_1^- \leq t_1$ and $b_1' = \epsilon; b_2; b_1$. To see that $b_2 = \epsilon$ we use:

**Fact B.1** If $cenv, tenv \vdash w \mid t \;\&\; b$ and $w$ is a weakly evaluated expression then $b = \epsilon$. □

**Proof** A simple induction over weakly evaluated expressions. □

Inspection of the proof tree ($\star$) reveals a premiss

$$cenv[i \mapsto t_1] \vdash e_0 \mid t_2 \;\&\; b_1$$

and by Lemma 4.3 (with a trivial substitution) this yields

$$cenv[i \mapsto t_1^-] \vdash e_0 \mid t_2^- \;\&\; b_1^-$$

for $t_2^- \leq t_2$ and $b_1^- \leq b_1$. Using Lemma 4.2 we then obtain

$$cenv \vdash e_0[w/i] \mid t_2^- \ \& \ b_1^-$$

Since $t_2^- \leq t_2$ and $b_1^- \leq b_1 \leq \epsilon; \epsilon; b_1 \leq b_1'$ the desired result then follows from Corollary 4.4.

*Let-abstraction.* This case is slightly simpler than the one for application but we shall nonetheless provide the details. The inference $e \rightarrow e'$ must have the form

$$E[\texttt{let}\ i = w\ \texttt{in}\ e_0] \rightarrow E[e_0[w/i]]$$

Inspection of the proof tree for $e$ once more identifies a node

$$cenv \vdash \texttt{let}\ i = w\ \texttt{in}\ e_0 \mid t_2 \ \& \ b_1'$$

corresponding to the hole in $E$. It must have premisses

$$cenv \vdash w \mid t_1 \ \& \ \epsilon$$
$$cenv[i \mapsto t_1] \vdash e_0 \mid t_2 \ \& \ b_2$$

where $b_1' = \epsilon; b_2$ and we have used Fact B.1. Using Lemma 4.2 we obtain

$$cenv \vdash e_0[w/i] \mid t_2 \ \& \ b_2$$

and since $t_2 \leq t_2$ and $b_2 \leq \epsilon; b_2 \leq b_1'$ the desired result follows from Proposition 4.4.

*Conditional.* The inference $e \rightarrow e'$ must have the form

$$E[\texttt{if}\ w\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : t_0] \rightarrow E[e_1]$$

where, without loss of generality, we assume that $w = \text{true}$. Inspection of the proof tree for $cenv \vdash e \mid t \ \& \ b$ once more identifies a node

$$cenv \vdash \texttt{if}\ w\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : t_0 \mid t_0 \ \& \ b_0$$

corresponding to the hole in $E$. It must have premisses

$$cenv \vdash w \mid \texttt{bool} \;\&\; \epsilon$$
$$cenv \vdash e_1 \mid t_1 \;\&\; b_1$$
$$cenv \vdash e_2 \mid t_2 \;\&\; b_2$$

where $t_1 \leq t_0, t_2 \leq t_0, b_0 = \epsilon; (b_1 + b_2)$ and we have used Fact B.1. Applying Corollary 4.4 to $cenv \vdash e_1 \mid t_1 \;\&\; b_1$ we then get the desired result because $t_1 \leq t_0$ and $b_1 \leq b_1 + b_2 \leq \epsilon; (b_1 + b_2) \leq b_0$.

*Application ($\delta$-reduction).* The inference $e \to e'$ must have the form

$$E[w_1\ w_2] \to E[w_3]$$

where $(w_1, w_2, w_3) \in \delta_{\to}$(see Figure 7). Inspection of the proof tree for $cenv \vdash e \mid t \;\&\; b$ once more identifies a node

$$cenv \vdash w_1 w_2 \mid t_2 \;\&\; b_0$$

corresponding to the hole in $E$. It must have premisses

$$cenv \vdash w_1 \mid t_1 \to^{b_1} t_2 \;\&\; \epsilon$$
$$cenv \vdash w_2 \mid t_1^- \;\&\; \epsilon$$

where $t_1^- \leq t_1$ and $b_0 = \epsilon; \epsilon; b_1$ and we have used Fact B.1. To obtain the desired result using Corollary 4.4 it suffices to find $t_2^-$ and show

$$cenv \vdash w_3 \mid t_2^- \;\&\; \epsilon$$
$$t_2^- \leq t_2$$
$$b_1 \equiv \epsilon$$

as then $\epsilon \leq \epsilon; \epsilon; b_1 \leq b_0$.

This may be achieved by inspection of Figure 7. We only consider two typical cases. The case where

$$w_1 = \texttt{pair} : t_3 \to^{\epsilon} t_4 \to^{\epsilon} t_3 \times t_4$$
$$w_2 = w$$
$$w_3 = \langle w_1 w_2 \rangle$$

is typical of the case where weakly evaluated expressions are constructed. In this case $t_1 = t_3, b_1 = \epsilon, t_2 = t_4 \rightarrow^\epsilon t_3 \times t_4$ and taking $t_2^- = t_2$ gives the desired result. The other case where

$$
\begin{aligned}
&w_1 = \texttt{fst} : t_3 \times t_4 \rightarrow^\epsilon t_3 \\
&w_2 = \langle \texttt{pair} : t_3' \rightarrow^\epsilon t_4' \rightarrow^\epsilon t_3' \times t_4'\ w_a\ w_b \rangle \\
&w_3 = w_a
\end{aligned}
$$

is typical of the case where weakly evaluated expressions are taken apart. In this case $t_1 = t_3 \times t_4, b_1 = \epsilon, t_2 = t_3, t_1^- = t_3' \times t_4'$ and it follows that $t_3' \leq t_2$. From $cenv \vdash w_2 \mid t_1^- \ \& \ \epsilon$ we then get $cenv \vdash w_a \mid t_3^{'-} \ \& \ \epsilon$ where $t_3^{'-} \leq t_3'$ and we have used Fact B.1. Taking $t_2^- = t_3^-$ then gives the desired result. □

## Proof of Proposition 4.5

We proceed by induction on the transition relation for matching.

*The axiom for send and receive.* In this case

$$
\begin{aligned}
&w_1 = \langle \texttt{send} : (t_{11} \rightarrow^\epsilon t_1\ \texttt{com}\ b_1) \langle \texttt{pair} : t_{12}\ ci\ w \rangle \rangle \\
&w_2 = \langle \texttt{receive} : (t_{21} \rightarrow^\epsilon t_2\ \texttt{com}\ b_2) ci \rangle
\end{aligned}
$$

where it is essential that the two occurrences of $ci$ are indeed the same channel identifier. It is immediate from the typing rule for weakly evaluated constants that $t_1 = t_{01}, t_2 = t_{02}$, $b_1 = b_{01}$ and $b_2 = b_{02}$. Furthermore, $b_1$ may be written $r_1!t_1$ and $b_2$ may be written $r_2?t_2$. Finally, write $cenv(ci) = t_0\ \texttt{chan}\ r_0$.

Turning the attention to $w$ and $ci$ of $w_1$ it follows from

$$cenv \vdash w_1 \mid (t_{01}\ \texttt{com}\ b_{01})\ \&\ \epsilon$$

that

$$
\begin{aligned}
&cenv \vdash w \mid t_{01}^- \ \&\ \epsilon \\
&cenv \vdash ci \mid (t_{01}'\ \texttt{chan}\ r_1^-)\ \&\ \epsilon
\end{aligned}
$$

for some $t_{01}^- \leq t_{01}, t'_{01} \equiv t_{01}$ and $r_1^- \leq r_1$. (The detailed argument observes that $t_{11}$ must have the form $(t_1 \texttt{ chan } r_1) \times t_1$ with $b_1$ as above, and hence $t_{12}$ must have the form $t_3 \rightarrow^\epsilon t_4 \rightarrow^\epsilon t_3 \times t_4$ with $t_3 \leq t_1 \texttt{ chan } r_1$ and $t_4 \leq t_1$, and hence $cenv \vdash ci \mid t_3^- \ \& \ \epsilon$ and $cenv \vdash w \mid t_4^- \ \& \ \epsilon$ for some $t_3^- \leq t_3$ and $t_4^- \leq t_4$.)

Turning the attention to $ci$ of $w_2$ it follows from

$$cenv \vdash w_2 \mid (t_{02} \texttt{ com } b_{02}) \ \& \ \epsilon$$

that

$$cenv \vdash ci \mid (t'_{02} \texttt{ chan } r_2^-) \ \& \ \epsilon$$

for some $t'_{02} \equiv t_{02}$ and $r_2^- \leq r_2$. By the typing axiom for identifiers it follows that $t'_{01} = t_0 = t'_{02}$ and $r_1^- = r_0 = r_2^-$ . A consequence of this is that $t_{01} \equiv t_{02}$

Taking

$$t_{02}^- = t_{01}^-, b'_1 = \epsilon \text{ and } b'_2 = \epsilon$$

we then get

$$cenv \vdash w \mid t_{01}^- \ \& \ b'_1 \text{ with } b_1; b'_1 \leq b_{01}$$
$$cenv \vdash w \mid t_{02}^- \ \& \ b'_2 \text{ with } b_2; b'_2 \leq b_{02}$$

and we also have $t_{01}^- \leq t_{01}$ and $t_{02}^- \leq t_{02}$ (since $t_{02}^- = t_{01}^-, t_{01}^- \leq t_{01}, t_{01} \equiv t_{02}$). Furthermore, $b_1 = r_1!t_1$ and $b_2 = r_2?t_2$ with $t_1 \equiv t_2$ and $\exists r : r \leq r_1 \wedge r \leq r_2$.

*The rule for choosing heads.* In this case

$$w_1 = \langle \texttt{choose} : t_{11} \langle \texttt{cons} : t_{12} \ w_{11} \ w_{12} \rangle \rangle$$

It is immediate from the typing rule for weakly evaluated constants to infer from

$$cenv \vdash w_1 \mid (t_{01} \texttt{ com } b_{01}) \ \& \ \epsilon$$

that

$$cenv \vdash w_{11} \mid (t'_{01}\ \texttt{com}\ b'_{01})\ \&\ \epsilon$$

for some $t'_{01} \leq t_{01}$ and $b'_{01} \leq b_{01}$.

By assumption

$$(w_{11}, w_2) \rightsquigarrow (e_1, e_2) : (b_1, b_2)$$

and the induction hypothesis gives $t'^{-}_{01} \leq t'_{01}, t'^{-}_{02} \leq t_{02}, b'_1$ and $b'_2$ such that

$$cenv \vdash e_1 \mid t'^{-}_{01}\ \&\ b'_1 \text{ with } b_1; b'_1 \leq b'_{01}$$
$$cenv \vdash e_2 \mid t^{-}_{02}\ \&\ b'_2 \text{ with } b_2; b'_2 \leq b_{02}$$

By taking $t^{-}_{01} = t'^{-}_{01}$ it is immediate that

$$cenv \vdash e_1 \mid t^{-}_{01}\ \&\ b'_1 \text{ with } b_1; b'_1 \leq b_{01}$$
$$cenv \vdash e_2 \mid t^{-}_{02}\ \&\ b'_2 \text{ with } b_2; b'_2 \leq b_{02}$$

and that also $t^{-}_{01} \leq t_{01}$ and $t^{-}_{02} \leq t_{02}$

*The rule for choosing tails.* In this case

$$w_1 = \langle \texttt{choose} : t_{11} \langle \texttt{cons} : t_{12}\ w_{11}\ w_{12} \rangle \rangle$$

and much as before

$$cenv \vdash \langle \texttt{choose} : t_{11}\ w_{12} \rangle \mid (t_{01}\ \texttt{com}\ b_{01})\ \&\ \epsilon$$

(except that there is no need to consider $t'_{01} \leq t_{01}$ and $b'_{01} \leq b_{01}$). By assumption

$$(\langle \texttt{choose} : t_{11}\ w_{12} \rangle, w_2) \rightsquigarrow (e_1, e_2) : (b_1, b_2)$$

and the induction hypothesis gives $t^{-}_{01} \leq t_{01},\ t^{-}_{02} \leq t_{02},\ b'_1$ and $b'_2$ such that

$$cenv \vdash e_1 \mid t_{01}^- \;\&\; b_1' \text{ with } b_1; b_1' \leq b_{01}$$
$$cenv \vdash e_2 \mid t_{02}^- \;\&\; b_2' \text{ with } b_2; b_2' \leq b_{02}$$

This is indeed the desired result.

*The rule for wrap.* In this case

$$w_1 = \langle \texttt{wrap} : t_{11} \langle \texttt{pair} : t_{12}\ w_{11}\ w_{12} \rangle \rangle$$

The type $t_{11}$ must be of the form

$$t_{11} = (t_3\ \texttt{com}\ b_3) \times (t_3 \rightarrow^{b_4} t_4) \rightarrow^{\epsilon} (t_4\ \texttt{com}\ (b_3; b_4))$$

and from

$$cenv \vdash w_1 \mid (t_{01}\ \texttt{com}\ b_{01}) \;\&\; \epsilon$$

it follows that $t_{01} = t_4$ and $b_{01} = b_3; b_4$. It further follows that

$$cenv \vdash w_{11} \mid (t_3^-\ \texttt{com}\ b_3^-) \;\&\; \epsilon$$
$$cenv \vdash w_{12} \mid (t_3^+ \rightarrow^{b_4^-} t_4^-) \;\&\; \epsilon$$

for some $t_3^- \leq t_3, b_3^- \leq b_3, t_3^+ \leq t_3, b_4^- \leq b_4$ and $t_4^- \leq t_4$.

By assumption

$$(w_{11}, w_2) \rightsquigarrow (e_1, e_2) : (b_1, b_2)$$

and the induction hypothesis yields $t_3^{--} \leq t_3^-, t_{02}^- \leq t_{02}, b_3^{-'}$ and $b_2'$ such that

$$cenv \vdash e_1 \mid t_3^{--} \;\&\; b_3^{-'} \text{ with } b_1; b_3^{-'} \leq b_3^-$$
$$cenv \vdash e_2 \mid t_{02}^- \;\&\; b_2' \text{ with } b_2; b_2' \leq b_{02}$$

It then follows that

$$cenv \vdash w_{12}\ e_1 \mid t_4^- \;\&\; (b_3^{-'}; b_4^-) \text{ with } b_1; b_3^{-'}; b_4^- \leq b_3^-; b_4^-$$

and taking $t_{01}^- = t_4^-$ and $b_1' = b_3^{-'}; b_4$ we have

$$cenv \vdash w_{12}\ e_1 \mid t_{01}^- \ \&\ b_1' \text{ with } b_1; b_1' \leq b_{01}$$
$$cenv \vdash e_2 \mid t_{02}^- \ \&\ b_2' \text{ with } b_2; b_2' \leq b_{02}$$

as well as $t_{01}^- \leq t_{01}$ and $t_{02}^- \leq t_{02}$

*The rule for rearranging the components.* This case is straightforward (due to the symmetrical formulation of the proposition). □

## Proof of Lemma 4.7

We proceed by induction on the structure of the evaluation context $E$.

*The case* $E ::= [\,]$. This is immediate since $t = t_0$ and $b = b_0$ and we may therefore take $t' = t_0'$ and $b' = b_0'$.

*The case* $E ::= E_0\ e$. Inspection of $cenv \vdash E[e_0] \mid t \ \&\ b$ reveals premisses of the form

$$cenv \vdash E_0[e_0] \mid t_1 \rightarrow^{b_3} t \ \&\ b_1$$
$$cenv \vdash e \mid t_1^- \ \&\ b_2$$

where $t_1^- \leq t_1$ and $b = b_1; b_2; b_3$. From the induction hypothesis we get

$$cenv \vdash E_0[e_0'] \mid t_1^+ \rightarrow^{b_3^-} t^- \ \&\ b_1'$$

where $t_1^+ \geq t_1, b_3^- \leq b_3, t^- \leq t$ and $b^\bullet; b_1' \leq b_1$. Taking $t' = t^-$ and $b' = b_1'; b_2; b_3$ we then have

$$cenv \vdash E_0[e_0']e \mid t^- \ \&\ b'$$

as well as the desired $t' \leq t$ and $b^\bullet; b' \leq b$.

*The case* $E ::= w\ E_0$. Inspection of $cenv \vdash E[e_0] \mid t \ \&\ b$ reveals premisses of the form

$$cenv \vdash w \mid t_1^+ \rightarrow^{b^2} t \;\&\; \epsilon$$
$$cenv \vdash E_0[e_0] \mid t_1 \;\&\; b_1$$

where $t_1^+ \leq t_1$ and $b = \epsilon; b_1; b_2$ and we also used Fact B.1. From the induction hypothesis we get

$$cenv \vdash E_0[e_0'] \mid t_1^- \;\&\; b_1'$$

where $t_1^- \leq t_1$ and $b^\bullet; b_1' \leq b_1$. Taking $t' = t$ and $b' = \epsilon; b_1'; b_2$ we then have

$$cenv \vdash wE_0[e_0'] \mid t' \;\&\; b'$$

as well as the desired $t' \leq t$ and $b^\bullet; b' \leq b$.

*The case* $E ::= \texttt{let}\ i = E_0\ \texttt{in}\ e$. Inspection of $cenv \vdash E[e_0] \mid t \;\&\; b$ reveals premisses of the form

$$cenv \vdash E_0[e_0] \mid t_1 \;\&\; b_1$$
$$cenv[i \mapsto t_1] \vdash e \mid t \;\&\; b_2$$

where $b = b_1; b_2$. From the induction hypothesis we get

$$cenv \vdash E_0[e_0'] \mid t_1^- \;\&\; b_1'$$

with $t_1^- \leq t_1$ and $b^\bullet; b_1' \leq b_1$. From Lemma 4.3 we get

$$cenv[i \mapsto t_1^-] \vdash e \mid t^- \;\&\; b_2$$

where $t^- \leq t$ and $b_2^- \leq b_2$. Taking $t' = t^-$ and $b' = b_1'; b_2^-$ we then have

$$cenv \vdash \texttt{let}\ i = E_0[e_0']\ \texttt{in}\ e \mid t' \;\&\; b'$$

as well as the desired $t' \leq t$ and $b^\bullet; b' \leq b$. (Note that the "decrease" in behaviour may apparently take place "deeply embedded" in the behaviour rather than only at the front!)

*The case* $E ::= \texttt{if}\ E_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : t$. (Note that no confusion arises from using the type $t$ here.) Inspection of $cenv \vdash E[e_0] \mid t \;\&\; b$ reveals premisses of the form

$$cenv \vdash E_0[e_0] \mid \texttt{bool} \;\&\; b_1$$
$$cenv \vdash e_1 \mid t_1 \;\&\; b_2$$
$$cenv \vdash e_2 \mid t_2 \;\&\; b_3$$

where $t_1 \leq t, t_2 \leq t$ and $b = b_1; (b_2 + b_3)$. From the induction hypothesis we get

$$cenv \vdash E_0[e_0'] \mid \texttt{bool} \;\&\; b_1$$

where $b^\bullet, b_1' \leq b_1$ because the condition $t_4 \leq \texttt{bool}$ is equivalent to $t_4 = \texttt{bool}$. Taking $t' = t$ and $b' = b_1'; (b_2 + b_3)$ we then have

$$cenv \vdash \texttt{if}\ E_0[e_0']\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : t \mid t \;\&\; b'$$

as well as the desired $t' \leq t$ and $b^\bullet; b' \leq b$. □

## Proof of Proposition 4.6

The proof is by cases on the rule used for the concurrent transition.

*Sequential evaluation.* In this case we have $\vec{b} = \epsilon$ and $\vec{pi} = pi$ for some $pi \in$ **PIdent** and

$$PP' = PP[pi \mapsto PP'(pi)]$$
$$cenv' = cenv$$
$$PP(pi) \to PP'(pi)$$

From $cenv \vdash PP(pi) \mid PT(pi) \;\&\; PB(pi)$ and Proposition 4.1 we get $t' \leq PT(pi)$ and $b' \leq PB(pi)$ such that $cenv \vdash PP'(pi) \mid t' \;\&\; b'$. Taking

$$PT' = PT[pi \mapsto t']$$
$$PB' = PB[pi \mapsto b']$$

all conditions are satisfied.

*Channel allocation.* In this case we have $\vec{b} = b = t_0$ CHAN $i_0$ and $\vec{pi} = pi$ for some $pi \in$ **Pident** and

$$
\begin{array}{l}
PP' = pp[pi \mapsto PP'(pi)] \\
cenv' = cenv[ci \mapsto t_0 \ \texttt{chan} \ i_0] \\
PP(pi) = E[\texttt{channel} : (\texttt{unit} \rightarrow^b t)()] \\
PP(pi) = E[ci]
\end{array}
$$

where $ci \notin \text{dom } (cenv)$ and $t = t_0 \ \texttt{chan} \ i_0$. From

$$cenv \vdash E[\texttt{channel} : (\texttt{unit} \rightarrow^b t) \ ()] \mid PT(pi) \ \& \ PB(pi)$$

we immediately get

$$cenv' \vdash E[\texttt{channel} : (\texttt{unit} \rightarrow^b t) \ ()] \mid PT(pi) \ \& \ PB(pi)$$

and it is immediate that

$$
\begin{array}{l}
cenv' \vdash \texttt{channel} : (\texttt{unit} \rightarrow^b t) \ ()] \mid t \ \& \ \epsilon; \epsilon; b \\
cenv' \vdash ci \mid t \ \& \ \epsilon
\end{array}
$$

Using Lemma 4.7 we then get $t'$ and $b'$ such that

$$cenv' \vdash E[ci] \mid t' \ \& \ b'$$

and where $t' \leq t$ and $b; b' \leq PB(pi)$. Taking

$$
\begin{array}{l}
PT' = PT[pi \mapsto t'] \\
PB' = PB[pi \mapsto b']
\end{array}
$$

all conditions are satisfied.

*Process creation.* In this case we have $\vec{b} = b = t_0 \ \text{FORK} \ b_0$ and $\vec{pi} = pi_1, pi_2$ for some $pi_1, pi_2 \in \textbf{PIdent}$ and

$$
\begin{array}{l}
PP' = PP[pi_1 \mapsto PP'(pi_1)][pi_2 \mapsto PP'(pi_2)] \\
cenv' = cenv \\
PP(pi_1) = E[\texttt{fork} : (t \rightarrow^b \texttt{unit})w] \\
PP'(pi_1) = E[()] \\
PP'(pi_2) = w \ ()
\end{array}
$$

where $pi_2 \notin \mathrm{dom}(PP)$ and $t = \mathtt{unit} \rightarrow^{b_0} t_0$. From

$$cenv \vdash E[\mathtt{fork}\,:(t \rightarrow^{b} \mathtt{unit})w] \mid PT(pi_1)\&PB(pi_1)$$

we get

$$cenv \vdash \mathtt{fork} : (t \rightarrow^{b} \mathtt{unit})w] \mid \mathtt{unit}\ \&\ \epsilon;\epsilon;b$$

and hence

$$cenv \vdash w \mid \mathtt{unit} \rightarrow^{b_0^-}\ t_0^-\ \&\ \epsilon$$

where $b_0^- \leq b_0$ and $t_0^- \leq t_0$ and we have used Fact B.1. Since

$$cenv \vdash () \mid \mathtt{unit}\ \&\epsilon$$

we get $t'$ and $b'$ from Lemma 4.7 such that

$$cenv \vdash E[()] \mid t'\ \&\ b'$$

and where $t' \leq PT(pi_1)$ and $b;b' \leq PB(pi_1)$. Taking

$$PT' = PT[pi_1 \mapsto t'][pi_2 \mapsto t_0^-]$$
$$PB' = PB[pi_1 \mapsto b'][pi_2 \mapsto \epsilon;\epsilon;b_0^-]$$

all conditions are satisfied.

*Synchronization.* In this case we have $\vec{b} = b_1, b_2$ and $pi = pi_1, pi_2$ for some $pi \in$ **PIdent** and

$$PP' = PP[pi_1 \mapsto PP'(pi_1)][pi_2 \mapsto PP'(pi_2)]$$
$$cenv' = cenv$$
$$PP(pi_1) = E_1[\mathtt{sync} : (t_1\ \mathtt{com}\ b_1 \rightarrow^{b_1} t_1)w_1]$$
$$PP(pi_2) = E_2[\mathtt{sync} : (t_2\ \mathtt{com}\ b_2 \rightarrow^{b_2} t_2)w_2]$$
$$PP'(pi_1) = E_1[e_1]$$
$$PP'(pi_2) = E_2[e_2]$$
$$(w_1, w_2) \rightsquigarrow (e_1, e_2) : (b_1, b_2)$$

For $j \in \{1,2\}$ we have

$$cenv \vdash E_j[\texttt{sync} : (t_j \ \texttt{com} \ b_j \rightarrow^{b_j} t_j)w_j] \mid PT(pi_j) \ \& \ PB(pi_j)$$

and get

$$cenv \vdash \texttt{sync} : (t_j \ \texttt{com} \ b_j \rightarrow^{b_j} t_j)w_j \mid t_j \ \& \ \epsilon;\epsilon;b_j$$

and hence

$$cenv \vdash w_j \mid t_j^- \ \texttt{com} \ b_j^- \ \& \ \epsilon$$

where $t_j^- \leq t_j$ and $b_j^- \leq b_j$ and we have used Fact B.1.

Proposition 4.5 then gives $t_1'', t_2'', b_1''$ and $b_2''$ such that for $j \in \{1,2\}$ we have

$$cenv \vdash e_j \mid t_j''\&b_j''$$

with $t_j'' \leq t_j^-$ and $b_j;b_j'' \leq b_j$. Using Lemma 4.7 we then get $t_1', t_2', b_1'$ and $b_2'$ such that for $j \in \{1,2\}$ we have

$$cenv \vdash E_j[e_j] \mid t_j' \ \& \ b_j'$$

with $t_j' \leq PT(pi_j)$ and $b_j;b_j' \leq PB(pi_j)$. Taking

$$PT' = PT[pi_1 \mapsto t_1'][pi_2 \mapsto t_2']$$
$$PB' = PB[pi_1 \mapsto b_1'][pi_2 \mapsto b_2']$$

all conditions are satisfied (and $b_1$ and $b_2$ are as stated in Proposition 4.5). □