Typed Event Structures and the π-Calculus

Daniele Varacca, Nobuko Yoshida
Imperial College London

November 8, 2005

Abstract. We propose a typing system for the true concurrent model of event structures that guarantees an interesting behavioural property known as *confusion freeness*. A system is confusion free if nondeterministic choices are localised and do not depend on the scheduling of independent components. It is a generalisation of confluence to systems that allow nondeterminism. Ours is the first typing system to control behaviour in a true concurrent model. To demonstrate its applicability, we show that typed event structures give a semantics of linearly typed version of the π-calculi with internal mobility. The semantics we provide is the first event structure semantics of the π-calculus and generalises Winskel's original event structure semantics of CCS.

1 Introduction

Models for concurrency can be classified according to different criteria. One possible classification distinguishes between *interleaving* models and *causal* models (also known as *true concurrent* models). In interleaving models, concurrency is reduced to the nondeterministic choice between all possible sequential schedulings of the concurrent actions. Instances of such models are *traces* and *labelled transition systems* [39]. In casual models, causality, concurrency and conflict are explicitly represented. Instances of such models are *Petri nets* [30], *Mazurkiewicz traces* [23] and *event structures* [28].

Interleaving models are very successful in defining observational equivalence, by means of bisimulation [25]. Although bisimulation can be defined for true concurrent models too [19], it has not been used to characterise interesting observational congruences. The reason lies in the fact that true concurrent models explicitly represent causality, which is arguably not an observable property. On the other hand, true concurrent models can easily represent interesting behavioural properties of a given system: absence of conflict, independence of the choices and sequentiality [30]. Some of these properties are complicated to express in interleaving models.

In this paper we address a particular true concurrent model: the model of *event structures* [28, 36]. Event structures have been used to give semantics to concurrent process languages. Possibly the earliest and the most intuitive is Winskel's semantics of Milner's CCS [35].

The first contribution of this paper is to present a compositional typing system for event structures that ensures an important behavioural property: *confusion freeness*. This property was first identified in the context the theory of Petri nets [30]. It has been studied in that context, in the form of free choice nets [12]. Confusion free event structures are also known as *concrete data structures* [4], and their domain-theoretic counterpart are the *concrete domains* [20]. Finally, confusion freeness has been recognised as an important property in the context of probabilistic models [32, 1].

To illustrate this important notion, let us suppose that a system is composed of two processes P and Q. Suppose the system can reach a state where P has a choice between two different actions a_1, a_2, and where Q, independently, can perform action b. We say that such a state is *confused* if the occurrence of b changes the choices available to P (for instance by disabling a_2, or by enabling a third action a_3). Intuitively the choice of process P is not local to that process in that it can be influenced by an independent action. We say that a system is *confusion free* if none of its reachable states is confused. Confusion freeness is a generalisation of *confluence* to systems that allow nondeterminism. It is best expressed within a true concurrent model. The new typing system guarantees that all typable event structures are confusion free. Moreover, a restricted form of types guarantees the stronger property of *conflict freeness*, which is, in a sense, the true concurrent version of confluence.

The second contribution of this paper is to give the first sound event structure semantics of a fragment of the π-calculus [26]. Various causal semantics of the π-calculus exist [18, 8, 13, 5, 11, 9], but none is given in terms of event structures. The technical difficulty in extending CCS semantics to the π-calculus lies in the handling of α-conversion, which is the main ingredient to represent dynamic creation of names. We are able to solve this problem for a restricted version of the π-calculus, a linearly typed version of Sangiorgi's πI-calculus [31, 41]. This fragment is expressive enough to encode the typed λ-calculus (in fact, to encode it *fully abstractly* [41]). We argue that in this fragment, α-conversion need not be performed dynamically (at "run time"), but can be done during the typing (at "compile time"), by choosing in advance all the names that will be created during the computation. This is possible because the typing system guarantees that, in a sense, every process knows in advance which processes it will communicate with.

To substantiate this intuition, we soundly encode the linearly typed fragment of the π-calculus into an intermediate process language, which is syntactically similar to the π-calculus except that α-conversion is not allowed. We devise a typing system for this language that makes use of the event structure types. We then provide the language with a semantics in terms of typed event structures. Via this intermediate translation, we thus obtain a sound event structure semantics of the π-calculus, which follows the same lines as Winskel's; syntactic nondeterministic choice is modelled by *conflict*, prefix is modelled using *causality*, and parallel composition generates *concurrent* events. Moreover, since our semantics is given in terms of typed event structures, we obtain that all processes of this fragment are confusion free, and this is the first time a causal model has been used to prove a behavioural property of a process language. Our typing system generalises an early idea by Milner, who devised a syntactic restriction of CCS (a kind of a typing system) that guarantees confluence of the interleaving semantics [25]. As a corollary of our work we show that a similar restriction applied to the π-calculus guarantees the property of conflict freeness.

The tight correspondence between the linear π-calculus and programming language semantics opens the door for event structure semantics to the λ-calculus and other functional and imperative languages.

Structure of the paper Section 2 introduces the basic definitions of event structures and defines formally the notion of confusion freeness. The original work of the section consists in the description of a novel characterisation of the product in the category of event structures. The product of event structures is one of the base ingredients in the definition of the parallel composition. Our characterisation allows us to carry out the proofs in the following sections. Section 3 presents our new typing system and an event structure semantics of the types. We then define a notion of typing of event structures by means of the morphisms of the category of event structures. Typed event structures are confusion free by definition. The main theorem of this section is that the parallel composition of typed event structures is again typed, and thus confusion free. In Section 4, we present the intermediate process language which is used to bridge between the typed event structures and the linear π-calculus. We call this calculus *Name Sharing CCS* or NCCS. We define a notion of typing for NCCS processes and its typed operational semantics. In Section 5, we give a semantics of typed NCCS processes in terms of event structures. The main result of this section is that the semantics of a typed process is a typed event structure. We also show that this semantics is sound with respect to bisimulation. Section 6 presents a linearly typed version of the πI-calculus. This section is inspired from [41], but our fragment is extended to allow nondeterministic choice. In Section 7, we provide a sound translation of the the typed πI-calculus, into NCCS. Through the sound event structure semantics of NCCS, we obtain a sound semantics of the π-calculus in terms of event structures. All proofs can be found in the Appendix.

2 Event structures

Event structures were introduced by Nielsen, Plotkin and Winskel [28, 34], and have been subject of several studies since. They appear in different forms. The one we introduce in this work is sometimes referred to as *prime event structures* [36]. For the relations of event structures with other models for concurrency, the standard reference is [39].

2.1 Basic definitions

An *event structure* is a triple $\mathscr{E} = \langle E, \leq, \smile \rangle$ such that

- E is a countable set of *events*;
- $\langle E, \leq \rangle$ is a partial order, called the *causal order*;
- for every $e \in E$, the set $[e) := \{e' \mid e' < e\}$, called the *enabling set* of e, is finite;
- $\smile$ is an irreflexive and symmetric relation, called the *conflict relation*, satisfying the following: for every $e_1, e_2, e_3 \in E$ if $e_1 \leq e_2$ and $e_1 \smile e_3$ then $e_2 \smile e_3$.

The reflexive closure of conflict is denoted by $\asymp$. We say that the conflict $e_2 \smile e_3$ is *inherited* from the conflict $e_1 \smile e_3$, when $e_1 < e_2$. If a conflict $e_1 \smile e_2$ is not inherited from any other conflict we say that it is *immediate*, denoted by $e_1 \smile_\mu e_2$. The reflexive closure of immediate conflict is denoted by $\asymp_\mu$. Causal order and conflict are mutually exclusive. If two events are not causally related nor in conflict they are said to be *concurrent*. The set of maximal elements of $[e)$ is denoted by $parents(e)$.

A *configuration* x of an event structure $\mathscr{E}$ is a conflict free downward closed subset of E, i.e. a subset x of E satisfying: (1) if $e \in x$ then $[e) \subseteq x$ and (2) for every $e, e' \in x$, it is not the case that $e \smile e'$. Therefore, two events of a configuration are either causally dependent or concurrent, i.e., a configuration represents a run of an event structure where events are partially ordered. The set of configurations of $\mathscr{E}$, partially ordered by inclusion, is denoted as $\mathscr{L}(\mathscr{E})$. It is a coherent ω-algebraic domain [28], whose compact elements are the finite configurations.

If x is a configuration and e is an event such that $e \notin x$ and $x \cup \{e\}$ is a configuration, then we say that e is *enabled* at x. Two configurations x, x' are said to be *compatible* if $x \cup x'$ is a configuration. For every event e of an event structure $\mathscr{E}$, we define $[e] := [e) \cup \{e\}$. It is easy to see that both $[e]$ and $[e)$ are configurations for every event e and that therefore any event e is enabled at $[e)$.

A *labelled event structure* is an event structure $\mathscr{E}$ together with a labelling function $\lambda : E \to L$, where L is a set of labels. Events should be though of as occurrences of actions. Labels allow us to identify events which represent different occurrences of the same action. Labels are also essential in defining the parallel composition, and will also play a major role in the typed setting. Given a labelled event structure $\mathscr{E} = \langle E, \leq, \smile, \lambda \rangle$ we generate a labelled transition system $TS(\mathscr{E})$ as follows: states are configurations, and $x \xrightarrow{a} x'$ if $x' = x \uplus \{e\}$ and $\lambda(e) = a$.

2.2 Conflict free and confusion free event structures

A interesting subclass of event structures is the following.

Definition 2.1. *An event structure is* conflict free *if its conflict relation is empty.*

It is easy to verify that in conflict free event structures, every two configurations are compatible. More generally, if $\mathscr{E}$ is conflict free, then $\mathscr{L}(\mathscr{E})$ is a complete lattice. Conflict freeness is the true concurrent version of confluence. Indeed it is easy to verify that if $\mathscr{E}$ is conflict free, then $TS(\mathscr{E})$ is confluent.

We introduce another interesting class of event structures where every choice is *localised*. To specify what "local" means in this context, we need the notion of *cell*, a set of events that are pairwise in immediate conflict and have the same enabling sets.

Definition 2.2. *A* partial cell *is a set c of events such that $e, e' \in c$ implies $e \asymp_\mu e'$ and $[e) = [e')$. A maximal partial cell is called a* cell. *We say that $\asymp_\mu$ is* cellular *if $e \asymp_\mu e' \Longrightarrow [e) = [e')$.*

To avoid this, it is enough to assume that cells are closed under immediate conflict.

Definition 2.3. *An event structure is* confusion free *if its cells are closed under immediate conflict.*

The above definition was introduced in [32]. It is equivalent to the more traditional definition, which we state below.

Proposition 2.4. *An event structure is confusion free if and only if the relation $\asymp_\mu$ is transitive and cellular.*

It follows that, in a confusion free event structure, the relation $\asymp_\mu$ is an equivalence with cells being its equivalence classes.

2.3 A category of event structures

Event structures form the class of objects of a category [39]. The morphisms are defined as follows. Let $\mathscr{E}_1 = \langle E_1, \leq_1, \smile_1 \rangle$, $\mathscr{E}_2 = \langle E_2, \leq_2, \smile_2 \rangle$ be two event structures. A *morphism* $f : \mathscr{E}_1 \to \mathscr{E}_2$ is a partial function $f : E_1 \to E_2$ such that

- f preserves configurations: if x is a configuration of $\mathscr{E}_1$, then $f(x)$ is a configuration of $\mathscr{E}_2$;
- f is locally injective: let x be a configuration of $\mathscr{E}_1$, if $e, e' \in x$ and $f(e), f(e')$ are both defined with $f(e) = f(e')$, then $e = e'$.

It is straightforward to verify that the identity is a morphism and that morphisms compose, so that what we obtain is indeed a category.

Morphisms reflect conflict and causality and preserve concurrency. They can be equivalently characterised as follows.

Proposition 2.5 ([39]). *A partial function $f : E_1 \to E_2$ is a morphism of event structures $f : \mathscr{E}_1 \to \mathscr{E}_2$ if and only if the following are satisfied:*

- *f preserves downward closure: if $f(e_1)$ is defined, then $[f(e_1)] \subseteq f([e_1])$;*
- *f reflect reflexive conflict: if $f(e_1), f(e_2)$ are defined, and if $f(e_1) \asymp f(e_2)$, then $e_1 \asymp e_2$.*

There are various ways of dealing with labels. For the general treatment we refer to [39]. Here we present the simplest notion: take two labelled event structures $\mathscr{E}_1 = \langle E_1, \leq_1, \smile_1, \lambda_1 \rangle$, $\mathscr{E}_2 = \langle E_2, \leq_2, \smile_2, \lambda_2 \rangle$ on the same set of labels L. A morphism $f : \mathscr{E}_1 \to \mathscr{E}_2$ is said to be *label preserving* if, whenever $f(e_1)$ is defined, $\lambda_2(f(e_1)) = \lambda_1(e_1)$.

2.4 Operators on event structures

We can define several operations on labelled event structures.

- prefixing $a.\mathscr{E}$, where $\mathscr{E} = \langle E, \leq, \smile, \lambda \rangle$. It is the event structure $\langle E', \leq', \smile', \lambda' \rangle$, where $E' = E \uplus \{e'\}$ for some new event e', $\leq'$ coincides with $\leq$ on E and moreover, for every $e \in E$ we have $e' \leq e$, the conflict relation $\smile'$ coincides with $\smile$, that is e' is in conflict with no event. Finally λ' coincides with λ on E and $\lambda'(e') = a$. Intuitively, we add a new initial event, labelled by a.

- prefixed sum $\sum_{i \in I} a_i.\mathscr{E}_i$, where $\mathscr{E}_i = \langle E_i, \leq_i, \smile_i, \lambda_i \rangle$. This is obtained by disjoint union of copies of the event structures $a_i.\mathscr{E}_i$, where the order relation is the disjoint union of the orders, the labelling function is the disjoint union of the labelling functions, and the conflict is the disjoint union of the conflicts extended by putting in conflict every two events in two different copies. This is a generalisation of prefixing, where we add an initial *cell*, instead of an initial event.
- restriction $\mathscr{E} \setminus X$ where $\mathscr{E} = \langle E, \leq, \smile, \lambda \rangle$ and $X \subseteq A$ is a set of labels. This is obtained by removing from E all events with label in X and all events that are above one of those. On the remaining events, order, conflict and labelling are unchanged.
- relabelling $\mathscr{E}[f]$. This is just composing the labelling function λ with a function $f : L \to L$. The new event structure has thus labelling function $f \circ \lambda$.

It is easy to verify that all these constructions preserve the class of confusion free event structures. Also, with the obvious exception of the prefixed sum, they preserve the class of conflict free event structures

2.5 The parallel composition

The parallel composition of event structures is difficult to define. In [39] is defined as the categorical product followed by restriction and relabelling. The existence of the product is deduced via general categorical arguments, but not explicitly constructed. In order to carry out our proofs, we needed a more concrete representation of the product. We have devised such a representation, which is inspired by the one given in [10], but which is more suitable to an inductive reasoning.

Let $\mathscr{E}_1 := \langle E_1, \leq_1, \smile_1 \rangle$ and $\mathscr{E}_2 := \langle E_2, \leq_2, \smile_2 \rangle$ be two event structures. Let $E_i^* := E_i \uplus \{*\}$. Consider the set $\tilde{E}$ obtained as the initial solution of the equation $X = \mathscr{P}_{fin}(X) \times E_1^* \times E_2^*$. Its elements have the form (x, e_1, e_2) for x finite, $x \subseteq \tilde{E}$. Initiality allows us to define inductively a notion of *height* of an element of $\tilde{E}$.

$$h(\emptyset, e_1, e_2) = 0$$
$$h(x, e_1, e_2) = \max\{h(e) \mid e \in x\} + 1$$

Most of our reasoning will be by induction on the height of the elements. We now carve out of $\tilde{E}$ a set E which will be the support of our product event structure $\mathscr{E}$. At the same time we define the order relation and the conflict relation on $\mathscr{E}$.

Base: we have that $(\emptyset, e_1, e_2) \in E$ if

- $e_1 \in E_1, e_2 \in E_2$, and e_1 minimal in E_1, e_2 minimal in E_2 or
- $e_1 \in E_1, e_2 = *$ and e_1 minimal in E_1 or
- $e_1 = *, e_2 \in E_2$ and e_2 minimal in E_2.

The order on the elements of height 0 is trivial.

Finally we have $(\emptyset, e_1, e_2) \asymp (\emptyset, d_1, d_2)$ if $e_1 \asymp d_1$ or $e_2 \asymp d_2$.

Inductive Case: assume that all elements in E of height $\leq n$ have been defined. Assume that an order relation and a conflict relation has been defined on them. Let (x, e_1, e_2) of height $n+1$. Let y be the set of maximal elements of x. Let $y_1 = \{d_1 \in E_1 \mid (z, d_1, d_2) \in y\}$ and $y_2 = \{d_2 \in E_2 \mid (z, d_1, d_2) \in y\}$, be the projections of y onto the two components. We have that $(x, e_1, e_2) \in E$ if x is downward closed and conflict free, and furthermore:

- Suppose $e_1 \in E_1, e_2 = *$. Then it must be the case that $y_1 = parents(e_1)$.
- Suppose $e_2 \in E_2, e_1 = *$. Then it must be the case that $y_2 = parents(e_2)$.
- Suppose $e_1 \in E_1, e_2 \in E_2$. Then
  - if $(z, d_1, d_2) \in y$, then either $d_1 \in parents(e_1)$ or $d_2 \in parents(e_2)$;
  - for all $d_1 \in parents(e_1)$, there exists $(z, d_1, d_2) \in x$;
  - for all $d_2 \in parents(e_2)$ there exists $(z, d_1, d_2) \in x$.

– Let $x_1 = \{d_1 \in E_1 \mid (z, d_1, d_2) \in x\}$ and $x_2 = \{d_2 \in E_2 \mid (z, d_1, d_2) \in x\}$. Then for no $d_1 \in x_1$, $d_1 \asymp e_1$ and for no $d_2 \in x_2$, $d_2 \asymp e_2$.

The partial order is extended by $e \leq (x, e_1, e_2)$ if $e \in x$, or $e = (x, e_1, e_2)$. Note that if $e < e'$ then $h(e) < h(e')$.

Finally for the conflict, take $e = (x, e_1, e_2)$ and $d = (z, d_1, d_2)$, where either $h(e) = n+1$ or $h(d) = n+1$ or both. Then we define $e \smile d$ if one of the following holds:

– $e_1 \asymp d_1$ or $e_2 \asymp d_2$, and $e \neq d$;
– there exists $e' = (x', e'_1, e'_2) \in x$ such that $e'_1 \asymp d_1$ or $e'_2 \asymp d_2$, and $e' \neq d$;
– there exists $d' = (z', d'_1, d'_2) \in z$ such that $e_1 \asymp d'_1$ or $e_2 \asymp d'_2$, and $e \neq d'$;
– there exists $e \in x, d \in z$ such that $e \smile d$.

As the following lemma shows, some of the clauses above are redundant, but are kept for simplicity.

Lemma 2.6 (Stability). *If $(x, e_1, e_2), (x', e_1, e_2) \in E$ and $x \neq x'$, then there exist $d \in x, d' \in x'$ such that $d \smile d'$.*

Now we are ready to state the main new result of this section: take two event structures $\mathscr{E}_1, \mathscr{E}_2$, and let $\mathscr{E} = \langle E, \leq, \smile \rangle$ be defined as above. Then we have:

Theorem 2.7. *The structure $\mathscr{E}$ is an event structure and it is the categorical product of $\mathscr{E}_1, \mathscr{E}_2$.*

We will not make use of this fact, except that projections preserve configurations. However this theorem is necessary to fit in the general framework of models for concurrency, and to avoid building "ad hoc" models.

For event structures with labels in L, we make the convention that the labelling function of the product takes on the set $L_* \times L_*$, where $L_* := L \uplus \{*\}$. We define $\lambda(x, e_1, e_2) = (\lambda_1^*(e_1), \lambda_2^*(e_2))$, where $\lambda_i^*(e_i) = \lambda_i(e_i)$ if $e_i \neq *$, and $\lambda_i^*(*) = *$. A *synchronisation algebra S* is given by a partial binary operation $\bullet_S$ defined on L_* [39]. Given two labelled event structures $\mathscr{E}_1, \mathscr{E}_2$, the parallel composition $\mathscr{E}_1 \|_S \mathscr{E}_2$ is defined as the categorical product followed by restriction and relabelling: $(\mathscr{E}_1 \times \mathscr{E}_2 \setminus X)[f]$ where X is the set of pairs $(l_1, l_2) \in L_* \times L_*$ for which $l_1 \bullet_S l_2$ is undefined, while the function f : is defined as $f(l_1, l_2) = l_1 \bullet_S l_2$. The subscripts S are omitted when the synchronisation algebra is clear from the context.

The simplest possible synchronisation algebra is defined as $l \bullet * = * \bullet l = l$, and undefined in all other cases. In this particular case, the induced parallel composition can be represented as the disjoint union of the sets of events, of the causal orders, and of the conflict. This can be also generalised to an arbitrary family of event structures $(\mathscr{E}_i)_{i \in I}$. In such a case we denote the parallel composition as $\prod_{i \in I} \mathscr{E}_i$.

Parallel composition does not preserve in general the classes of conflict free and confusion free event structures. New conflict can be created through synchronisation. One of the main reasons to devise a typing system for event structures is to guarantee the preservation of these two important behavioural properties.

2.6 Examples

We collect in this section a series of examples, with graphical representation.

Example 2.8. Consider the following event structures $\mathscr{E}_1, \mathscr{E}_2, \mathscr{E}_3$, defined on the same set of events $E := \{a, b, c, d, e\}$. In $\mathscr{E}_1$, we have $a \leq b, c, d, e$ and $b \smile_\mu c,\ c \smile_\mu d,\ b \smile_\mu d$. In $\mathscr{E}_2$, we do not have $a \leq d$, while in $\mathscr{E}_3$, we do not have $b \smile_\mu d$. The three event structures are represented in Figure 1, where curly lines represent immediate conflict, while the causal order proceeds upwards along the straight lines.

The event structure $\mathscr{E}_1$ is confusion free, with three cells: $\{a\}, \{b, c, d\}, \{e\}$. In $\mathscr{E}_2$, there are four cells: $\{a\}, \{b, c\}, \{d\}, \{e\}$. $\mathscr{E}_2$ is not confusion free, because immediate conflict is not cellular. This is an example of *asymmetric* confusion [29]. In $\mathscr{E}_3$ there are four cells: $\{a\}, \{b, c\}, \{c, d\}, \{e\}$. $\mathscr{E}_3$ is not confusion free, because immediate conflict is not transitive. This is an example of *symmetric* confusion.

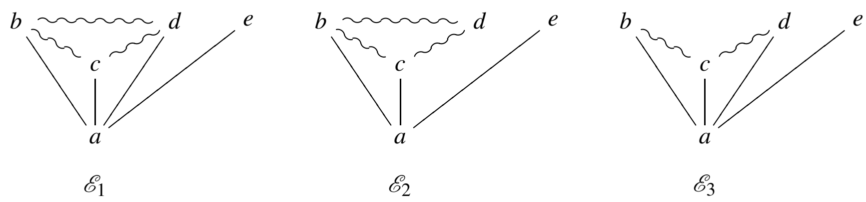

Fig. 1. Event structures

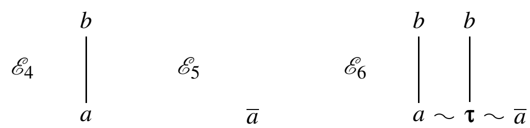

Fig. 2. Parallel composition of event structures

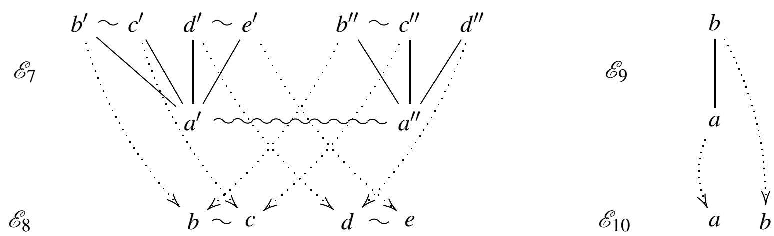

Fig. 3. Morphisms of event structures

Example 2.9. Next we show an example of parallel composition, see Figure 2. Consider the two labelled event structures $\mathscr{E}_4, \mathscr{E}_5$, where $E_4 = \{e_4, d_4\}, E_5 = \{e_5\}$, conflict and order being trivial, and $\lambda(e_4) = a, \lambda(d_4) = b, \lambda(e_5) = \overline{a}$. Consider the symmetric synchronisation algebra $a \bullet \overline{a} = \boldsymbol{\tau}$, $a \bullet * = a$, $\overline{a} \bullet * = \overline{a}$, $b \bullet * = b$ and undefined otherwise. Then $\mathscr{E}_6 := \mathscr{E}_4 \| \mathscr{E}_5$ is as follows: $E_6 = \{e := (\emptyset, e_4, *), e' := (\emptyset, *, E_5), e'' := (\emptyset, e_4, e_5), d := (\{e\}, d_4, *), d'' := (\{e''\}, d_4, *)\}$, with the ordering defined as $e \leq d, e'' \leq d''$, while the conflict is defined as $e \smile e''$, $e' \smile e''$, $e \smile d''$, $e' \smile d''$, $e'' \smile d$, $d \smile d''$. The labelling function is $\lambda(e) = a, \lambda(e') = \overline{a}, \lambda(e'') = \boldsymbol{\tau}, \lambda(d) = \lambda(d'') = b$. Note that, while $\mathscr{E}_4, \mathscr{E}_5$ are confusion free, $\mathscr{E}_6$ is not, since reflexive immediate conflict is not transitive. Note also an instance of the stability for d, d''.

Example 2.10. Finally we show an example of morphism. Consider the two event structures $\mathscr{E}_7, \mathscr{E}_8$ defined as follows:

- $E_7 = \{a', b', c', d', e', a'', b'', c'', d''\}$ with $a' \smile_\mu a''$ $b' \smile_\mu c', d' \smile_\mu e', b'' \smile_\mu c''$ and $a' \leq b', c', d', e'$ and $a'' \leq b'', c'', d''$.
- $E_8 = \{b, c, d, e\}$ with $b \smile_\mu c, d \smile_\mu e$, and trivial ordering.

Note that both $\mathscr{E}_7$ and $\mathscr{E}_8$ are confusion free.

We define a morphism $f : \mathscr{E}_7 \to \mathscr{E}_8$ by putting $f(x') = f(x'') = x$ for $x = b, c, d, e$ while f is undefined on a', a''. Note that b' and b'' are mapped to the same element b, and they are indeed in conflict, because they inherit the conflict $a' \smile a''$.

For another example consider the two event structures $\mathscr{E}_9, \mathscr{E}_{10}$, where $E_9 = E_{10} = \{a,b\}$, both have empty conflict, and in $\mathscr{E}_9$ we have $a \leq b$. The identity function on $\{a,b\}$ is a morphism $\mathscr{E}_9 \to \mathscr{E}_{10}$ but not vice versa. We can say that the causal order of $\mathscr{E}_9$ refines the causal order of $\mathscr{E}_{10}$.

3 Typed event structures

In this section we present a notion of types for an event structure, which are inspired from the types for the linear π-calculus [41, 3, 21]. We will clearly see this connection later, when we devise a process calculus that makes use of these types. The event structure which interprets a type records the causality between the names contained in the types. We then assign types to event structures by allowing a more general notion of causality.

3.1 Types and environments

We assume a countable set of *names*, ranged over by a,b,c,x,y,z. In this setting, names are used to identify "clusters" of events. Names will also be used in generating the labels of the event structure. Types, type environments, and the mode of a type are generated by the following grammar

$\Gamma,\Delta ::=$	$y_1 : \sigma_1, \ldots, y_n : \sigma_n$	type environment	$\boxed{\textbf{mode}}$
$\tau,\sigma ::=$	$\&_{i\in I}\Gamma_i$	branching	$\downarrow$
$\mid$	$\bigoplus_{i\in I}\Gamma_i$	selection	$\uparrow$
$\mid$	$\bigotimes_{i\in I}\Gamma_i$	offer	!
$\mid$	$\biguplus_{i\in I}\Gamma_i$	request	?
$\mid$	$\updownarrow$	closed type	$\updownarrow$

An environment can be thought of as a partial function from names to types. In this view we will talk of *domain* and *range* of an environment. A type environment Γ is *well formed* if any name appears at most once. In the following we consider only well formed environments.

We say a name is *confidential* for a type environment Γ if it appears inside a type in the range of Γ. A name is *public* if it is in the domain of Γ. Intuitively, confidential names are used to identify different occurrences of events that have the same public label. We will see this explicitly when we introduce the event structure semantics.

Branching types represent the notion of "environmental choice": several choices are available for the environment to choose. Selection types represent the notion of "process choice": some choice is made by the process. In both cases the choice is alternative: one excludes all the others. Server types represent the notion of "available resource": I offer to the environment something that is available regardless of whatever else happens. Client types represent the notion of "concurrent request": I want to reserve a resource that I may use at any time.

It is straightforward to define duality between types by exchanging branching and offer, with selection and request, respectively. Therefore, for every type τ and environment Γ, we can define their dual $\overline{\tau}$, $\overline{\Gamma}$. However types and environments enjoy a more general notion of duality that is expressed by the following definition. We define a notion of matching for types. The matching of two types also produces a set of names that are to be considered as "closed", as they have met their dual. Finally, after two types have matched, they produce a "residual" type.

We define the relations $match[\tau,\sigma] \to S$, $match[\Gamma,\Delta] \to S$ symmetric in the first two arguments, and the partial function $res[\tau,\sigma]$ as follows:

- let $\Gamma = x_1 : \sigma_1 \ldots x_n : \sigma_n$ and $\Delta = y_1 : \tau_1 \ldots y_m : \tau_m$. Then $match[\Gamma,\Delta] \to S$ if $n = m$, for every $i \leq n$ $x_i = y_i$, $match[\sigma_i,\tau_i] \to S_i$ and $S = \bigcup_{i\leq n} S_i \cup \{x_i\}$;
- let $\tau = \&_{i\in I}\Gamma_i$ and $\sigma = \bigoplus_{j\in J}\Delta_j$. Then $match[\tau,\sigma] \to S$ if $I = J$, for all $i \in I$, $match[\Gamma_i,\Delta_i] \to S_i$ and $S = \bigcup_{i\in I} S_i$ In such a case $res[\tau,\sigma] = \updownarrow$;

– let $\tau = \bigotimes_{i\in I}\Gamma_i$ and $\sigma = \biguplus_{j\in J}\Gamma_j$ Then $match[\tau,\sigma] \to S$ if $J \subseteq I$, for all $j \in J$, $match[\Gamma_j,\Delta_j] \to S_j$, and $S = \bigcup_{j\in J} S_j$ In such a case $res[\tau,\sigma] = \bigotimes_{i\in I\setminus J}\Gamma_i$.
– $match[\updownarrow,\updownarrow] \to \emptyset$, $res[\updownarrow,\updownarrow] = \updownarrow$.

A branching type matches a corresponding selection types, all their names are closed and the residual type is the special type recording that the matching has taken place. A client type matches a server type if every request corresponds to an available resource. The residual type records which resources are still available.

We now define the composition of two environments. Two environments can be composed if the types of the common names match. Such names are given the residual type by the resulting environment. All the closed names are recorded. Client types can by joined, so that the two environments are allowed to independently reserve some resources. Given two type environments Γ_1,Γ_2 we define the environment $\Gamma_1 \odot \Gamma_2 \stackrel{\text{def}}{=} \Gamma$ and the set of names $cl(\Gamma_1,\Gamma_2)$ as follows:

– if $x \notin Dom(\Gamma_1)$ and no name in $\Gamma_2(x)$ appears in Γ_1, then $\Gamma(x) = \Gamma_2(x)$, $S_x = \emptyset$ and symmetrically;
– if $\Gamma_1(x) = \tau, \Gamma_2(x) = \sigma$ and $match[\tau,\sigma] \to S$, then $\Gamma(x) = res[\tau,\sigma]$ and $S_x = S$;
– if $\Gamma_1(x) = \biguplus_{i\in I}\Delta_i$ and $\Gamma_2(x) = \biguplus_{j\in J}\Delta_j$ and no name appears in both Δ_i and Δ_j for every $i, j \in I \cup J$ we have then $\Gamma(x) = \biguplus_{i\in I\cup J}\Delta_i$ and $S_x = \emptyset$;
– if any of the other cases arises, then Γ is not defined;
– $cl(\Gamma_1,\Gamma_2) = \bigcup_{x\in Dom(\Gamma_1,\Gamma_2)} S_x$.

We can perform injective renaming on environments: if ρ is an injective endofunction on names which leaves $Dom(\Gamma)$ alone, then $\Gamma[\rho]$ is the environment where every name x has been replaced with $\rho(x)$.

3.2 Semantic of types

Type environments are given a semantics in terms of labelled confusion free event structures. Labels have the following form:

$$
\begin{array}{rlll}
\alpha,\beta ::= & x\mathtt{in}_i\langle\tilde y\rangle & \text{branching} & \boldsymbol{\tau} ::= (x,\bar x)\mathtt{in}_i\langle\tilde y\rangle \\
\mid & \bar x\mathtt{in}_i\langle\tilde y\rangle & \text{selection} & \qquad\mid\ (x,\bar x)\mathtt{pr}_i\langle\tilde y\rangle \\
\mid & x\mathtt{pr}_i\langle\tilde y\rangle & \text{offer} & \\
\mid & \bar x\mathtt{pr}_i\langle\tilde y\rangle & \text{request} & \\
\mid & \boldsymbol{\tau} & \text{synchronisation} &
\end{array}
$$

Given a branching $\beta = x\mathtt{in}_j\langle\tilde y\rangle$, we say that x is the *subject* of the label, written $x = subj(\beta)$, the index i is the *branch* (for branching/selection only) and $\tilde y = y_1,\ldots,y_n$ are the *confidential* names written $\tilde y = conf(\beta)$. Similarly for selection, offer and request. The notation "$\mathtt{in}_i$" comes from the injection of the typed λ-calculus. The notation "$\mathtt{pr}_i$" is to suggest duality with the notation for branching. We will use a set of names S also to denote the set of non-synchronisation labels whose subject is in S.

We now define what it means for a label α to be *allowed* by a type environment Γ. Suppose $\Gamma(x) = \sigma$, then:

– if $\alpha = x\mathtt{in}_j\langle\tilde y\rangle$, and if $\sigma = \&_{i\in I}\Gamma_i$ where $\tilde y$ is the domain of Γ_j, then α is allowed;
– if $\alpha = \bar x\mathtt{in}_j\langle\tilde y\rangle$, and if $\sigma = \bigoplus_{i\in I}\Gamma_i$ where $\tilde y$ is the domain of Γ_j then α is allowed;
– if $\alpha = x\mathtt{pr}_j\langle\tilde y\rangle$, and if $\sigma = \bigotimes_{i\in I}\Gamma_i$ where $\tilde y$ is the domain of Γ_j then α is allowed;
– if $\alpha = \bar x\mathtt{pr}_j\langle\tilde y\rangle$, and if $\sigma = \biguplus_{i\in I}\Gamma_i$ where $\tilde y$ is the domain of Γ_j then α is allowed;
– if $\alpha = \tau$, then α is allowed.

Finally, α is allowed by Γ if α is allowed by any of the environments appearing in the types in the range of Γ. Note that if a label is allowed, the definition of well-formedness guarantees that it is allowed in a unique way. Note also that if a label α has subject x and x does not appear in Γ, then α is not allowed by Γ. Let $Dis(\Gamma)$ be the set of labels that are *not allowed* by the environment Γ.

$$[\![y_1:\sigma_1,\ldots,y_n:\sigma_n]\!] = [\![y_1:\sigma_1]\!] \| \ldots \| [\![y_n:\sigma_n]\!]$$

$$[\![x:\&_{i\in I}\Gamma_i]\!] = \sum_{i\in I} x\mathtt{in}_i\langle\tilde{y}_i\rangle.[\![\Gamma_i]\!] \quad [\![x:\bigoplus_{i\in I}\Gamma_i]\!] = \sum_{i\in I} \overline{x}\mathtt{in}_i\langle\tilde{y}_i\rangle.[\![\Gamma_i]\!]$$

$$[\![x:\bigotimes_{i\in I}\Gamma_i]\!] = \prod_{i\in I} x\mathtt{pr}_i\langle\tilde{y}_i\rangle.[\![\Gamma_i]\!] \quad [\![x:\biguplus_{i\in I}\Gamma_i]\!] = \prod_{i\in I} \overline{x}\mathtt{pr}_i\langle\tilde{y}_i\rangle.[\![\Gamma_i]\!] \; [\![x:\updownarrow]\!] = \emptyset$$

Fig. 4. Denotational semantics of types

To define the parallel composition, we use the following symmetric synchronisation algebra: $\alpha \bullet * = \alpha$, $x\mathtt{in}_i\langle\tilde{y}_i\rangle \bullet \overline{x}\mathtt{in}_i\langle\tilde{y}_i\rangle = (x,\overline{x})\mathtt{in}_i\langle\tilde{y}_i\rangle$, $x\mathtt{pr}_i\langle\tilde{y}\rangle \bullet \overline{x}\mathtt{pr}_i\langle\tilde{y}\rangle = (x,\overline{x})\mathtt{pr}_i\langle\tilde{y}\rangle$, and undefined otherwise. The semantics of an environment is the parallel composition of the semantics of the types, with initial events labelled using the corresponding names. The parallel composition is also used to give semantics to client and server types. Such parallel compositions do not involve synchronisation due to the condition on uniqueness of names and thus, as we already explained, they can be thought of as disjoint unions.

The semantics of selection and branching is obtained using the sum of event structures. The semantics is presented in Figure 4, where we assume that $\tilde{y}_i$ represents the sequence of names in the domain of Γ_i. A name used for branching/selection identifies a cell. A name used for offer/request identifies a "cluster" of parallel events.

The following result is a sanity check for our definitions. It shows that matching of types corresponds to parallel composition with synchronisation.

Lemma 3.1. *Take two environments Γ_1 and Γ_2 and suppose $\Gamma_1 \odot \Gamma_2$ is defined. Then $cl(\Gamma_1,\Gamma_2) \subseteq Dis(\Gamma_1 \odot \Gamma_2)$.*

Proposition 3.2. *Take two environments Γ_1, Γ_2, and suppose $\Gamma_1 \odot \Gamma_2$ is defined. Then $([\![\Gamma_1]\!] \| [\![\Gamma_2]\!]) \setminus (Dis(\Gamma_1 \odot \Gamma_2) \cup \boldsymbol{\tau}) = [\![\Gamma_1 \odot \Gamma_2]\!]$.*

In particular, we have that for every environment Γ, $\Gamma \odot \overline{\Gamma}$ is defined and $[\![\Gamma \odot \overline{\Gamma}]\!] = \emptyset$.

3.3 Typing event structures

Given a labelled confusion free event structure $\mathscr{E}$ on the same set of labels as above, we define when $\mathscr{E}$ is typed in the environment Γ, written as $\mathscr{E} \rhd \Gamma$. A type environment Γ defines a general behavioural pattern via its semantics $[\![\Gamma]\!]$. The intuition is that for an event structure $\mathscr{E}$ to have type Γ, $\mathscr{E}$ should follow the pattern of $[\![\Gamma]\!]$, possibly "refining" the causal structure of $[\![\Gamma]\!]$ and possibly omitting some of its actions. We will strengthen the notion of typing later, when standard typing rules allow a finer tuning on the notion of typing.

Definition 3.3. *We say that $\mathscr{E} \rhd \Gamma$, if the following conditions are satisfied:*

- *each cell in $\mathscr{E}$ is labelled by x, $\overline{x}$ or $(x,\overline{x})$, and labels of the events correspond to the label of their cell in the obvious way;*
- *there exists a label-preserving morphism of labelled event structures $f : \mathscr{E} \to [\![\Gamma]\!]$ such that $f(e)$ is undefined if and only if $\lambda(e) \in \boldsymbol{\tau}$.*

Roughly speaking a confusion free event structure $\mathscr{E}$ has type Γ if cells are partitioned in branching, selection, request, offer and synchronisation cells, all the non-synchronisation events of $\mathscr{E}$ are represented in Γ and causality in $\mathscr{E}$ refines causality in $[\![\Gamma]\!]$.

As we said, the parallel composition of confusion free event structures is not confusion free in general. The main result of this section shows that the parallel composition of typed event structures is still confusion free, and moreover is typed.

Lemma 3.4. *Suppose $\mathscr{E} \rhd \Gamma$, and let $e, e' \in E$ be distinct events.*

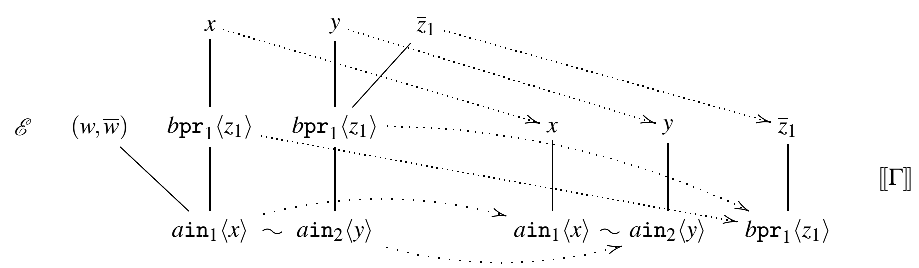

Fig. 5. Typed event structure

- *If $\lambda(e) = \lambda(e') \neq \boldsymbol{\tau}$, then $e \smile e'$.*
- *If $\lambda(e), \lambda(e') \neq \boldsymbol{\tau}$ and $\lambda(e)$ and $\lambda(e')$ have the same subject and different branch,then $e \smile e'$.*
- *If $e \smile_\mu e'$, then $\lambda(e)$ and $\lambda(e')$ have the same subject and different branch.*

Theorem 3.5. *Take two labelled confusion free event structures $\mathscr{E}_1, \mathscr{E}_2$. Suppose $\mathscr{E}_1 \triangleright \Gamma_1$ and $\mathscr{E}_2 \triangleright \Gamma_2$. Assume $\Gamma_1 \odot \Gamma_2$ is defined. Then $(\mathscr{E}_1 \| \mathscr{E}_2) \setminus (Dis(\Gamma_1 \odot \Gamma_2))$ is confusion free and*

$$(\mathscr{E}_1 \| \mathscr{E}_2) \setminus (Dis(\Gamma_1 \odot \Gamma_2)) \triangleright \Gamma_1 \odot \Gamma_2 \ .$$

The proof relies on the fact that the typing system, in particular the uniqueness condition on well formed environments, guarantees that no new conflict is introduced through synchronisation. Key role is played by Lemma 3.4, see the Appendix.

A special case is obtained when the selection cells are all singletons. We call an event structure *deterministic* if its selection cells and its $\boldsymbol{\tau}$ cells are singletons. In particular if all events are labelled $\boldsymbol{\tau}$, a deterministic event structure is conflict free.

Theorem 3.6. *Take two labelled deterministic confusion free event structures $\mathscr{E}_1, \mathscr{E}_2$. Suppose $\mathscr{E}_1 \triangleright \Gamma_1$ and $\mathscr{E}_2 \triangleright \Gamma_2$. Suppose $\Gamma_1 \odot \Gamma_2$ is defined. Then $(\mathscr{E}_1 \| \mathscr{E}_2) \setminus Dis(\Gamma_1 \odot \Gamma_2)$ is deterministic.*

3.4 Examples

In the following, when the indexing set of a branching type is a singleton, we use the abbreviation $(\Gamma)^\downarrow$. Similarly, for a singleton selection type we write $(\Gamma)^\uparrow$. Also, when the indexing set of a type is $\{1,2\}$, we write $(\Gamma_1 \& \Gamma_2)$ or $(\Gamma_1 \otimes \Gamma_2)$, etc...

Example 3.7. Consider the types $\tau_1 = (x : ()^\downarrow \ \& \ y : ()^\downarrow), \sigma_1 = \biguplus_{i \in \{1\}} (z_i : \updownarrow), \tau_2 = (x : ()^\uparrow \oplus y : ()^\uparrow), \sigma_2 = z_1 : \updownarrow \otimes z_2 : \updownarrow$. We have $match[\tau_1, \tau_2] \to \emptyset$, with $res[\tau_1, \tau_2] = \updownarrow$; and $match[\sigma_1, \sigma_2] \to \{z_1\}$, with $res[\sigma_1, \sigma_2] = \bigotimes_{i \in \{2\}} (z_i : \updownarrow)$. Thus if we put $\Gamma_1 = a : \tau_1, b : \sigma_1$, and $\Gamma_2 = a : \tau_2, \sigma_2$, we have that $\Gamma_1 \odot \Gamma_2 = a : \updownarrow, b : \bigotimes_{i \in \{2\}} (z_i : \updownarrow)$.

Example 3.8. As an example of typed event structures, consider the environment $\Gamma = a : (x : ()^\downarrow \ \& \ y : ()^\downarrow), b : \biguplus_{i \in \{1\}} (z_i : ()^\uparrow)$. Figure 5 shows an event structure $\mathscr{E}$, such that $\mathscr{E} \triangleright \Gamma$, together with a morphism $\mathscr{E} \to [\![\Gamma]\!]$. Note that the two events in $\mathscr{E}$ labelled with $b\mathtt{pr}_1\langle z_1\rangle$ are mapped to the same event and indeed they are in conflict.

4 Name sharing CCS

4.1 Syntax

We introduce a variant of CCS that will be interpreted using typed event structures. Our language differs from CCS in many technical details, but the only relevant difference is that synchronisation between actions happens only if the actions share the same

$$\overline{a} \bigoplus_{i\in I} \mathtt{in}_i\langle\tilde{y}_i\rangle.P_i \xrightarrow{\overline{a}\mathtt{in}_j\langle\tilde{y}_j\rangle} P_j \qquad a \&_{i\in I} \mathtt{in}_i\langle\tilde{y}_i\rangle.P_i \xrightarrow{a\mathtt{in}_j\langle\tilde{y}_j\rangle} P_j$$

$$\overline{a}\mathtt{pr}_j\langle\tilde{y}\rangle.P \xrightarrow{\overline{a}\mathtt{pr}_j\langle\tilde{y}\rangle} P \qquad a\mathtt{pr}_j\langle\tilde{y}\rangle.P \xrightarrow{a\mathtt{pr}_j\langle\tilde{y}\rangle} P$$

$$\frac{P \xrightarrow{\beta} P' \quad subj(\beta) \notin S}{P \setminus S \xrightarrow{\beta} P' \setminus S} \qquad \frac{P \xrightarrow{\tau} P'}{P \setminus S \xrightarrow{\tau} P' \setminus S}$$

$$\frac{P_n \xrightarrow{\beta} P'}{\prod_{i\in\mathbb{N}} P_i \xrightarrow{\beta} (\prod_{i\in\mathbb{N}\setminus\{n\}} P_i) \mid P'} \qquad \frac{P_n \xrightarrow{\alpha} P' \quad P_m \xrightarrow{\beta} P''}{\prod_{i\in\mathbb{N}} P_i \xrightarrow{\alpha\bullet\beta} (\prod_{i\in\mathbb{N}\setminus\{n,m\}} P_i) \mid P' \mid P''}$$

Fig. 6. Labelled Transition System for Name Sharing CCS

confidential names. Syntactically this looks like name passing, with the difference that processes decide their confidential names before communicating, and there is not α-conversion. If the chosen names do not coincide, the processes do not synchronise. We will formalise the relation between our calculus and the typed π-calculus later.

Another minor difference with standard CCS is that we allow infinite parallel composition and infinite restriction. The former is necessary in order to translate replicated processes of the π-calculus. The standard intuition in the π-calculus is that the process $!P$ represents the parallel composition of infinitely many copies of P. We need to represent this explicitly in order to be able to provide each copy with different confidential names. Infinite restriction is also necessary, because we need to restrict all confidential names that are shared between two processes in parallel, and these are in general infinitely many.

We call this language Name Sharing CCS, or NCCS. The syntax is as follows:

$$\begin{array}{rll} P ::= & a \&_{i\in I} \mathtt{in}_i\langle\tilde{y}_i\rangle.P_i & \text{branching} \\ \mid & \overline{a} \bigoplus_{i\in I} \mathtt{in}_i\langle\tilde{y}_i\rangle.P_i & \text{selection} \\ \mid & a\mathtt{pr}_j\langle\tilde{y}\rangle.P & \text{single offer} \\ \mid & \overline{a}\mathtt{pr}_j\langle\tilde{y}\rangle.P & \text{single request} \\ \mid & \prod_{i\in I} P_i & \text{parallel composition} \\ \mid & P \setminus S & \text{restriction} \\ \mid & \mathbf{0} & \text{zero} \end{array}$$

We sometime denote parallel composition where the indexing set is $\{1,2\}$, by $P_1 \| P_2$. When the indexing set of branching is a singleton, we sometimes write $a\langle\tilde{y}\rangle.P$, and similarly for selection, offer and requests. As before $\tilde{y}$ denotes a finite sequence of distinct names $y_1 \ldots y_n$, whenever the length of the sequence and the identity of the individual names do not matter. We will also sometimes abuse the notation by using set theoretic notions applied to the sequences. So, for instance, $\tilde{y} \cap \tilde{y}' = \emptyset$ means that for all i, j $y_i \neq y'_j$. S denotes a set of names. Finally, processes are identified up to a straightforward structural congruence, which includes the rule $(P \setminus S) \setminus T \equiv P \setminus (S \cup T)$, but no notion of α-equivalence.

A name is *confidential* in P if it appears in a confidential position inside P.

We want to identify different fragments of the language. The fragment where the indexing sets of branching and selection are always singleton is called *simple*. The fragment when the selection is always a singleton, but the branching is arbitrary is called *deterministic*. The general language is for clarity denoted as the *nondeterministic* fragment.

The operational semantics is completely analogous to the one of CCS, and it is shown in Figure 6. Labels are the same as the one we have seen for event structures. Again, pair labels are globally denoted by $\boldsymbol{\tau}$.

As in CCS, prefixes generate the labelled actions. Processes in parallel can proceed independently or synchronise over complementary actions. Restriction inhibits actions over a particular set of names, but not $\boldsymbol{\tau}$. The main difference with CCS is the presence of "confidential names" that are used only for synchronisation. Note also that only the subject of an action is taken into account for restriction.

Example 4.1. For instance the process

$$(a\langle x\rangle.P \,|\, \overline{a}\langle y\rangle.R) \setminus \{a\}$$

cannot perform any transition, because x and y do not match. The process

$$(a\langle x\rangle.P \,|\, \overline{a}\langle x\rangle.Q \,|\, \overline{a}\langle x\rangle.R) \setminus \{a\}$$

can perform two different initial $\boldsymbol{\tau}$ transitions. Since the name x is not bound, it does not become private to the subprocesses involved in the communication. The process

$$(a \,\&_{i\in\{1,2\}}\, \mathtt{in}_i\langle\rangle.P_i \,|\, \overline{a} \bigoplus_{i\in\{1,2\}} \mathtt{in}_i\langle\rangle.R_i) \setminus \{a\}$$

can perform, nondeterministically, two $\boldsymbol{\tau}$ transitions to $(P_1\,|\,R_1)\setminus\{a\}$ or to $(P_2\,|\,R_2)\setminus\{a\}$.

4.2 Typing Rules

Before introducing the typing rules, we have to define the operation of "parallel composition of environments".

If Γ_h $h \in H$ is a family of types such that for every name x, either for every h, $\Gamma_h(x) = \biguplus_{k_h\in K_h} \Delta_{k_h}$, or $x \in Dom(\Gamma_h)$ for at most one h. We define $\Gamma = \prod_{h\in H} \Gamma_h$ as follows. If for every h, $\Gamma_h(x) = \biguplus_{k_h\in K_h} \Delta_{k_h}$, then $\Gamma(x) = \biguplus_{k_h\in K_h, h\in H} \Delta_{k_h}$, assuming all the names involved are distinct. If $x \in Dom(\Gamma_h)$ for at most one h, then $\Gamma(x) = \Gamma_h(x)$.

A special case is when all the Γ_h are different instances of the same environment, up to renaming of the confidential names. For any set K, let $F_K : \mathit{Names} \to \mathscr{P}(\mathit{Names})$ be a function such that, for every name x, there is a bijection between K and $F_K(x)$. Concretely we can represent $F_K(x) = \{x^k \mid k \in K\}$. In the following we assume that each set K is associated to a unique F_K, and that for distinct x, y, $F_K(x) \cap F_K(y) = \emptyset$.

Given a type τ, and an index k, define τ^k as follows:

- $\bigotimes_{h\in H}(\tilde{y}_h : \tilde{\tau}_h)^k = \bigotimes_{h\in H}(\tilde{y}^k_h : \tilde{\tau}^k_h)$, where $\tilde{y}_h = (y_{i,h})_{i\in I}$ and $\tilde{y}^k_h = (y^k_{i,h})_{i\in I}$;
- and similarly for all other types.

Given an environment Γ, we define Γ^k were for every name $x \in Dom(\Gamma)$, $\Gamma^k(x) = \Gamma(x)^k$. The environment $\Gamma[K]$ is defined as $\prod_{k\in K} \Gamma^k$, and is thus defined only when for every $x \in Dom(\Gamma)$, $MD(\Gamma(x)) = ?$. We will also assume that all names in the range of the substitution are fresh, in the sense that no name in the range of F_K appears in the domain of Γ. Under this assumption we easily have that if Γ is well formed and if $\Gamma[K]$ is defined, then $\Gamma[K]$ is also well formed.

We are now ready to write the rules. The rule for weakening of the client type tells us that we can request a resource even if we are not actually using it. The rule for the selection tells us that we can choose less than what the types offers. The parallel composition is well typed only if the names used for communication have matching types, and if the matched names are restricted. This makes sure that communication can happen, and that the shared names are indeed private to the processes involved. The rules are shown in Figure 7.

$$\frac{}{0 \triangleright \emptyset}\ \text{Zero} \quad \frac{P \triangleright \Gamma \quad a \notin \Gamma}{P \triangleright \Gamma, a : \biguplus_{h \in H} \Gamma_h}\ \text{WeakReq} \quad \frac{P \triangleright \Gamma \quad a \notin \Gamma}{P \triangleright \Gamma, a : \updownarrow}\ \text{WeakCl}$$

$$\frac{P \triangleright \Gamma, a : \tau \quad MD(\tau) = !, \updownarrow}{P \setminus a \triangleright \Gamma}\ \text{Res}$$

$$\frac{P_i \triangleright \Gamma, \tilde{y}_i : \tilde{\tau}_i \quad a \notin \Gamma}{a \mathop{\&}_{i \in I} \mathtt{in}_i \langle \tilde{y}_i \rangle . P_i \triangleright \Gamma, a : \mathop{\&}_{i \in I} (\tilde{y}_i : \tilde{\tau}_i)}\ \text{Branch}$$

$$\frac{P_i \triangleright \Gamma, \tilde{y}_i : \tilde{\tau}_i \quad a \notin \Gamma \quad I \subseteq J}{\overline{a} \bigoplus_{i \in I} \mathtt{in}_i p_i \langle \tilde{y}_i \rangle . P_i \triangleright \Gamma, a : \bigoplus_{i \in J} (\tilde{y}_i : \tilde{\tau}_i)}\ \text{Sel}$$

$$\frac{P \triangleright \Gamma, \tilde{w}_j : \tilde{\tau}_j, a : \biguplus_{h \in H} (\tilde{w}_h : \tilde{\tau}_h) \quad \tilde{w}_j \text{ fresh}}{\overline{a}\mathtt{pr}_j \langle \tilde{w}_j \rangle . P \triangleright \Gamma, a : \biguplus_{h \in H \uplus \{j\}} (\tilde{w}_h : \tilde{\tau}_h)}\ \text{Req}$$

$$\frac{P_h \triangleright \Gamma_h, \tilde{y}_h : \tilde{\tau}_h \quad a \notin \Gamma}{\prod_{h \in H} a\mathtt{pr}_h \langle \tilde{y}_h \rangle . P_h \triangleright \prod_{h \in H} \Gamma_h, a : \bigotimes_{h \in H} (\tilde{y}_h : \tilde{\tau}_h)}\ \text{Offer}$$

$$\frac{P_i \triangleright \Gamma_i \quad (i = 1, 2) \quad S = cl(\Gamma_1, \Gamma_2)}{(P_1 \| P_2) \setminus S \triangleright \Gamma_1 \odot \Gamma_2}\ \text{Par}$$

Fig. 7. Typing Rules for NCCS

4.3 Typed Semantics

The relation Γ allows β was defined in Section 3. We also need a definition of the environment $\Gamma \setminus \beta$, which is as follows.

- $\Gamma \setminus \boldsymbol{\tau} = \Gamma$;
- if $\Gamma = \Delta, x : \mathop{\&}_{i \in I} (\tilde{y}_i : \tilde{\tau}_i)$, then $\Gamma \setminus x\mathtt{in}_i(\tilde{y}_i) = \Delta, \tilde{y}_i : \tilde{\tau}_i$;
- if $\Gamma = \Delta, x : \bigoplus_{i \in I} (\tilde{y}_i : \tilde{\tau}_i)$, then $\Gamma \setminus \overline{x}\mathtt{in}_i(\tilde{y}_i) = \Delta, \tilde{y}_i : \tilde{\tau}_i$;
- if $\Gamma = \Delta, x : \bigotimes_{h \in H \uplus \{j\}} (\tilde{y}_h : \tilde{\tau}_h)$,
  then $\Gamma \setminus x\mathtt{pr}_j(\tilde{y}_j) = \Delta, \tilde{y}_j : \tilde{\tau}_j, \bigotimes_{h \in H} (\tilde{y}_h : \tilde{\tau}_h)$;
- if $\Gamma = \Delta, x : \biguplus_{h \in H \uplus \{j\}} (\tilde{y}_h : \tilde{\tau}_h)$,
  then $\Gamma \setminus \overline{x}\mathtt{pr}_j(\tilde{y}_j) = \Delta, \tilde{y}_j : \tilde{\tau}_j, \biguplus_{h \in H} (\tilde{y}_h : \tilde{\tau}_h)$.

Note that $\Gamma \setminus \beta$ is defined precisely when Γ allows β. We have the following

Proposition 4.2. *If $P \triangleright \Gamma$, $P \xrightarrow{\beta} Q$ and Γ allows β, then $Q \triangleright \Gamma \setminus \beta$.*

Corollary 4.3 (Subject Reduction). *If $P \triangleright \Gamma$, $P \xrightarrow{\tau} Q$ then $Q \triangleright \Gamma$.*

Proposition 4.2 allows us to define the notion of *typed transition*, written $P \triangleright \Gamma \xrightarrow{\beta} Q \triangleright \Gamma'$ by adding the constraint:

$$\frac{P \xrightarrow{\beta} Q \quad \Gamma \text{ allows } \beta}{P \triangleright \Gamma \xrightarrow{\beta} Q \triangleright \Gamma \setminus \beta}$$

The relations of bisimulation and bisimilarity between typed processes are defined as usual, using typed transitions. Bisimilarity is denoted as $P \triangleright \Gamma \approx Q \triangleright \Gamma$.

5 Event structure semantics of Name Sharing CCS

5.1 Semantics of nondeterministic NCCS

The semantics of a typed NCCS process is given in terms of labelled event structures, using the operations, in particular the parallel composition, as defined in Section 2.5. This construction is perfectly analogous to the one in [39], the only difference

$$
\begin{aligned}
[\![0 \triangleright \emptyset]\!] &= \emptyset \\
[\![P \triangleright \Gamma, x : \biguplus_{h \in H} \Gamma_h]\!] &= [\![P \triangleright \Gamma]\!] \\
[\![P \triangleright \Gamma, x : \updownarrow]\!] &= [\![P \triangleright \Gamma]\!] \\
[\![P \setminus a \triangleright \Gamma]\!] &= [\![P \triangleright \Gamma, a : \tau]\!] \setminus \{a\} \\
[\![\overline{a} \bigoplus_{i \in I} \mathtt{in}_i \langle \tilde{y}_i \rangle . P_i \triangleright \Gamma, a : \bigoplus_{i \in I} (\tilde{y}_i : \tilde{\tau}_i)]\!] &= \textstyle\sum_{i \in I} \overline{a} \mathtt{in}_i \langle \tilde{y}_i \rangle . [\![P_i \triangleright \Gamma, \tilde{y}_i : \tilde{\tau}_i]\!] \\
[\![a \, \&_{i \in I} \mathtt{in}_i \langle \tilde{y}_i \rangle . P_i \triangleright \Gamma, a : \&_{i \in I} (\tilde{y}_i : \tilde{\tau}_i)]\!] &= \textstyle\sum_{i \in I} a \mathtt{in}_i \langle \tilde{y}_i \rangle . [\![P_i \triangleright \Gamma, \tilde{y}_i : \tilde{\tau}_i]\!] \\
[\![\overline{a} \mathtt{pr}_j \langle \tilde{y} \rangle . P \triangleright \Gamma, a : \biguplus_{k \in K \uplus \{j\}} (\tilde{y}_k : \tilde{\tau}_k)]\!] &= \overline{a} \mathtt{pr}_j \langle \tilde{y} \rangle . [\![P \triangleright \Gamma, a : \biguplus_{k \in K} (\tilde{y}_k : \tilde{\tau}_k), \tilde{y}_j : \tilde{\tau}_j]\!] \\
[\![\textstyle\prod_{k \in K} a \mathtt{pr}_k \langle \tilde{y}_k \rangle P_k \triangleright \prod_{k \in K} \Gamma_k, a : \bigotimes_{k \in K} (\tilde{y}_k : \tilde{\tau}_k)]\!] &= \textstyle\prod_{k \in K} a \mathtt{pr}_k \langle \tilde{y}_k \rangle . [\![P_k \triangleright \Gamma_k, \tilde{y}_k : \tilde{\tau}_k]\!] \\
[\![(P_1 \| P_2) \setminus S \triangleright \Gamma_1 \odot \Gamma_2]\!] &= [\![P_1 \triangleright \Gamma_1]\!] \mid [\![P_2 \triangleright \Gamma_2]\!] \setminus (Dis(\Gamma_1 \odot \Gamma_2))
\end{aligned}
$$

Fig. 8. Denotational semantics of simple Name Sharing CCS

being the synchronisation algebra. However, since the synchronisation algebra is the same for both the operational and the denotational semantics, we obtain automatically the correspondence between the two, as in [39].

The semantic is presented in Figure 8.

In the parallel composition, we have to restrict all the channels that are subject of communication. More generally, we need to restrict all the actions that are not allowed by the new type environment.

The main property of the typed semantics is that all denoted event structures are confusion free. More generally the semantics of a typed process is a typed event structure

Theorem 5.1. *Let P be a process and Γ an environment such that $P \triangleright \Gamma$. Then*

- $[\![P \triangleright \Gamma]\!]$ *is confusion free;*
- $[\![P \triangleright \Gamma]\!] \triangleright [\![\Gamma]\!]$.

5.2 Semantics of deterministic NCCS

The syntax of NCCS introduces the conflict explicitly, therefore we cannot obtain conflict free event structures. The result above shows that no new conflict is introduced through synchronisation. Moreover, in the deterministic fragment, synchronisation does indeed resolve the conflicts.

First it is easy to show that the semantics of deterministic NCCS is in term of deterministic event structures:

Proposition 5.2. *Suppose P is a deterministic process, and that $P \triangleright \Gamma$. Then $[\![P \triangleright \Gamma]\!]$ is deterministic.*

The main theorem is the following, which justifies the term "deterministic". It states that once all choices have been matched with selections, or cancelled out, what remains is a conflict free event structure.

Theorem 5.3. *If let X be the set of names in P, then $[\![P \triangleright \Gamma]\!] \setminus X$ is a conflict free event structure*

Corollary 5.4. *If $[\![\Gamma]\!] = \emptyset$, then $[\![P \triangleright \Gamma]\!]$ is conflict free.*

5.3 Semantics of simple NCCS

Although the syntax of NCCS does not introduce directly any conflict, there is in principle the possibility that conflict is introduced by the parallel composition. The typing system is designed in such a way that this is not the case.

Theorem 5.5. *Suppose P is a simple process such that $P \triangleright \Gamma$. Then $[\![P \triangleright \Gamma]\!]$ is conflict free.*

5.4 Correspondence between the semantics

In order to show the correspondence between the operational and the denotational semantics, we invoke Winskel and Nielsen's handbook chapter [39]. Note that our semantics are a straightforward modification of the standard CCS semantics. This is the main reason why we chose the formalism presented in this paper: we wanted to depart as little as possible from the treatment of [39].

The main difference is that typed semantics modifies the behaviour, by forbidding some of the actions. However this modification acts precisely as a special form of name restriction: in the labelled transition system it blocks some action, while in event structure it cancel them out (together with all events enabled by them). With a straightforward generalisation of the notion of restriction, we then preserve the correspondence between the two semantics and the technique of [39] carries over.

A denotational model which is very close to the operational semantics is that of *synchronisation trees*. Synchronisation trees are just labelled transition system structured as trees, with an initial state as the root. They form a category in a straightforward way, and they support constructions similar to the one used for event structures (prefixing, hiding, product). Using these constructions, we can define a new semantics of NCCS in terms of synchronisation trees $[\![P]\!]_T$.

There is a functor F between the category of labelled event structures and the category of synchronisation trees that unfolds event structures into trees. F is right adjoint of the functor that sees trees as event structures (where every two events are causally related or in conflict). It can be shown that F commutes with the semantics, that is $F[\![P]\!] = [\![P]\!]_T$, the key point being that F is right adjoint and therefore preserves products. Then, it can be shown that the synchronisation tree $[\![P]\!]_T$ is bisimilar to the operational semantics described in Section 4. The proof of this is quite technical, and can be found in [39]. Recall that the typing restrict the behaviour in the same way as the hiding.

Using this correspondence it is easy to prove that the semantics in terms of event structures is sound with respect to bisimilarity.

Theorem 5.6. *Take two typed NCCS processes $P \triangleright \Gamma, Q \triangleright \Gamma$. Suppose that $[\![P \triangleright \Gamma]\!] = [\![Q \triangleright \Gamma]\!]$, then $P \triangleright \Gamma \approx Q \triangleright \Gamma$.*

Proof. If $[\![P \triangleright \Gamma]\!] = [\![Q \triangleright \Gamma]\!]$, then $[\![P \triangleright \Gamma]\!]_T = [\![Q \triangleright \Gamma]\!]_T$, and thus $P \triangleright \Gamma \approx Q \triangleright \Gamma$. ☐

This theorem is the best result we can get: indeed, as for standard CCS, we cannot expect the event structure semantics to be fully abstract. Bisimilarity is a “interleaving” semantics, equating the two processes $\overline{a} \| \overline{a}$ and $\overline{a}.\overline{a}$, which have different event structure semantics.

A more direct correspondence is described in the following.

Definition 5.7. *Let $\mathscr{E} = \langle E, \leq, \smile, \lambda \rangle$ be a labelled event structure and let e be one of its minimal events. The event structure $\mathscr{E}\lfloor e = \langle E', \leq', \smile', \lambda' \rangle$ is defined as follows: $E' = \{e' \in E \mid e' \not\smile e\}$, $\leq' = \leq_{|E'}$, $\smile' = \smile_{|E'}$, and $\lambda' = \lambda_{E'}$.*

Roughly speaking, $\mathscr{E}\lfloor e$ is $\mathscr{E}$ minus the event e, and minus all events that are in conflict with e. We can then generate a labelled transition system as follows: if $\lambda(e) = \beta$, then

$$\mathscr{E} \xrightarrow{\beta} \mathscr{E}\lfloor e \ .$$

We can therefore state the following correspondence:

Theorem 5.8. *Let $\cong$ denote isomorphism of labelled event structures;*

- *if $P \triangleright \Gamma \xrightarrow{\beta} P' \triangleright \Gamma \setminus \beta$, then $[\![P \triangleright \Gamma]\!] \xrightarrow{\beta} \cong [\![P' \triangleright \Gamma \setminus \beta]\!]$.*
- *if $[\![P \triangleright \Gamma]\!] \xrightarrow{\beta} \mathscr{E}'$ then $P \triangleright \Gamma \xrightarrow{\beta} P' \triangleright \Gamma \setminus \beta$ and $\mathscr{E}' \cong [\![P' \triangleright \Gamma \setminus \beta]\!]$.*

The proof can be found in the appendix. It is by induction on the operational rules. The only difficult case is the parallel composition.

6 A linear version of the π-calculus

This section briefly summarises an extension of linear version of the π-calculus in [3] to non-determinism [40]. Although this summary is technically self-contained, the reader may refer to [3, 40] for detailed illustration and more examples.

6.1 Syntax and reduction

The following gives the reduction rule of the standard π-calculus:

$$x(\tilde{y}).P \,|\, \overline{x}\langle\tilde{v}\rangle.Q \;\longrightarrow\; P\{\tilde{v}/\tilde{y}\} \,|\, Q$$

Operationally, this reduction represents the consumption of a message by a receptor. As anticipated, we consider a restricted version of the π-calculus, where only bound names are passed in interaction. Besides producing a simpler and more elegant theory, this restriction allows tighter control of sharing and aliasing without losing essential expressiveness, making it easier to administer name usage in more stringent ways. The resulting calculus is called the πI-calculus in the literature [31] and has the same expressive power as the version with free name passing (as proved Section 6 in [41]). Syntactically we restrict an output to the form $(\nu\,\tilde{y})\overline{x}\langle\tilde{y}\rangle.P$ (where names in $\tilde{y}$ are pairwise distinct), which we henceforth write $\overline{x}(\tilde{y}).P$. For dynamics, we have the following forms of reduction by the restriction $\longrightarrow$ to the bound output.

$$x(\tilde{y}).P \,|\, \overline{x}(\tilde{y}).Q \longrightarrow (\nu\,\tilde{y})(P \,|\, Q)$$
$$!x(\tilde{y}).P \,|\, \overline{x}(\tilde{y}).Q \longrightarrow\, !x(\tilde{y}).P \,|\, (\nu\,\tilde{y})(P \,|\, Q)$$

After communication, $\tilde{y}$ are shared between P and Q. Our framework is applicable to more general nondeterministic version of the calculus, where input and output can be non-deterministic branching and selection. Branching is similar to the "case" construct and selection is "injection" in the typed λ-calculi; these constructs have been studied in other typed π-calculi [33]. The branching variant of the reduction becomes:

$$x\,\&_{i\in I}\,\mathtt{in}_i(\tilde{y}_i).P_i \,|\, \overline{x}\bigoplus_{j\in J}\mathtt{in}_j(\tilde{y}_j).Q_j \,|\; \longrightarrow\; (\nu\,\tilde{y}_h)(P_h \,|\, Q_h)$$

where we assume $h \in J \cap I$, with I, J denoting finite or countably infinite indexing sets.

The formal grammar of the calculus is defined below.

$$P \;::=\; x\,\&_{i\in I}\,\mathtt{in}_i(\tilde{y}_i).P_i \;\;|\;\; \overline{x}\bigoplus_{i\in I}\mathtt{in}_i(\tilde{y}_i).P_i \;\;|\;\; P\,|\,Q \;\;|\;\; (\nu\,x)P \;\;|\;\; \mathbf{0} \;\;|\;\; !x(\tilde{y}).P$$

$x\,\&_{i\in I}\,\mathtt{in}_i(\tilde{y}_i).P_i$ (resp. $\overline{x}\bigoplus_{i\in I}\mathtt{in}_i(\tilde{y}_i).P_i$) is a branching input (resp. selecting output). $P\,|\,Q$ is a parallel composition, $(\nu\,x)P$ is a restriction and $!x(\tilde{y}).P$ is a replicated input. We omit the empty vector: for example, $\overline{a}$ stands for $\overline{a}()$. When the index in the branching or selection indexing set is a singleton we use the notation $x(\tilde{y}).P$ or $\overline{x}(\tilde{y}).P$; when it is binary, we use $x((\tilde{y_1}).P_1 \& (\tilde{y_2}).P_2)$ or $\overline{x}((\tilde{y_1}).P_1 \oplus (\tilde{y_2}).P_2)$. The bound/free names are defined as usual. We assume that names in a vector $\tilde{y}$ are pairwise distinct. We use $\equiv_\alpha$ and $\equiv$ for the standard α and structured equivalences [26, 3, 41, 17].

We can identify important fragments of the calculus. Processes where all selection indexing sets are singletons are called *deterministic*. Deterministic processes where also branching indexing sets are singletons are called *simple*.

6.2 Types and typings

This subsection reviews the basic idea of the linear type discipline in [3]. We can easily extend the result in this paper to other family of the linear calculi studied in [3, 41, 17]. The linear type discipline restricts the behaviour of processes as follows.

(A) for each linear name there are a unique input and a unique output; and
(B) for each replicated name there is a unique stateless replicated input with zero or more dual outputs.

In the context of deterministic processes, the typing system guarantees *confluence*. We will see that in the presence of nondeterminism this typing system guarantees *confusion freeness*.

Example 6.1. As an example for the first condition, let us consider:

$$Q_1 \stackrel{\text{def}}{=} \overline{a}.b \,|\, \overline{a}.c \,|\, a \qquad\qquad Q_2 \stackrel{\text{def}}{=} b.\overline{a} \,|\, c.\overline{b} \,|\, a.(\overline{c} \,|\, \overline{e})$$

Then Q_1 is not typable as a appears twice as output, while Q_2 is typable since each channel appears at most once as input and output. Typability of simple processes such as Q_2 offers only deterministic behaviour. However branching and selection can provide non-deterministic behaviour, preserving linearity:

$$Q_3 \stackrel{\text{def}}{=} \overline{a}.(b \oplus c) \,|\, a.(\overline{d} \,\&\, \overline{e})$$

Q_3 is typable, and we have either $Q_3 \longrightarrow (b \,|\, \overline{d})$ or $Q_3 \longrightarrow (c \,|\, \overline{e})$. As an example of the second constraint, let us consider the following two processes:

$$Q_4 \stackrel{\text{def}}{=} !\,b.\overline{a} \,|\, !\,b.\overline{c} \qquad\qquad Q_5 \stackrel{\text{def}}{=} !\,b.\overline{a} \,|\, \overline{b} \,|\, !\,c.\overline{b}$$

Q_4 is untypable because b is associated with two replicators: but Q_5 is typable since, while output at b appears twice, a replicated input at b appears only once.

Types Channel types are inductively made up from type variables and action modes. The four *action modes* $\downarrow, \uparrow, !, ?$ where introduced in Section 3. *Input modes* are $\downarrow, !$, while $\uparrow, ?$ are *output modes*. We let $p, p', \ldots$ denote modes. We define $\overline{p}$, the *dual* of p, by: $\overline{\downarrow} = \uparrow$, $\overline{!} = ?$ and $\overline{\overline{p}} = p$. Then the syntax of types are given as follows:

$$\begin{array}{rll}
\sigma ::= & \&_{i\in I}\,(\tilde{\sigma}_i)^{\downarrow} & \text{branching} \\
| & \bigoplus_{i\in I}\,(\tilde{\sigma}_i)^{\uparrow} & \text{selection} \\
| & (\tilde{\sigma})^{!} & \text{offer} \\
| & (\tilde{\sigma})^{?} & \text{request} \\
\tau ::= & \sigma \;\mid\; \updownarrow &
\end{array}$$

where $\tilde{\sigma}$ is a vector of types. We write $MD(\tau)$ for the outermost mode of τ. The *dual of* τ, written $\overline{\tau}$, is the result of dualising all action modes, with $\updownarrow$ being self-dual. A type environment Γ is a finite mapping from channels to channel types. Sometimes we will write $x \in \Gamma$ to mean $x \in Dom(\Gamma)$.

Types restrict the composability of processes: for example, for parallel composition, if P is typed under environment Γ_1, Q is under Γ_2 and $\Gamma_1 \odot \Gamma_2$ is defined for a partial operator $\odot$ with the resulting Γ, then we assign Γ to $P \,|\, Q$. If $\Gamma_1 \odot \Gamma_2$ is not defined, the composition is not allowed. Formally, $\odot$ is the partial commutative operation on Γ_1 and Γ_2 where $\Gamma_1 \odot \Gamma_2 \stackrel{\text{def}}{=} \Gamma$ is defined as follows:

(1)
- if $\Gamma_1(x) = \&_{i\in I}\,(\tilde{\tau}_i)^{\downarrow}$ and $\Gamma_2(x) = \bigoplus_{i\in I}\,(\tilde{\overline{\tau}}_i)^{\uparrow}$ then $\Gamma(x) = \updownarrow$, and symmetrically;
- if $\Gamma_1(x) = \&_{i\in I}\,(\tilde{\tau}_i)^{\downarrow}$ and $x \notin Dom(\Gamma_2)$ then $\Gamma(x) = \&_{i\in I}\,(\tilde{\tau}_i)^{\downarrow}$, and symmetrically;
- if $\Gamma_1(x) = \bigoplus_{i\in I}\,(\tilde{\tau}_i)^{\uparrow}$ and $x \notin Dom(\Gamma_2)$ then $\Gamma(x) = \bigoplus_{i\in I}\,(\tilde{\tau}_i)^{\uparrow}$, and symmetrically;

(2)
- if $\Gamma_1(x) = (\tilde{\tau})^{!}$ and $\Gamma_2(x) = (\tilde{\overline{\tau}})^{?}$ then $\Gamma(x) = (\tilde{\tau})^{!}$, and symmetrically;
- if $\Gamma_1(x) = (\tilde{\tau})^{?}$ and $\Gamma_2(x) = (\tilde{\tau})^{?}$ then $\Gamma(x) = (\tilde{\tau})^{?}$.

(3) undefined in any other cases (if any of the other cases arises, then the whole $\Gamma_1 \odot \Gamma_2$ is not defined).

Intuitively, the rules in (2) say that a server should be unique, but an arbitrary number of clients can request interactions. The rules in (1) say that once we compose input-output linear channels, the channel becomes uncomposable. Note that (3) says other compositions are undefined. (1) and (2) ensure the two constraints (A) and (B) in § 6.2, respectively.

$$\frac{}{\mathbf{0} \triangleright \emptyset}\ \text{Zero} \qquad \frac{P_i \triangleright \Gamma_i \quad (i=1,2)}{P_1 \mid P_2 \triangleright \Gamma_1 \odot \Gamma_2}\ \text{Par} \qquad \frac{P \triangleright \Gamma, a:\tau \quad a \notin \Gamma \quad MD(\tau) = !, \updownarrow}{(\nu a) P \triangleright \Gamma}\ \text{Res}$$

$$\frac{P \triangleright \Gamma \quad x \notin \Gamma}{P \triangleright \Gamma, x : (\tilde{\tau})^{?}}\ \text{WeakOut} \qquad \frac{P \triangleright \Gamma \quad x \notin \Gamma}{P \triangleright \Gamma, x : \updownarrow}\ \text{WeakCl}$$

$$\frac{P_i \triangleright \Gamma, \tilde{y}_i : \tilde{\tau}_i \quad a \notin \Gamma}{a \&_{i \in I} \mathtt{in}_i(\tilde{y}_i).P_i \triangleright \Gamma, a : \&_{i \in I} (\tilde{\tau}_i)^{\downarrow}}\ \text{LIn} \qquad \frac{P_i \triangleright \Gamma, \tilde{y}_i : \tilde{\tau}_i \quad a \notin \Gamma \quad I \subseteq J}{\overline{a} \bigoplus_{i \in I} \mathtt{in}_i(\tilde{y}_i).P_i \triangleright \Gamma, a : \bigoplus_{i \in J} (\tilde{\tau}_i)^{\uparrow}}\ \text{LOut}$$

$$\frac{P \triangleright \Gamma, \tilde{y} : \tilde{\tau} \quad a \notin \Gamma \quad \forall (x{:}\tau) \in \Gamma.\, MD(\tau) = ?}{!a(\tilde{y}).P \triangleright \Gamma, a : (\tilde{\tau})^{!}}\ \text{RIn} \qquad \frac{P \triangleright \Gamma, a : (\tilde{\tau})^{?}, \tilde{y} : \tilde{\tau}}{\overline{a}(\tilde{y}).P \triangleright \Gamma, a : (\tilde{\tau})^{?}}\ \text{ROut}$$

Fig. 9. Linear Typing Rules

Typing system is defined in Figure 9. These are identical to the affine π-calculus [3] except the non-deterministic linear output rule. The (Zero) rule types $\mathbf{0}$. As $\mathbf{0}$ has no free names, it is not being given any channel types. In (Par), $\Gamma_1 \odot \Gamma_2$ guarantees the consistent channel usage like linear inputs being only composed with linear outputs, etc. In (Res), we do not allow $\uparrow$, ? or $\downarrow$-channels to be restricted since they carry actions which expect their dual actions to exist in the environment. (WeakOut) and (WeakCl) weaken with ?-names or $\updownarrow$-names, respectively, since these modes do not require *further* interaction. (LIn) ensures that x occurs precisely once. (LOut) is dual. (RIn) is the same as (LIn) except that no free linear channels are suppressed. This is because a linear channel under replication could be used more than once. (ROut) is similar with (LOut). Note we need to apply (WeakOut) before the first application of (ROut).

6.3 A typed labelled transition relation

Typed transitions describe the observations a typed observer can make of a typed process. The typed transition relation is a proper subset of the untyped transition relation, while not restricting $\boldsymbol{\tau}$-actions: hence typed transitions restrict observability, not computation. Let the set of *labels* $\alpha, \beta, \dots$ be the one defined in Section 3. For a label β we denote its subject as $subj(\beta)$ and its names as $conf(\beta)$; the operation $\alpha \bullet \beta$ was introduced in § 3.2.

The standard untyped transition relation is defined in Figure 10. We define the predicate "Γ allows β" which represents how an environment restricts observability;

for all Γ, Γ allows $\boldsymbol{\tau}$;
if $MD(\Gamma(x)) = \downarrow$, then Γ allows $x\mathtt{in}_i\langle\tilde{y}\rangle$;
if $MD(\Gamma(x)) = \uparrow$, then Γ allows $\overline{x}\mathtt{in}_i\langle\tilde{y}\rangle$;
if $MD(\Gamma(x)) = !$, then Γ allows $x\mathtt{pr}_i\langle\tilde{y}\rangle$;
if $MD(\Gamma(x)) = ?$, then Γ allows $\overline{x}\mathtt{pr}_i\langle\tilde{y}\rangle$.

Intuitively, labels only allowed when the type environment is coherent with them.

Whenever Γ allows β, we define $\Gamma \setminus \beta$ as follows:

for all Γ, $\Gamma \setminus \boldsymbol{\tau} = \Gamma$;
if $\Gamma = \Delta, x : \&_{i \in I} (\tilde{\tau}_i)^{\downarrow}$, then $\Gamma \setminus x\mathtt{in}_i\langle\tilde{y}\rangle = \Delta, \tilde{y} : \tilde{\tau}$;
if $\Gamma = \Delta, x : \bigoplus_{i \in I} (\tilde{\tau}_i)^{\uparrow}$, then $\Gamma \setminus \overline{x}\mathtt{in}_i\langle\tilde{y}\rangle = \Delta, \tilde{y} : \tilde{\tau}$;
if $\Gamma = \Delta, x : (\tilde{\tau})^{!}$, then $\Gamma \setminus x\mathtt{pr}_i\langle\tilde{y}\rangle = \Gamma, \tilde{y} : \tilde{\tau}$;
if $\Gamma = \Delta, x : (\tilde{\tau})^{?}$, then $\Gamma \setminus \overline{x}\mathtt{pr}_i\langle\tilde{y}\rangle = \Gamma, \tilde{y} : \tilde{\tau}$.

The environment $\Gamma \setminus \beta$ is what remains after the transition labelled by β has happened. Linear channels are consumed, while replicated channels are not consumed. The new previously bound channels are released. Then the typed transition, written $P \triangleright \Gamma \xrightarrow{\beta} Q \triangleright \Gamma'$ is defined by adding the constraint:

$$\overline{a}\bigoplus_{i\in I}(\tilde{y}_i).P_i \xrightarrow{\overline{a}\mathtt{in}_j\langle\tilde{y}_j\rangle} P_j \quad a\&_{i\in I}(\tilde{y}_i).P_i \xrightarrow{a\mathtt{in}_j\langle\tilde{y}_j\rangle} P_j \quad !a(\tilde{y}).P \xrightarrow{a\mathtt{pr}_i\langle\tilde{y}\rangle} P \mid !a(\tilde{y}).P \quad \overline{a}(\tilde{y}).P \xrightarrow{\overline{a}\mathtt{pr}_i\langle\tilde{y}\rangle} P$$

$$\frac{P\xrightarrow{\beta}P' \quad subj(\beta)\neq x}{(\nu x)P\xrightarrow{\beta}(\nu x)P'} \quad \frac{P\xrightarrow{\beta}P'}{P\mid Q\xrightarrow{\beta}P'\mid Q} \quad \frac{P\xrightarrow{\alpha}P' \quad Q\xrightarrow{\beta}Q' \quad conf(\alpha)=\tilde{y}}{P\mid Q\xrightarrow{\alpha\bullet\beta}(\nu\tilde{y})(P'\mid Q')} \quad \frac{P\equiv_\alpha P' \quad P\xrightarrow{\beta}Q}{P'\xrightarrow{\beta}Q}$$

Fig. 10. Labelled Transition System for the πI-Calculus

$$\text{if } P\xrightarrow{\beta}Q \text{ and } \Gamma \text{ allows } \beta \quad \text{then} \quad P\triangleright\Gamma\xrightarrow{\beta}Q\triangleright\Gamma\setminus\beta$$

The above rule does not allow a linear input action and an output action when there is a complementary channel in the process. For example, if a process has $x:(\tilde{\tau})^!$ in its action type, then output at x is excluded since such actions can never be observed in a typed context – cf. [3]. For a concrete example, consider the process $\overline{a}.b\mid\overline{b}.a$ which is typed in the environment $a:\updownarrow, b:\updownarrow$. Although the process has some untyped transition, none of them is allowed by the environment.

By induction on the rules in Figure 10, we can obtain:

Proposition 6.2. *1. If $P\triangleright\Gamma$, $P\xrightarrow{\beta}Q$ and Γ allows β, then $Q\triangleright\Gamma\setminus\beta$.*
2. (Subject reduction) *If $P\triangleright\Gamma$ and $P\xrightarrow{\tau}Q$, then $Q\triangleright\Gamma$.*
3. (Church Rosser for deterministic processes) *Suppose $P\triangleright\Gamma$ and P is deterministic. Assume $P\xrightarrow{\tau}Q_1$, and $P\xrightarrow{\tau}Q_1$. Then $Q_1\equiv_\alpha Q_2$ or there exists R such that $Q_1\xrightarrow{\tau}R$ and $Q_2\xrightarrow{\tau}R$.*

This is proved by induction on the rules in Figure 10.

Finally we define the notion of typed bisimulation. Let $\mathscr{R}$ be a symmetric relation between judgements such that if $(P\triangleright\Gamma)\,\mathscr{R}\,(P'\triangleright\Gamma')$, then $\Gamma=\Gamma'$. We say that $\mathscr{R}$ is a bisimulation if the following is satisfied:

– whenever $(P\triangleright\Gamma)\,\mathscr{R}\,(P'\triangleright\Gamma)$, $P\triangleright\Gamma\xrightarrow{\beta}Q\triangleright\Gamma\setminus\beta$, then there exists Q' such that $P'\triangleright\Gamma\xrightarrow{\beta}Q'\triangleright\Gamma\setminus\beta$, and $(Q\triangleright\Gamma\setminus\beta)\,\mathscr{R}\,(Q'\triangleright\Gamma\setminus\beta)$.

If there exists a bisimulation between two judgements, we say that they are bisimilar $(P\triangleright\Gamma)\approx(P'\triangleright\Gamma)$.

Then we have:

Proposition 6.3. *$\approx$ is congruent.*

The proof is the same as the proof of Proposition 4.4 in Appendix C.3 of [41].

7 Correspondence between the calculi

7.1 Translation

We are now ready to translate the π-calculus into Name Sharing CCS. The translation is parametrised over a fixed choice for the confidential names. This parametrisation is necessary because π-calculus terms are identified up to α-conversion, and so the identity of bound names is irrelevant, while in Name Sharing CCS, the identity of confidential names is important.

The translation is a family of partial functions $pc[\![-]\!]^\Delta$, indexed by a NCCS type environment Δ, that take a judgment of the π-calculus and return a judgment of NCCS. The functions are only partial because for some choice of names, the parallel composition in NCCS will not be typed.

$$pc\llbracket 0 \triangleright x_i : (\tau_i)^?, y_j : \updownarrow \rrbracket^{x_i : \biguplus_{h\in H}\Gamma_h, y_j : \updownarrow} = 0 \triangleright x_i : \biguplus_{h\in H}\Gamma_h, y_j : \updownarrow$$

$$pc\llbracket (\nu a)P \triangleright \Gamma \rrbracket^{\Delta} = \hat{P} \setminus a \triangleright \Delta$$

$$pc\llbracket \overline{a} \bigoplus_{i\in I} \mathtt{in}_i(\tilde{y}_i).P_i \triangleright \Gamma, a : \bigoplus_{i\in I}(\tilde{\tau}_i)^{\uparrow} \rrbracket^{\Delta, a : \bigoplus_{i\in I}\tilde{z}_i : \hat{\tilde{\tau}}_i} = \overline{a} \bigoplus_{i\in I} \mathtt{in}_i\langle \tilde{z}_i\rangle.\hat{P}_i[\tilde{z}_i/\tilde{y}_i] \triangleright \Delta, a : \bigoplus_{i\in I}\tilde{z}_i : \hat{\tilde{\tau}}_i$$

$$pc\llbracket a \&_{i\in I} \mathtt{in}_i(\tilde{y}_i).P_i \triangleright \Gamma, a : \&_{i\in I}(\tilde{\tau}_i)^{\downarrow} \rrbracket^{\Delta, a : \&_{i\in I}\tilde{z}_i : \hat{\tilde{\tau}}_i} = a \&_{i\in I} \mathtt{in}_i\langle \tilde{z}_i\rangle.\hat{P}_i[\tilde{z}_i/\tilde{y}_i] \triangleright \Delta, a : \&_{i\in I}\tilde{z}_i : \hat{\tilde{\tau}}_i$$

$$pc\llbracket !a(\tilde{y}).P \triangleright \Gamma, a : (\tilde{\tau})^{!} \rrbracket^{\Delta[K], a : \bigotimes_{k\in K}(\tilde{y}^k : \hat{\tilde{\tau}}^k)} =$$
$$\prod_{k\in K} a\mathtt{pr}_k\langle \tilde{y}^k\rangle.\hat{P}[\tilde{y}^k/\tilde{y}][Y^k/Y] \triangleright \Delta[K], a : \bigotimes_{k\in K}(\tilde{y}^k : \hat{\tilde{\tau}}^k)$$

$$pc\llbracket \overline{a}(\tilde{y}).P \triangleright \Gamma, a : (\tilde{\tau})^{?} \rrbracket^{\Delta, a : \biguplus_{h\in H\uplus\{*\}}(\tilde{w}_h : \tilde{\tau}_h)} =$$
$$\overline{a}\mathtt{pr}_j\langle \tilde{w}_j\rangle.\hat{P}[\tilde{w}_j/\tilde{y}] \triangleright \hat{\Gamma}, a : \biguplus_{h\in H\uplus\{j\}}(\tilde{w}_h : \tilde{\tau}_h)$$

$$pc\llbracket P_1 \| P_2 \triangleright \Gamma_1 \odot \Gamma_2 \rrbracket^{\Delta_1 \odot \Delta_2} = (\hat{P}_1 \| \hat{P}_2) \setminus S \triangleright \Delta_1 \odot \Delta_2$$

Fig. 11. Translation from π to NCCS

We define the translation by induction on the derivation of the typing judgment. Without loss of generality, we will assume that all the weakenings are applied to the empty process.

The translation is defined in Figure 11. There, we assume that $pc\llbracket P \triangleright \Gamma \rrbracket^{\Delta} = \hat{P} \triangleright \Delta$, and that $y \in Dom(\Gamma) \Longrightarrow y \in Dom(\Delta)$. In particular, in the translation of the replicated output, we assume $pc\llbracket P \triangleright \Gamma, a : (\tilde{\tau})^?, \tilde{y} : \tilde{\tau} \rrbracket^{\Delta, \tilde{y} : \hat{\tilde{\tau}}_*, a : \biguplus_{h\in H}(\tilde{w}_h : \hat{\tilde{\tau}}_h)^?} = \hat{P} \triangleright \Delta, \tilde{y} : \hat{\tilde{\tau}}_*, a : \biguplus_{h\in H}(\tilde{w}_i : \hat{\tilde{\tau}}_h)^?$. When the assumptions are not satisfied, the translation is not defined. We also put $Y = conf(\hat{P})$, and $S = cl(\Delta_1 \odot \Delta_2)$. Note the way bound variables become confidential information.

We said that the translation is only a partial function. In particular, for the wrong choice of Δ_1, Δ_2, the translation of the parallel composition could be undefined, because $\Delta_1 \odot \Delta_2$ may be undefined. However it is always possible to find suitable Δ_1, Δ_2. Intuitively we can say that in translating typed π into typed NCCS, we perform α-conversion "at compile time".

Lemma 7.1. *For every judgment $P \triangleright \Gamma$ in the π-calculus, there exists an environment Δ such that $pc\llbracket P \triangleright \Gamma \rrbracket^{\Delta}$ is defined. Moreover, for every injective fresh renaming ρ, if $pc\llbracket P \triangleright \Gamma \rrbracket^{\Delta}$ is defined then $pc\llbracket P \triangleright \Gamma \rrbracket^{\Delta[\rho]}$ is defined.*

Example 7.2. We demonstrate how the process which generates an infinite behaviour with infinite new name creation is interpreted into NCCS. Consider the process $\mathtt{Fw}\langle ab\rangle = !a(x).\overline{b}(y).y.\overline{x}$. This agent links two locations a and b and it is called a *forwarder*. It can be derived that $\mathtt{Fw}\langle ab\rangle \triangleright a : \tau, b : \overline{\tau}$ with $\tau = (()^{\uparrow})^{!}$. Consider the process $P_\omega = \mathtt{Fw}\langle ab\rangle \mid \mathtt{Fw}\langle ba\rangle$ so that $P_\omega \triangleright (a : \tau, b : \overline{\tau}) \odot (b : \tau, a : \overline{\tau})$, that is $P_\omega \triangleright a, b : \tau$. One possible translation for $\mathtt{Fw}\langle ab\rangle \triangleright a : (()^{\uparrow})^{!}, b : (()^{\downarrow})^{?}$ is

$$Q_1 = \prod_{k\in K} a\langle x^k\rangle.\overline{b}\langle y^k\rangle.y^k.\overline{x}^k \triangleright a : \bigotimes_{k\in K}(x^k : ()^{\uparrow}), b : \biguplus_{k\in K}(y^k : ()^{\downarrow})$$

while for $\mathtt{Fw}\langle ba\rangle \triangleright b : (()^{\uparrow})^{!}, a : (()^{\downarrow})^{?}$ is

$$Q_2 = \prod_{h\in H} b\langle z^h\rangle.\overline{a}\langle w^h\rangle.w^h.\overline{z}^h \triangleright b : \bigotimes_{h\in H}(z^h : ()^{\uparrow}), a : \biguplus_{h\in H}(()^{?}w^h : ()^{\downarrow})$$

Assuming there are two "synchronising" injective functions $f : K \to H, g : H \to K$, such that $y^k = z^{f(k)}, w^h = x^{g(h)}$ (if not, we can independently perform a fresh injective renaming on both environments), we obtain that the corresponding types for a, b match, so that we can compose the two environments. Therefore the translation of $P_\omega \triangleright a, b : \tau$ is $(Q_1 \mid Q_2) \setminus S \triangleright \Delta$ for

$$\Delta = a : \bigotimes_{k\in K\setminus g(H)}(x^k : ()^\uparrow), b : \bigotimes_{h\in H\setminus f(K)}(z^h : ()^\uparrow).$$

The reader can check that any transition of P_ω is matched by a corresponding transition of its translation. This is what we formally show next.

7.2 Adequacy

To show the correctness of the translation, we first prove the correspondence between the labelled transition semantics.

Theorem 7.3. *Suppose $P \triangleright \Gamma \xrightarrow{\beta} P' \triangleright \Gamma \setminus \beta$ in the π-calculus, and that $pc[\![P \triangleright \Gamma]\!]^\Delta$ is defined. Then for every injective fresh renaming ρ $pc[\![P \triangleright \Gamma]\!]^{\Delta[\rho]} \xrightarrow{\beta[\rho]} pc[\![P' \triangleright \Gamma \setminus \beta]\!]^{\Delta[\rho]\setminus\beta[\rho]}$.*

Conversely, suppose $pc[\![P \triangleright \Gamma]\!]^\Delta \xrightarrow{\beta} Q \triangleright \Delta \setminus \beta$. Then there exists P' such that $P \triangleright \Gamma \xrightarrow{\beta} P' \triangleright \Gamma \setminus \beta$ and $pc[\![P' \triangleright \Gamma \setminus \beta]\!]^{\Delta\setminus\beta} = Q \triangleright \Delta \setminus \beta$.

The soundness is then a corollary.

Corollary 7.4 (Soundness). *Suppose that for some Δ, $pc[\![P \triangleright \Gamma]\!]^\Delta \approx pc[\![P' \triangleright \Gamma]\!]^\Delta$. Then $P \triangleright \Gamma \approx P' \triangleright \Gamma$.*

Despite Theorem 7.3, full abstraction fails. This happens for subtle reasons, and we conjecture that the translation is fully abstract if we consider some "observational congruence" for NCCS.

However, soundness is precisely what we needed to obtain a sound semantics in terms of event structures. Moreover we have already argued that we cannot expect the event structure semantics to be fully abstract.

7.3 Event structure semantics of the π-calculus

By composing the translation obtained in this section with the event structure semantics of Section 5, we obtain an event structure semantics of the π-calculus.

Given a π-calculus judgment $P \triangleright \Gamma$, we define

$$[\![P \triangleright \Gamma]\!]^\Delta = [\![pc[\![P \triangleright \Gamma]\!]^\Delta]\!]$$

We thus have

Lemma 7.5. *For every judgment $P \triangleright \Gamma$ in the π-calculus, there exists an environment Δ such that $[\![P \triangleright \Gamma]\!]^\Delta$ is defined. When this is the case $[\![P \triangleright \Gamma]\!]^\Delta$ is a confusion free event structure, and $[\![P \triangleright \Gamma]\!]^\Delta \triangleright \Delta$.*

Proposition 7.6 (Soundness). *Suppose that for some Δ, $[\![P \triangleright \Gamma]\!]^\Delta = [\![P' \triangleright \Gamma]\!]^\Delta$. Then $P \triangleright \Gamma \approx P' \triangleright \Gamma$.*

Note that the event structure semantics of CCS is already not fully abstract with respect to bisimulation [35], hence the other direction does not hold in our case either.

However, there is another kind of correspondence between the labelled transition systems and the event structures, analogous to the one discussed in Section 5.4. Combining Theorem 5.8 with Theorem 7.3, we obtain:

Theorem 7.7. *Suppose $P \triangleright \Gamma \xrightarrow{\beta} P' \triangleright \Gamma \setminus \beta$ in the π-calculus, and that $[\![P \triangleright \Gamma]\!]^\Delta$ is defined. Then for every injective fresh renaming ρ $[\![P \triangleright \Gamma]\!]^{\Delta[\rho]} \xrightarrow{\beta[\rho]} \cong [\![P' \triangleright \Gamma \setminus \beta]\!]^{\Delta[\rho]\setminus\beta[\rho]}$.*

Conversely, suppose $[\![P \triangleright \Gamma]\!]^\Delta \xrightarrow{\beta} \mathscr{E}'$. Then there exists P' such that $P \triangleright \Gamma \xrightarrow{\beta} P' \triangleright \Gamma \setminus \beta$ and $[\![P' \triangleright \Gamma \setminus \beta]\!]^{\Delta\setminus\beta} \cong \mathscr{E}'$.

8 Conclusions and related work

This paper has provided a typing system for event structures and exploited it to give an event structure semantics of the π-calculus. As far as we know, this work offers the first formalisation of a notion of types in event structures, and the first event structure semantics of the π-calculus.

There are several causal models for the π-calculus, that use different techniques. In [5, 11], the causal relations between transitions are represented by "proofs" of the transitions which identify different occurrences of the same transition. In our case a similar role is played by names in types. In [9], a more abstract approach is followed, which involves indexed transition systems. In [18], a semantics of the π-calculus in terms of pomsets is given, following ideas from dataflow theory. The two papers [8, 13] present Petri nets semantics of the π-calculus. Since we can unfold Petri nets into event structures, these could indirectly provide event structure semantics of the π-calculus. In [2], an event structure unfolding of double push-out rewriting systems is studied, and this also could indirectly provide an event structure semantics of the π-calculus via the double push-out semantics of the π-calculus presented in [27]. In [6], Petri Nets are used to provide a type theory for the Join-calculus, a language with several features in common with the π-calculus. None of the above semantics directly uses event structures and no notion of compositional typing systems in true concurrent models is presented. In addition, none of them is used to study a correspondence between semantics and behavioural properties of the π-calculus in our sense.

In [38], event structures are used in a different way to give semantics to a process language, a kind of value passing CCS. That technique does not apply yet to the π-calculus where we need to model creation of new names, although recent work [37] is moving in that direction.

A syntactic condition that imposes a similar restriction to our typing system was first introduced by Milner, in his *confluent* CCS [25]. The typing system we introduce is inspired by the linear typing system for the π-calculus [21, 41, 3].

Infinite behaviour is introduced in our version of CCS by means of the infinite parallel composition. NCCS does not support recursion. Infinite parallel composition is similar to replication in that it provides infinite behaviour "in width" rather that "in depth". Recent studies on recursion versus replication are [7, 15].

Future works include extending this approach to a probabilistic framework, for instance the probabilistic π-calculus [16], by using a typed version of probabilistic event structures [32]. The typed λ-calculus can be encoded into the typed π-calculus. This provides an event structure semantics of the λ-calculus, that we want to study in details. Also the types of the λ-calculus are given an event structure semantics. We aim at comparing this "true concurrent" semantics of the λ-types with concurrent games [24, 22], and with ludics nets [14].

An event structure *terminates* if all its maximal configurations are finite. It would be interesting to study a typing system of event structures that guarantees termination applying the idea of the strongly normalising typing system of the π-calculus [41].

A Proofs

Proof of Lemma 2.6 We prove it by induction on the joint size of x, x'. The base case is vacuously true. Now take $(x, e_1, e_2), (x', e_1, e_2) \in E$ with $x \neq x'$. Since x, x' are downward closed sets, if their maximal elements coincide, they coincide. Therefore, w.l.o.g. there must be a maximal element $(y, d_1, d_2) \in x$ such that $(y, d_1, d_2) \notin x'$. By definition of E, and without loss of generality, we can assume that $d_1 \in \mathit{parents}(e_1)$. Therefore, by definition of E, there must be a $(y', d_1, d_2') \in x'$. Suppose $d_2 \neq d_2'$. Then by definition of conflict $(y, d_1, d_2) \smile (y', d_1, d_2')$. If $d_2 = d_2'$ then it must be $y \neq y'$. Then by induction hypothesis there exist $f \in y, f' \in y'$ such that $f \smile f'$. And since x, x' are downward closed, we have $f \in x, f' \in x'$.

Proof of Theorem 2.7 Recall the the definition of $(E, \leq, \smile)$. In order to show that it is an event structure, we first o have to show that the relation $\leq$ is a partial order. We have that

- it is reflexive by construction;
- it is antisymmetric: suppose $e' \leq e = (x, e_1, e_2)$. If $e' \neq e$, then, by construction $h(e') < h(e)$, so that it cannot be $e \leq e'$.
- it is transitive: suppose $e' \leq e \leq d = (y, d_1, d_2)$. This means that $e \in y$. Since, by construction, y is downward closed, this means that $e' \in y$, so that $e' \leq d$.

Next, for every event $e = (x, e_1, e_2)$, we have that $[e)$ is finite, as it coincides with x.

Then we need to show that the conflict is irreflexive and hereditary. It is hereditary essentially by definition: suppose $e := (x, e_1, e_2) \smile d := (y, d_1, d_2)$, and let $d \leq d' := (y', d'_1, d'_2)$. By considering all the cases of the definition of $e \smile d$, we derive $e \smile d'$. For instance, suppose there exists $e' := (x', e'_1, e'_2) \leq e$ such that $e'_1 \asymp d_1$, and $e' \neq d$. This means that $e' \smile d$. Notice that $e' \leq e$, and $d \leq d'$. By the fourth clause of the definition, $e \smile d'$. The other cases are analogous.

To prove that the conflict relation is irreflexive, suppose $(x, e_1, e_2) \smile (x, e_1, e_2)$. This cannot be because there are $e, d \in x$ such that $e \smile d$, as it contradicts the fact that x is a configuration. Therefore, there must exist $(x', e'_1, e'_2) \in x$ such that $(x', e'_1, e'_2) \smile (x, e_1, e_2)$. Take a minimal such. Then it must be $e'_1 \asymp e_1$ or $e'_2 \asymp e_2$. But this contradicts the definition of E.

Now we have to show that such event structure is the categorical product of $\mathscr{E}_1$, $\mathscr{E}_2$. First thing to show is that projections are morphisms. Using Proposition 2.5, it is enough to show that they reflect reflexive conflict and preserves downward closure.

- Take $e, e' \in E$ and suppose by that $\pi_1(e) \asymp \pi_1(e')$. Then, by definition we have $e \asymp e'$.
- To show that π_1 preserves downward closure let $e = (x, e_1, e_2)$ suppose $e'_1 \leq e_1 = \pi_1(e)$. Then we show that there is a $e' \leq e$ such that $\pi_1(e') = e'_1$. By induction on the height of e: the basis is vacuously true, since e_1 is minimal. For the step, consider first the case where $e'_1 \in \mathit{parents}(e_1)$. Then, by definition of E, we have that there exists $e' = (x', e'_1, e'_2) \in x$. Therefore $e' \leq e$ and $\pi_1(e') = e'_1$. If $e'_1 \notin \mathit{parents}(e_1)$, then there is a $e''_1 \in \mathit{parents}(e_1)$ such that $e'_1 \leq e''_1 \leq e_1$ so that there is $e'' = (x'', e''_1, e''_2) \in x$. By induction hypothesis there is $e' \in x''$ such that $\pi_1(e') = e'_1$. And by transitivity, $e' \leq e$.

Now we want to show that $\mathscr{E}$ enjoys the universal property that makes it a categorical product. That is for every event structure $\mathscr{D}$, such that there are morphisms $f_1 : \mathscr{D} \to \mathscr{E}_1, f_2 : \mathscr{D} \to \mathscr{E}_2$, there exists a unique $f : \mathscr{D} \to \mathscr{E}$ such that $\pi_1 \circ f = f_1$ and $\pi_2 \circ f = f_2$.

Clearly, if such f exists, it must be defined as $f(d) = (x, f_1(d), f_2(d))$, for some x. By this we mean $f(d) = (x, f_1(d), *)$, if $f_2(d)$ is undefined, $f(d) = (x, *, f_2(d))$, if $f_1(d)$ is undefined, and undefined if both are undefined. We now define x, by induction on the size of $[d)$. Suppose d is minimal. Then, since f_1, f_2 are morphisms and in particular preserve downward closure, we have that $f_1(d), f_2(d)$ are both minimal. Since every maximal element of x must contain the parent of at least one of them, the only possibility is that x be empty.

Putting $f(d) = (\emptyset, f_1(d), f_2(d))$, we obtain, that, on element of height 0,

- $f(d)$ is uniquely defined: we have seen that all choices are forced
- f reflects reflexive conflict: suppose $(\emptyset, f_1(d), f_2(d)) \asymp (\emptyset, f_1(d'), f_2(d'))$, then either $f_1(d) \asymp f_1(d')$ or $f_2(d) \asymp f_2(d')$. In the first case, since f_1 is a morphism, and thus reflects reflexive conflict, we have $d \asymp d'$. Symmetrically for the other case.
- f preserves downward closure vacuously

Now suppose f is uniquely defined for all elements of height less or equal than n, it reflects reflexive conflict and preserves downward closure. Consider d of height $n+1$. We want to define $f(d) = (x, f_1(d), f_2(d))$. Define x as follows. For a set A, let $\downarrow A$ be the downward closure of A. Let $X = \{f(d') \mid d' < d \;\&\; [\; f_1(d') \in \mathit{parents}(f_1(d))$ or $f_2(d') \in \mathit{parents}(f_2(d)) \;]\}$ and define x as $\downarrow X$. First of all we should check that this is indeed an element of E. x is downward closed by definition. It is finite because X is and each element of X has finitely many predecessors. Suppose there are $d', d'' < d$ such that $f(d') \smile f(d'')$. We know by induction that f reflects reflexive conflict on elements of height smaller than d, which means that $d' \smile d''$, contradiction.

Now the maximal elements of x contain either a parent of $f_1(d)$ or a parent of $f_2(d)$ by construction. Take a parent e_1 of $f_1(d)$. I claim that e_1 is of the form $f_1(d')$ for some $d' < d$. Since $e_1 \in \mathit{parents}(f_1(d))$, in particular $e_1 \leq f_1(d)$. since f_1 preserves downward closure, there must exists d' as above. Thus all parents are represented in X. Finally, suppose there is $(z, e_1, e_2) \in x$ such that $e_1 \asymp f_1(d)$ or $e_2 \asymp f_2(d)$. If $(z, e_1, e_2) \in X$, then $(z, e_1, e_2) = f(d')$ for some $d' < d$. So that $e_1 = f_1(d')$, and $e_2 = f_2(d')$. Since f_1, f_2 reflect reflexive conflict, we would have $d' \smile d$, contradiction. Otherwise there must be $f(d') \in X$ such that $(z, e_1, e_2) < f(d')$. Since f preserves downward closure on elements of height less or equal than n, there must be $d'' < d'$ such that $f(d'') = (z, e_1, e_2)$. As above we conclude $d'' \smile d$, contradiction.

Thus putting $f(d) = (x, f_1(d), f_2(d))$, we have that f is well defined on d. Moreover

- $f(d)$ is uniquely defined: suppose we have another possible x. Since f must preserve downward closure, for all $e \in x$, we have that $e = f(d')$ for some $d' < d$. Now, suppose there is an element $f(d') \in X$ which is not in x. W.l.o.g assume that $f_1(d') \in \mathit{parents}(f_1(d))$. Then, there must be an element $e' = (y, f_1(d'), d'_2)$ maximal in x. By the observation above it must be $e' = f(d')$, contradiction.
- f preserves downward closure: take d, and consider $e \leq f(d)$. By construction, either $e \in X$, in which case we have $e = f(d')$ for some $d' < d$, of $e \leq e' \in X$, in which case we have $e' = f(d')$ for $d' < d$. Since, by induction f preserves downward closure, we have $e = f(d'')$ for $d'' < d' < d$.
- f reflects reflexive conflict: suppose $(x, f_1(d), f_2(d)) \asymp (x', f_1(d'), f_2(d'))$, then
  - either $f_1(d) \asymp f_1(d')$ or $f_2(d) \asymp f_2(d')$. In either case, since f_1, f_2 reflects reflexive conflict, we have $d \asymp d'$.
  - there exists $(x'', e_1, e_2) \leq (x', f_1(d'), f_2(d')$, such that $f_1(d) \asymp e_1$ or $f_2(d) \asymp e_2$. Since f preserves downward closure, we have $(x'', e_1, e_2) = f(d'')$ for some $d'' < d'$ and we reason as above.
  - the symmetric case is similar
  - there exists $(y, e_1, e_2) \leq (x, f_1(d), f_2(d))$ and there exists $(y', e'_1, e'_2) \leq (x', f_1(d'), f_2(d'))$, and the reasoning is as above, using that f preserves downward closure.

Thus f is a morphism, is uniquely defined for every $d \in D$, and commutes with the projections. This concludes the proof.

Proof of Lemma 3.1 Suppose $\Gamma_1 \odot \Gamma_2$ is defined. Take a name x in $cl(\Gamma_1 \odot \Gamma_2)$. Then there must exist a name y such that $\Gamma_1(y) = \tau_1$ and $\Gamma_2(y) = \tau_2$ and $\mathit{match}[\tau_1, \tau_2] \to S$ and $x \in S$. Then it means that x appears in τ_1, τ_2, and by uniqueness it cannot be in the domain of either Γ_1 or Γ_2, so it cannot appear in the domain of $\Gamma_1 \odot \Gamma_2$. Furthermore x cannot appear within any other type. If τ_1,τ_2 are branching/selection types, then $\Gamma_1 \odot \Gamma_2(y) = \updownarrow$, so that x does not appear in $\Gamma_1 \odot \Gamma_2$. If τ_1,τ_2 are offer/request types, then the residual type does not contain x, which again implies x does not appear in $\Gamma_1 \odot \Gamma_2$. Therefore x is not allowed.

Proof of Proposition 3.2 Consider a minimal element of $[\![\Gamma_1]\!]$.

- If it synchronises, by the condition on the definition of $\Gamma_1 \odot \Gamma_2$, it must synchronise with a dual minimal element in $[\![\Gamma_2]\!]$. Every event above these two events is either a $\boldsymbol{\tau}$, or it is not allowed, therefore it is deleted by the restriction.

– If it does not synchronise it is left alone, with all above it not synchronising either, and not being restricted.

Thus we can think of $[\![\Gamma_1 \odot \Gamma_2]\!]$, as a disjoint union of $[\![\Gamma_1]\!]$ and $[\![\Gamma_2]\!]$, plus some hiding.

Proof of Lemma 3.4 Suppose $\mathscr{E} \triangleright \Gamma$, witnessed by a morphism $f : \mathscr{E} \to [\![\Gamma]\!]$.

– Let $e, e' \in E$ be such that $\lambda(e) = \lambda(e') \neq \tau$. Therefore, by uniqueness of the labels in $[\![\Gamma]\!]$, $f(e) = f(e')$, and since f reflects reflexive conflict, we have $e \smile e'$.
– A similar reasoning applies for the case when $\lambda(e) = a\mathtt{in}_i\langle\tilde{x}\rangle$ and $\lambda(e') = a\mathtt{in}_j\langle\tilde{y}\rangle$. Then $f(e), f(e')$ belong to the same cell, and thus they are in conflict. Since f reflects conflict, we have $e \smile e'$.
– Suppose $E \triangleright \Gamma$, and let $e, e' \in E$ be such that $e \smile_\mu e'$. Then they belong to the same cell, and by definition they must have same subject but different branch.

Proof of Theorem 3.5 Define $\Gamma = \Gamma_1 \odot \Gamma_2$ Suppose $\mathscr{E}_1 \triangleright \Gamma_1$, and $\mathscr{E}_2 \triangleright \Gamma_2$. Let $\mathscr{E} = (\mathscr{E}_1 \| \mathscr{E}_2) \setminus Dis(\Gamma)$. We invite the reader to review the definition of the product of event structures, and the consequent definition of parallel composition.

Lemma A.1. *Let $(x, e_1, e_2), (y, d_1, d_2)$ be two events in $\mathscr{E}$. Suppose $(x, e_1, e_2) \smile (y, d_1, d_2)$. Then there exists $(x', e_1', e_2') \in x, (y', d_1', d_2') \in y$ such that either $e_1' \smile_\mu d_1'$ or $d_1' \smile_\mu d_2'$.*

We check this by cases, on the definition of conflict.

– $e_1 \smile d_1$. In this case there must exists $e_1' \leq e_1$ and $e_2' \leq e_2$ such that $e_1' \smile_\mu e_2'$. Since projection are morphisms of event structures, and since in particular preserve configurations, for every event f below e_1 there must be an event in E below (x, e_1, e_2) that is projected onto f. And similar for d_1. Therefore there are $(x', e_1', e_2') \in x, (y', d_1', d_2') \in y$ for some x', y', e_2', d_2'. Note also that $(x', e_1', e_2') \smile (y', d_1', d_2')$.
– $e_2 \smile d_2$ is symmetric.
– $e_1 = d_1$ and $e_2 \neq d_2$. This is the crucial case, where we use the typing. In this case it is not possible that $e_2 = *$ and $d_2 \neq *$ (nor symmetrically). This is because of the typing. If the label dual of e_1 is not in Γ_2 then both $e_2, d_2 = *$. If the label dual of e_1 is in Γ_2, then the label of e_1 is matched and thus it becomes disallowed, so that the event $(x, e_1, *)$ is removed. So both e_2 and d_2 have the same label (the dual of the label of e_1). Thus they are mapped on the same event in $[\![\Gamma_2]\!]$, and thus they must be in conflict. Then we reason as above.
– $e_2 = d_2$ and $e_1 \neq d_1$ is symmetric.
– $e_1 = d_1$ and $e_2 = d_2$. Then the conclusion follow from stability (Lemma 2.6).
– suppose there exists $(\bar{x}, \bar{e_1}, \bar{e_2} \in x$ such that $\bar{e_1} \asymp d_1$ or $\bar{e_2} \asymp d_2$. Then we reason as above to find $(x', e_1', e_2') \in \bar{x}, (y', d_1', d_2') \in y$ such that either $e_1' \smile_\mu d_1'$ or $d_1' \smile_\mu d_2'$. Note that, by transitivity, $(x', e_1', e_2') \in x$.
– the symmetric case is analogous.
– Suppose there is $e \in x$, and $d \in y$ such that $e \smile d$. By wellfoundedness this case can be reduce to one of the previous ones.

Lemma A.2. *If $(x, e_1, e_2) \asymp_\mu (y, d_1, d_2)$, then their labels have the same subject, but different branch and different confidential names.*

By Lemma A.1, either $e_1 \asymp_\mu d_1$ or $e_2 \asymp_\mu d_2$ (or both). In the first case, the labels of e_1, d_1 have the same subject. Thus the labels of $(x, e_1, e_2), (y, d_1, d_2)$ also have the same subject (whether they are synchronisation labels or not). The second case is symmetric.

Lemma A.3. *If $(x, e_1, e_2) \asymp_\mu (y, d_1, d_2)$, then $x = y$*

First suppose $e_2 = d_2 = *$. Then $e_1 \asymp_\mu d_1$. Dually when $e_1 = d_1 = *$. Finally, suppose $e_1, d_1, e_2, d_2 \neq *$. Without loss of generality we have $e_1 \asymp_\mu d_1$. But then $e_2 \asymp d_2$, because they have dual labels. Then it must be $e_2 \asymp_\mu d_2$ because otherwise we would not have $(x, e_1, e_2) \asymp_\mu (y, d_1, d_2)$.

In any cases we have that $(x, d_1, d_2) \in E$. Indeed it satisfies the condition for being in the product (because $parents(e_1) = parents(d_1)$ and $parents(e_2) = parents(d_2)$), and it is allowed if and only if (x, e_1, e_2) is allowed. Suppose $x \neq y$. By stability we have that there are $e' \in x, d' \in y$ such that $e' \smile d'$. Which contradicts $(x, e_1, e_2) \asymp_\mu (y, d_1, d_2)$.

Lemma A.4. *The relation $\asymp_\mu$ is transitive in $\mathscr{E}$.*

Suppose $(x, e_1, e_2) \asymp_\mu (y, d_1, d_2)$, and $(y, d_1, d_2) \asymp_\mu (z, g_1, g_2)$, Then reasoning as above we have that $e_1 \asymp_\mu d_1 \asymp_\mu g_1$ and $e_2 \asymp_\mu d_2 \asymp_\mu g_2$. Which implies $e_1 \asymp_\mu g_1$ and $e_2 \asymp_\mu g_2$, from which we derive $(x, e_1, e_2) \asymp_\mu (z, g_1, g_2)$.

Lemmas A.3, and A.4 together prove that $\mathscr{E}$ is confusion free.

To prove that $\mathscr{E} \rhd \Gamma$, suppose $f_1 : E_1 \to [\![\Gamma_1]\!]$ and $f_2 : E_2 \to [\![\Gamma_2]\!]$. Recall that $[\![\Gamma]\!] = ([\![\Gamma_1]\!] \| [\![\Gamma_2]\!]) \setminus (Dis(\Gamma) \cup \boldsymbol{\tau})$. As we observed we can think of $[\![\Gamma]\!]$ as the disjoint union of $[\![\Gamma_1]\!]$ and $[\![\Gamma_2]\!]$, plus some hiding.

We define the following partial function $f : \mathscr{E} \to [\![\Gamma]\!]$. $f(x, e_1, *) = f_1(e_1)$, $f(x, *, e_2) = f_2(e_2)$ (where by equality we mean *weak equality*), and undefined otherwise. We have to check that f satisfies the conditions required. The first two conditions are a consequence of (the proof) of the first part of the theorem. It remains to show that f is a morphism of event structures. This follows from general principles, but we repeat the proof here.

We have to check that if $d \leq f(x, e_1, e_2)$ in $[\![\Gamma]\!]$, then there exists (y, d_1, d_2) in $\mathscr{E}$ such that $f(y, d_1, d_2) = d$. Without loss of generality, we assume $e_2 = *$, so that $f(x, e_1, e_2) = f_1(e_1)$. Let $d \leq f_1(e_1)$. Since f_1 is a morphism, then there is $d_1 \leq e_1$ such that $f_1(d_1) = d$. Since projections are morphisms, there must be a $(y, d_1, d_2) \leq (x, e_1, e_2)$. I claim that d_2 must be equal to $*$, so that $f(y, d_1, d_2) = f_1(d_1) = d$. If d_2 were not $*$, then its label would be dual to label of d_1. This means that both labels are in $Dis(\Gamma)$, and that no event in $[\![\Gamma]\!]$, and in particular the d, can be labelled by either of them. This contradicts $f_1(d_1) = d$.

Then we have to check that f reflects $\asymp$. So, suppose $f(x, e_1, e_2) \asymp f(x', e'_1, e'_2)$. By the structure of $[\![\Gamma]\!]$ it cannot be that $f(x, e_1, *) \asymp f(x', *, e'_2)$, because they are mapped to disjoint concurrent components. Therefore, w.l.o.g, the only case to consider is $f(x, e_1, *) \asymp f(x', e'_1, *)$. This means $f_1(e_1) \asymp f_1(e'_1)$. Since f_1 is a morphism, then $e_1 \asymp e'_1$, which implies $(x, e_1, *) \asymp (x', e'_1, *)$.

Proof of Proposition 4.2 By a straightforward case analysis.

Proof of Theorem 5.1 The proof is by induction on the semantics. All the cases are easily done directly, with the exception of the parallel composition. The case of the parallel composition is a direct consequence of Theorem 3.5.

Proof of Theorem 5.8 The proof is by induction on the rules of the operational semantics. All cases are rather straightforward, except the parallel composition. For this we need the following lemma. To avoid distinguishing different cases, lets say that, for every event structure $\mathscr{E}$, we have $\mathscr{E} \xrightarrow{*} \mathscr{E} \lfloor * = \mathscr{E}$.

Lemma A.5. *Let $\cong$ denote isomorphism of event structures. We have that $\mathscr{E}_1 \xrightarrow{\alpha} \mathscr{E}_1 \lfloor e_1$, and $\mathscr{E}_2 \xrightarrow{\beta} \mathscr{E}_2 \lfloor e_2$ if and only if $\mathscr{E}_1 \| \mathscr{E}_2 \xrightarrow{\alpha \bullet \beta} \mathscr{E}_1 \| \mathscr{E}_2 \lfloor (\emptyset, e_1, e_2)$. Moreover, in such a case, we have $\mathscr{E}_1 \| \mathscr{E}_2 \lfloor (\emptyset, e_1, e_2) \cong (\mathscr{E}_1 \lfloor e_1) \| (\mathscr{E}_2 \lfloor e_2)$.*

The first part of theorem is straightforward: if e_1, e_2 are minimal in $\mathscr{E}_1, \mathscr{E}_2$, then $(\emptyset, e_1, e_2)$ is a minimal event in $\mathscr{E}_1 \| \mathscr{E}_2$, and vice versa. Assuming this is the case, we are now going to prove that $\mathscr{E}_1\|\mathscr{E}_2\lfloor(\emptyset,e_1,e_2) \cong (\mathscr{E}_1\lfloor e_1)\|(\mathscr{E}_2\lfloor e_2)$. We will define a bijective function $f : \mathscr{E}_1\|\mathscr{E}_2\lfloor(\emptyset,e_1,e_2) \to (\mathscr{E}_1\lfloor e_1)\|(\mathscr{E}_2\lfloor e_2)$, such that both f and f^{-1} are morphism of event structure. We define f by induction on the height of the events. Also by induction we show the properties required. That is we prove that

- for every n, f is bijective on elements of height n;
- f preserves and reflects the conflict relation;
- f preserves and reflects the order relation;
- $\Pi_1 \circ f = \Pi_1$ and $\Pi_2 \circ f = \Pi_2$, where Π_1, Π_2 denote the projections in the parallel composition.

In particular. the above properties imply that both f, and f^{-1} are morphisms of event structure. The preservation of the labels follows from the last point, noting that the labels of an event in the product depend only on the labels of the projected events.

Base: height = 0
Events of height 0 in $\mathscr{E}_1\|\mathscr{E}_2\lfloor(\emptyset,e_1,e_2)$ are of two forms:

- the form $(\emptyset,d_1,d_2)$, with d_1 minimal in $\mathscr{E}_1$ and d_2 minimal in $\mathscr{E}_2$ (when different from $*$)[1]. In such a case we define $f(\emptyset,d_1,d_2) = (\emptyset,d_1,d_2)$.
- the form $((\emptyset,e_1,e_2),d_1,d_2)$, with $e_1 \leq d_1$ and d_2 minimal in $\mathscr{E}_2$, or $e_2 \leq d_2$ and d_1 minimal in $\mathscr{E}_1$, or both $e_1 \leq d_1, e_2 \leq d_2$. In such a case we define $f(\emptyset,d_1,d_2) = (\emptyset,d_1,d_2)$.

Note that from the discussion above, it follows that $(\emptyset,d_1,d_2)$ and $((\emptyset,e_1,e_2),d_1,d_2)$ cannot be both events in $\mathscr{E}_1\|\mathscr{E}_2\lfloor(\emptyset,e_1,e_2)$. We prove that f is well defined on events of height 0. Consider $d = (\emptyset,d_1,d_2)$. Then both d_1,d_2 are minimal in $\mathscr{E}_1,\mathscr{E}_2$ respectively. Also it is not the case that $d_1 \asymp e_1$, nor $d_2 \asymp e_2$, as otherwise we would have $(\emptyset,d_1,d_2) \asymp (\emptyset,e_1,e_2)$. This means that d_1,d_2 belong to $\mathscr{E}_1\lfloor e_1, \mathscr{E}_2\lfloor e_2$ and are minimal there. So that $f(d) = (\emptyset,d_1,d_2) \in (\mathscr{E}_1\lfloor e_1)\|(\mathscr{E}_2\lfloor e_2)$. A similar reasoning applies when $d = ((\emptyset,e_1,e_2),d_1,d_2)$. Now we prove

- f is bijective on events of height 0; it is surjective: take an event $(\emptyset,d_1,d_2)$ in $(\mathscr{E}_1\lfloor e_1)\|(\mathscr{E}_2\lfloor e_2)$. There are several cases. If both d_1 is minimal in $\mathscr{E}_1$ and d_2 is minimal in $\mathscr{E}_2$, and it is not the case that $e_1 \asymp d_1$ nor $e_2 \asymp d_2$, then $(\emptyset,d_1,d_2) \in \mathscr{E}_1\|\mathscr{E}_2\lfloor(\emptyset,e_1,e_2)$. Similarly, in the other cases, it is easy to see that $((\emptyset,e_1,e_2),d_1,d_2) \in \mathscr{E}_1\|\mathscr{E}_2\lfloor(\emptyset,e_1,e_2)$. Also f is injective. The only thing to check is that $(\emptyset,d_1,d_2)$ and $((\emptyset,e_1,e_2),d_1,d_2)$ cannot be both events in $\mathscr{E}_1\|\mathscr{E}_2\lfloor(\emptyset,e_1,e_2)$, which, as we have observed, is the case.
- f preserves and reflects conflict on events of height 0. This is easily verified by checking all the cases of definition of conflict. Note that it cannot be the case that $(\emptyset,d_1,d_2) \asymp (\emptyset,e_1,e_2)$, as such events do not belong to $\mathscr{E}_1\|\mathscr{E}_2\lfloor(\emptyset,e_1,e_2)$.
- f preserves and reflects order on events of height 0, trivially.
- $\Pi_1 \circ f = \Pi_1$ and $\Pi_2 \circ f = \Pi_2$, by definition.

Step: height = $n+1$
We assume that f is defined for all events of height $\leq n$, and that it satisfies the required properties there. On events of height $n+1$, we define f as follows. $f(x,d_1,d_2) = (f(x),d_1,d_2)$. We prove that f is well defined. Note that in order to show that $(f(x),d_1,d_2)$ is an event, we only use properties of $\Pi_1(f(x))$ and $\Pi_2(f(x))$, by induction hypothesis they coincide with $\Pi_1(x), \Pi_2(x)$ respectively. We consider one case, the others being similar. Suppose $d_1 \in E_1, d_2 \in E_2$. Then let y be the set of maximal elements of x. Since f preserves and reflects order, we have that $f(y)$ is the set of maximal elements of $f(x)$. Let $y_1 = \Pi_1(y), y_2 = \Pi_2(y)$. Note that we also have $y_1 = \Pi_1(f(y)), y_2 = \Pi_2(f(y))$. Since (x,d_1,d_2) is an event, we have

- if $(z,d_1,d_2) \in y$, then either $d_1 \in \mathit{parents}(e_1)$ or$d_2 \in \mathit{parents}(e_2)$;

[1] We omit this remark in the following, but it has to be considered implicit throughout

- for all $d_1 \in \mathit{parents}(e_1)$, there exists $(z, d_1, d_2) \in x$;
- for all $d_2 \in \mathit{parents}(e_2)$ there exists $(z, d_1, d_2) \in x$.
- for no $d_1 \in \Pi_1(x)$, $d_1 \asymp e_1$ and for no $d_2 \in \Pi_2(x)$, $d_2 \asymp e_2$.

These conditions, show that $(f(x), d_1, d_2)$ is also an event.

We now prove that

- f is bijective on event of height $n+1$. First, if (x, d_1, d_2) is of height $n+1$, so is $(f(x), d_1, d_2)$, because by induction hypothesis, f is bijective on events of height n, so that x contains one such event if and only if $f(x)$ does. To prove that f is surjective, consider now an event $(y, d_1, d_2) \in (\mathscr{E}_1 \lfloor e_1) \| (\mathscr{E}_2 \lfloor e_2)$. Since f is biejctive on events of height $\leq n$, we have that there exists x such that $y = f(x)$, and moreover since f preserves and reflects order and conflict, x is a configuration if and only if $f(x)$ is. We have to argue that if $(f(x), d_1, d_2)$ is an event of $(\mathscr{E}_1 \lfloor e_1 \| \mathscr{E}_2 \lfloor e_2)$ then (x, d_1, d_2) is an event of $\mathscr{E}_1 \| \mathscr{E}_2 \lfloor (\emptyset, e_1, e_2)$. This is done in a similar way than the base case. To prove that f is injective, consider $(x, d_1, d_2), (x', d_1, d_2)$, such that $f(x) = f(x')$. By induction hypothesis f is injective, so that $x = x'$ and we are done.
- f preserves and reflects conflict. This is done as in the base case.
- f preserves and reflects order. In fact by definition $d \in x$ if and only if $f(d) \in f(x)$, which is precisely what we need.
- $\Pi_1 \circ f = \Pi_1$ and $\Pi_2 \circ f = \Pi_2$, by definition.

This concludes the proof.

Proof of Lemma 7.1 Given a NCCS type σ, we define its *erasure* $er(\sigma)$ to be the π type obtained from σ by removing all confidential names. It is a partial function defined as follows

- $er(y_1 : \sigma_1, \ldots, y_n : \sigma_n) = er(\sigma_1), \ldots, er(\sigma_n)$
- $er(\&_{i \in I} \Gamma_i) = (\&_{i \in I} er(\Gamma_i))^{\downarrow}$
- $er(\bigoplus_{i \in I} \Gamma_i) = (\bigoplus_{i \in I} er(\Gamma_i))^{\uparrow}$
- $er(\bigotimes_{i \in I} \Gamma_i) = (er(\Gamma))^{!}$ if for all $i \in I$, $er(\Gamma_i) = er(\Gamma)$.
- $er(\biguplus_{i \in I} \Gamma_i) = (er(\Gamma))^{?}$ if for all $i \in I$, $er(\Gamma_i) = er(\Gamma)$.
- $er(\updownarrow) = \updownarrow$

We have the following lemma.

Lemma A.6. *Suppose $er(\sigma) = \overline{er(\tau)}$, and suppose σ, τ have disjoint sets of names. Suppose for every type of the form $\bigotimes_{k \in K} \Gamma_k$, the set K is* infinite*. Then there is a renaming ρ, such that $\mathit{match}[\tau, \sigma[\rho]] \to S$ and if $\mathit{res}[\tau, \sigma[\rho]] = \bigotimes_{k \in K} \Gamma_k$, then K is infinite.*

By induction on the structure of the types.

We want to prove that for every judgement $P \triangleright \Gamma$, there exists a environment Δ such that $[\![P \triangleright \Gamma]\!]^{\Delta}$ is defined. We will prove it by induction on the typing rules. However we need a stronger statement for the induction to go through. We prove such a Δ exists that has the following properties

- if $\Delta(x) = \tau$, then $\Gamma(x) = er(\tau)$
- if $\Gamma(x) = \tau$, then there exists τ' such that $\Delta(x) = \tau'$ and $er(\tau') = \tau$.
- if $[\![P \triangleright \Gamma]\!]^{\Delta}$ is defined, for every fresh renaming ρ, $[\![P \triangleright \Gamma]\!]^{\Delta[\rho]}$ is also defined.
- for every type of the form $\bigotimes_{k \in K} \Gamma_k$, the set K is *infinite*.

The proof is trivial for Zero, WeakCl, WeakOut, Res, LIn, LOut, Rout. For Rin, one has just to take care to choose K to be infinite. For the parallel composition, assume $[\![P_1 \triangleright \Gamma_1]\!]^{\Delta_1}$ and $[\![P_2 \triangleright \Gamma_2]\!]^{\Delta_2}$ are defined. First rename all the variables in Δ_1, Δ_2, so that they are disjoint. In this way we can substitute a name of Δ_1 for a name in Δ_2, and Δ_2 would still be well formed.

Then consider a judgement $a : \tau$ in Γ_1 such that there is a matching judgement $a : \sigma$ in Γ_2. Consider the type τ' such that $a : \tau'$ is in Δ_1. Since $er(\tau) = \overline{er(\tau')}$, by Lemma A.6 we find a ρ_a such that $\mathit{match}[\tau, \sigma[\rho_a]] \to S$. For every matching name, we obtain such a renaming. All renamings can be joined to obtain a fresh injective renaming ρ, because no name is involved in two different renamings. Therefore $\Gamma_1 \odot \Gamma_2[\rho]$ is defined.

Proof of Theorem 7.3 The proof is by structural induction on $P \triangleright \Gamma$.

All the cases are rather easy, taking into account that π-calculus terms can perform any fresh α-variant of an action.

For the parallel composition, one has to notice that names that are closed *after* the transition in the π-calculus are closed *before* the transition in NCCS.

References

1. Samy Abbes and Albert Benveniste. Branching cells as local states for event structures and nets: Probabilistic applications. In *Proceedings of 8th FoSSaCS*, volume 3441 of *LNCS*, pages 95–109. Springer, 2005.
2. Paolo Baldan, Andrea Corradini, and Ugo Montanari. Unfolding and event structure semantics for graph grammars. In *Proceedings of 2nd FoSSaCS*, volume 1578 of *LNCS*, pages 73–89. Springer, 1999.
3. Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the π-calculus. In *Procceedingss of TLCA'01*, volume 2044 of *LNCS*, pages 29–45, 2001.
4. Gérard Berry and Pierre-Louis Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20(265–321), 1982.
5. Michele Boreale and Davide Sangiorgi. A fully abstract semantics for causality in the π-calculus. *Acta Inf.*, 35(5):353–400, 1998.
6. Maria Grazia Buscemi and Vladimiro Sassone. High-level petri nets as type theories in the join calculus. In *Proceedings of 4th FOSSACS*, volume 2030 of *LNCS*, pages 104–120. Springer, 2001.
7. Nadia Busi, Maurizio Gabbrielli, and Gianluigi Zavattaro. Replication vs. recursive definitions in channel based calculi. In *Proceedings of 30th ICALP*, volume 2719 of *LNCS*, pages 133–144. Springer, 2003.
8. Nadia Busi and Roberto Gorrieri. A petri net semantics for pi-calculus. In *Proceedings of 6th CONCUR*, pages 145–159, 1995.
9. Gian Luca Cattani and Peter Sewell. Models for name-passing processes: Interleaving and causal. In *Proceedings of 15th LICS*, pages 322–332, 2000.
10. Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. On the consistency of "truly concurrent" operational and denotational semantics (extended abstract). In *Proceedings of 3rd LICS*, pages 133–141, 1988.
11. Pierpaolo Degano and Corrado Priami. Non-interleaving semantics for mobile processes. *Theoretical Computer Science*, 216(1-2):237–270, 1999.
12. Jörg Desel and Javier Esparza. *Free Choice Petri Nets*. Cambridge University Press, 1995.
13. Joost Engelfriet. A multiset semantics for the pi-calculus with replication. *Theoretical Computer Science*, 153(1&2):65–94, 1996.
14. Claudia Faggian and François Maurel. Ludics nets, a game model of concurrent interaction. In *Proceedings of 20th LICS*, pages 376–385, 2005.
15. Pablo Giambiagi, Gerardo Schneider, and Frank D. Valencia. On the expressiveness of infinite behavior and name scoping in process calculi. In *Proceedings of 7th FoSSaCS*, volume 2987 of *LNCS*, pages 226–240. Springer, 2004.
16. Mihaela Herescu and Catuscia Palamidessi. Probabilistic asynchronous π-calculus. In *Proceedings of 3rd FoSSaCS*, volume 1784 of *LNCS*, pages 146–160. Springer, 2000.
17. Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *TCS*, 151, 1995.
18. Lalita Jategaonkar Jagadeesan and Radha Jagadeesan. Causality and true concurrency: A data-flow analysis of the pi-calculus (extended abstract). In *Proceedings of 4th AMAST*, volume 936 of *LNCS*, pages 277–291. Springer, 1995.
19. André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.
20. Gilles Kahn and Gordon D. Plotkin. Concrete domains. *Theoretical Computer Science*, 121(1-2):187–277, 1993.
21. Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the Pi-Calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, 1999.
22. Jim Laird. A game semantics of the asynchronous π-calculus. In *Proceedings of 16th CONCUR*, pages 51–65, 2005.
23. Antoni Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *LNCS*, pages 279–324. Springer, 1986.

24. Paul-André Melliès. Asynchronous games 4: A fully complete model of propositional linear logic. In *Proceedings of 20th LICS*, pages 386–395, 2005.
25. Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.
26. Robin Milner. *Communicating and Mobile Systems: The Pi Calculus*. Cambrige University Press, 1999.
27. Ugo Montanari and Marco Pistore. Concurrent semantics for the π-calculus. *Electr. Notes Theor. Comput. Sci.*, 1, 1995.
28. Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13(1):85–108, 1981.
29. Grzegorz Rozenberg and Joost Engelfriet. Elementary net systems. In *Dagstuhl Lecturs on Petri Nets*, volume 1491 of *LNCS*, pages 12–121. Springer, 1996.
30. Grzegorz Rozenberg and P.S. Thiagarajan. Petri nets: Basic notions, structure, behaviour. In *Current Trends in Concurrency*, volume 224 of *LNCS*, pages 585–668. Springer, 1986.
31. Davide Sangiorgi. Internal mobility and agent passing calculi. In *Proc. ICALP'95*, 1995.
32. Daniele Varacca, Hagen Völzer, and Glynn Winskel. Probabilistic event structures and domains. In *Proceedings of 15th CONCUR*, volume 3170 of *LNCS*, pages 481–496. Springer, 2004.
33. Vasco Vasconcelos. Typed concurrent objects. In *Proc. ECOOP'94*, 1994.
34. Glynn Winskel. *Events in Computation*. Ph.D. thesis, Dept. of Computer Science, University of Edinburgh, 1980.
35. Glynn Winskel. Event structure semantics for CCS and related languages. In *Proceedings of 9th ICALP*, volume 140 of *LNCS*, pages 561–576. Springer, 1982.
36. Glynn Winskel. Event structures. In *Advances in Petri Nets 1986, Part II; Proceedings of an Advanced Course*, volume 255 of *LNCS*, pages 325–392. Springer, 1987.
37. Glynn Winskel. Name generation and linearity. In *Proceedings of 20th LICS*, pages 301–310. IEEE Computer Society, 2005.
38. Glynn Winskel. Relations in concurrency. In *Proceedings of 20th LICS*, pages 2–11. IEEE Computer Society, 2005.
39. Glynn Winskel and Mogens Nielsen. Models for concurrency. In *Handbook of logic in Computer Science*, volume 4. Clarendon Press, 1995.
40. Nobuko Yoshida. Type-based liveness guarantee in the presence of nontermination and nondeterminism. Technical Report 2002-20, MCS Technical Report, University of Leicester, 2002.
41. Nobuko Yoshida, Martin Berger, and Kohei Honda. Strong Normalisation in the π-Calculus. In *Proceedings of LICS'01*, pages 311–322. IEEE, 2001. The full version in *Journal of Inf. & Comp.*., 191 (2004) 145–202, Elsevier.