INFORMATION AND COMPUTATION 114, 247–314 (1994)

# Flow Models of Distributed Computations: Three Equivalent Semantics for CCS*

GÉRARD BOUDOL AND ILARIA CASTELLANI

*INRIA Sophia-Antipolis, 06560 Valbonne Cedex, France*

We introduce three notions of computation for processes described as CCS (Calculus of Communicating Systems) terms. The first one uses an adaptation of the equivalence by permutations of Berry and Lévy. In this setting, a computation is an equivalence class of sequences of transitions, up to the permutation of independent steps. The second notion of computation is given by means of an interpretation of CCS into a new class of event structures, the flow event structures. This can be seen as a reformulation of Winskel's semantics for CCS by means of stable event structures. Here a computation is a configuration of an event structure. Finally, our third notion of computation is determined by an interpretation of CCS terms as Petri nets, and more precisely as flow nets. Here a computation is a set of events that are firable in sequence in the net. We then show that these three computation interpretations of CCS coincide, in the sense that for a given term, the three domains of computations are isomorphic. To this end we use an intermediary transition system for CCS, where the past is recorded; this appears to be a system of "trace computations," which provides another means to define the same abstract domain of computations. 

## 1. INTRODUCTION

This work is concerned with non-interleaving models for concurrency, sometimes qualified as "true-concurrency" models because they take the notion of concurrency—or the complementary notion of causality—as fundamental. Typical non-interleaving models are Petri nets [46], event structures [43, 58], and Mazurkiewicz traces [38, 39].

The other and standard approach is called the *interleaving* approach because it simulates concurrency with arbitrary interleaving, thus reducing the notion of concurrency to those of non-determinism and sequentiality: as a result, models have fewer primitive notions and are simpler to deal with. This is why research on the semantics of concurrency initially concentrated on interleaving models, at least outside the Petri net community. These models, particularly after the appearance of Milner's Calculus of Communicating Systems (CCS) [40] and Plotkin's method of Structural Operational Semantics (SOS) [47], have been mostly developed in

* This work has been partly supported by ESPRIT Basic Research Action 3011 CEDISYS.



0890-5401/94 $6.00



643/114/2-6

relation to CCS or similar algebraic process languages (like TCSP [11], ACP [2] etc.), according to the following two-step schema:

(1) First, terms are described operationally—using SOS rules—as *labelled transition systems*, evolving from one state to the other via successive transitions: each transition is labelled by an atomic action, representing an interaction with the environment or an internal computational task.

(2) Then, transition systems are factored by *behavioural equivalences* (or preorders) to yield a more abstract model. Often axiomatic theories are provided for the behavioural equivalences, as well as logical characterisations. Presentations of various such equivalences and related theories may be found, e.g., in [12, 22, 34, 41].

Note that in this two-step semantics, in spite of the variety of behavioural relations and resulting models, there is a general agreement about the first step: the basic operational model is almost always that of labelled transition systems.

Undoubtedly, the interleaving assumption has been useful for the construction of elegant theories of concurrency, with related specification and verification methods. However, this does not mean that concurrency itself is not an essential phenomenon: indeed, there are properties of concurrent systems which cannot be specified without a clear distinction between concurrency and non-determinism; see, e.g., [36] for the suitability of non-interleaving models as regards the treatment of fairness.

In the field of true concurrency, we find again the two-level schema for semantics, but we have here a variety of basic operational models. We have already mentioned Petri nets and event structures, but several other models have been proposed, essentially generalisations of labelled transition systems: transition systems with non-atomic labels, like concurrent histories [17, 18] and pomset transition systems [6], where labels are pomsets, i.e., partially ordered multisets [48]; transition systems with structured states, like distributed transition systems (see [13], but also the different notion of [18]), and transition systems satisfying a set of axioms, like asynchronous transition systems [51, 1], concurrent transition systems [52] and elementary transition systems [44]. An intensional tree model has also been proposed, the causal trees of [16], where arcs are provided with back pointers to their causes. It should be stressed that all these concrete models, here as in the interleaving approach, are operational in nature.

In this paper we shall only be interested in concrete non-interleaving models; that is, in the first level of the semantics. In particular, we shall not be concerned with equivalence relations respecting concurrency, which have been the subject of much recent research: we will just mention, among

others, the work by Van Glabbeek and Vaandrager on equivalence notions for Petri nets [26], and the papers by Van Glabbeek and Goltz on various equivalences for event structures [27, 28].

Because of the variety of basic models, and important task in the area of true concurrency is that of classification and comparison of models. A major contribution in this sense has been provided by Winskel: indeed, event structures were first conceived by Nielsen, Plotkin, and Winskel [43] to relate Petri nets with the now classical theory of domains à la Scott. This unifying concern has remained present in most of Winskel's subsequent works on event structures and the theory of concurrency. See for example [57, 59], where different models for parallel computations are specified and related in the framework of category theory. More recently, the comparison and unification of models has also been a main concern of Montanari and Ferrari [23, 24]. For the comparison of specific models we could mention Best and Devillers [4], Thiagarajan [54], and Rozoy [50]. When comparing models, a general requirement for considering them *equivalent* is that the resulting *domains of computations* be isomorphic. This is the abstract criterion used in the above-mentioned works [4, 54, 50]. A more concrete criterion for equivalence of models is that they generate the same *families of computations*—that is, computations consisting of the same sets of events. Indeed, this concrete criterion preserves the operational meaning of events, and may be the right one to use when one is interested in a notion of implementation of a model into an equivalent one.

Another major issue in true concurrency is the interpretation of algebraic languages like CCS or TCSP in non-interleaving models like Petri nets or event structures, and the study of the relations between different such interpretations. Petri nets have been used to give a "truly concurrent" semantics for CCS-like languages, for instance by Goltz [29–31], Winskel [57], Van Glabbeek and Vaandrager [26], Degano *et al.* [20], Olderog [45], and Taubner [53]. Similarly, event structure interpretations have been proposed by Winskel [56, 58], Goltz and Loogen [32], Degano *et al.* [19], and ourselves [6–8]. Now, in order to prove the equivalence of different such interpretations for a language, one has to show that they yield equivalent representations—in one sense or the other—for any term of the language.

The aim of this paper is to contribute to this area of "comparative non-interleaving semantics" for algebraic languages, by establishing the equivalence of three operational models for CCS, in a rather concrete sense. The three interpretations we will consider may be summarized as follows:

(i) A semantics by *permutations of concurrent transitions*, based on a description of processes as proved transition systems: these are labelled

transition systems where the labels are *proofs* of transitions, from which a concurrency relation between transitions may be deduced.

(ii) A semantics by means of *flow event structures*, which is a variation of Winskel's semantics by stable event structures.

(iii) For finite terms, that is, more accurately, for fixpoint-free terms, a semantics by means of *flow nets*—a class of Petri nets introduced in [10]—which coincides with the "standard" net semantics for CCS, as described by a number of authors [30, 31, 26, 20, 45].

The first two interpretations, (i) and (ii), were already presented in [8]: interpretation (i), which is somewhat unusual, will be described in more detail later in this introduction. The third interpretation, (iii), is essentially a recasting of the operational Petri net semantics given by Degano *et al.* in [20] (see also Olderog [45]). In particular we take from [20] the idea of defining the places of a net syntactically. For simplicity we restrict ourselves, for the net interpretation for the full calculus, by introducing "infinite choice places." On the other hand, it is very easy to deal with the fixpoint construct in the two other models. In particular it is easy for the flow event structure interpretation because this is based on a syntactic notion of event, and not on an abstract continuous algebra of event structures.

In fact, the particularity of our approach is that all interpretations are given *syntactically*: thus, for example, the labels of proved transition systems are terms representing the proofs of transitions in the inference system of CCS. Similiarly, events in event structures and nets have syntactic names which uniquely identify them as occurrences (or pairs of occurences, for a communication) in the syntactic tree of term. Moreover, the syntax is the same for the three models: the events generated by the proved transition system for a given term have the same names as the events of the corresponding flow event structure and flow net.

This similarity enables us to carry out a precise comparison of the three models, and to establish their equivalence in a very strong sense: we will see that the flow event structure and the flow net interpretations agree in that they generate exactly the same families of computations, hence the same domains of computations for any (finite) CCS term. On the other hand, this domain is isomorphic—this is a looser correspondence—to the domain of computations given by the permutation semantics, that is, equivalence classes of sequences ordered by prefix. The connection between flow event structure and permutation semantics was already stated in [8], but the proof was not included.

We assume some familiarity with event structures and Petri nets, so we will not discuss these models here: of course they will be defined in the sections where they are used, and the paper is self-contained in this sense. On

the other hand, we would like to introduce briefly the semantics by permutations, which, although very close in spirit to Mazurkiewicz's theory of traces [38], comes to us here from a different background. The notion of equivalence by permutations that we use here is rather general, and may be applied to various computational systems. This equivalence was first elaborated by Lévy for the $\lambda$-calculus [37], and then used for recursive program schemes in [3]. It was further extended to deterministic term rewriting systems by Huet and Lévy in [35], and to non-deterministic ones by Boudol in [5]. An abstract setting is presented by Stark in [52]. Let us shortly explain the idea.

A computational system evolves by elementary computations $s \to s'$ from one state to the other. Examples of state changes are transitions of a machine, $\beta$-reductions of $\lambda$-terms and rewritings in a term rewriting system. To be able to reason about such transitions—e.g., to show a Church–Rosser property—we need a concrete indication of what has been performed and *where* it has happened. We thus have to deal with *labelled* transitions $s \xrightarrow{w} s'$, where $w$ denotes a specific *occurrence* of some action. Now if two transitions from a given state, $s \xrightarrow{u} s_0$ and $s \xrightarrow{v} s_1$, are compatible, or concurrent, it should be possible to perform them in any order without affecting the result. In other words, we should be able to define what remains of one move after the other, the *residual* of $v$ by $u$, noted $v/u$, in such a way that $s_0 \xrightarrow{v/u} s'$ and $s_1 \xrightarrow{u/v} s'$, for some state $s'$. This is known as the *diamond property* for concurrent moves. This property induces an equivalence on sequences of transitions: two sequences are equivalent if they are the same up to permuation of compatible transitions, typically

$$s \xrightarrow{u} s_0 \xrightarrow{v/u} s' \simeq s \xrightarrow{v} s_1 \xrightarrow{u/v} s'.$$

This is the essence of Berry and Lévy's *equivalence by permutations* for sequences of transitions. In any case, this equivalence allows one to associate with each state a complete partial order of computations. These computations are equivalence classes of sequences of transitions, ordered by the prefix ordering modulo permutations.

The first semantics for CCS that we consider in this paper is precisely a semantics by permutations. As we said, to be able to define such a semantics, we need a notion of occurrence of action. The usual operational semantics of CCS describes processes as performing transitions labelled by actions. These transitions are inferred using a system of structural rules. What we shall take here as the occurrence of action associated with a transition is the proof of that transition in the given system of rules. Each proof identifies uniquely one transition, and we use this information to define a relation of *concurrency* on (proved) transitions, and a notion of

residual of a (proved) transition by a concurrent one. A diamond property for concurrent proved transitions may then be proved; this result is much simpler in CCS than in $\lambda$-calculus or term rewriting systems, since the proof of a transition cannot be duplicated or deleted by a concurrent one.

We may then move on to define equivalence by permutations on sequences of proved transitions. The computations of a CCS term in this model are thus equivalence classes of sequences of proved transitions. Now any such computation may be represented as a one step transition, labelled by a *pomset* [48] of actions. We thus obtain a partial order semantics for CCS, which is directly derived from its usual operational semantics. Also, the permutation semantics given here may be seen as an extension of our pomset semantics of [6, 7], where communication and restriction were not considered.

As regards the semantics of concurrency, the idea of a computation as an equivalence class of sequences was first formalised by Mazurkiewicz in his theory of traces [38, 39]. We recall that a trace is an equivalence class of sequences of actions up to independent actions. However, actions in Mazurkiewicz's setting do not have the same meaning as in CCS; e.g., an action cannot be concurrent with itself, as it happens in the CCS process $(a.\mathbf{nil} \parallel a.\mathbf{nil})$. In fact, actions in a trace are really *events*, and in this case the notion of residual is very simple: the residual of an event by a concurrent one is just the event itself. We shall see that this is not the case for our proved transitions: a proved transition may be modified by the occurrence of a concurrent one. Thus we need a more general notion of residual. Another difference between our approach and trace theory is that the concurrency relation is not given from the start here: instead, our starting point is the operational semantics, and we show how the concurrency relation may be extracted from it using the structure of labels.

Let us now briefly sketch how we establish the correspondence of our three models. The proved transition system generates at each step what we call proofs for a term $p$, which are initial occurrences of actions in $p$. This is a simple operational model, sufficient for the purpose of deriving a concurrency relation and a permutation semantics. On the other hand the two flow models—flow event structures and flow nests—give global representations of terms, modelling occurrences of actions as events, which are occurrences of arbitrary depth in a term.

Hence, in order to compare more easily the permutation semantics with the flow models, we will introduce an auxiliary transition system, where we keep track of the whole *past* of a transition, simply by recording in the states and actions the guards that have been passed and the choices that have been made. We call this system an *event transition system*, or simply event system, since its transitions are labelled by events. In this system the computations are again equivalence classes of sequences of transitions, up

to permutations. However, we show that these computations are fully determined by the set of event labelling transition sequences. Then the event system will act as an intermediate between the flow models and the proved transition system. More precisely, we will show that, for any term $p$ of CCS:

— the configurations of the flow event structure associated with $p$ are exactly the computations of $p$ in the event system, that is, the sets of events occurring in an event transition sequence from $p$;

— if $p$ is a "finite" term, then the event system and the flow net associated with $p$ are *isomorphic* transition systems: thus, in particular, they generate the same computations, i.e., sets of events occurring in the transition sequences.

Thus the three event-based—event system, flow event structures, flow nets (for finite terms)—determine the *same families of computations.* On the other hand, we will prove that, for any term $p$ of CCS:

— the event system and the proved transition system associated with $p$ yield *isomorphic domains of computations.*

In conclusion, we will have shown that the two flow models are *concretely* equivalent, while they are both *abstractly* equivalent to the permutation semantics. The most substantial part of our result is the study of the event system and its comparison with the flow event structure interpretation. We shall see that in the event equivalence classes of sequences are *trace computations* in the sense of [38, 39]. In fact, it may be shown—and this is an interesting connection—that the event system is a *labelled asynchronous system* in the sense of [1, 51], and it is known from [1, 36] that such systems generate trace computations. Moreover, our event system for CCS is an example of labelled asynchronous system determined by the structure of CCS terms, and thus gives a positive answer to a question raised by Kwiatkowska in her thesis [36, Sect. 7.2].

We conclude this introduction with a brief outline of the paper. In Section 2 we introduce the language CCS and present our proved transition system semantics for it. We show, in beginning of Section 3, how this concrete description of processes may be used to derive a relation of concurrency between transitions, in a purely syntactic way. The remainder of Section 3 is devoted to the permutation semantics: we give the diamond property for concurrent transitions and define the equivalence by permutations on sequences of transitions. In Sections 4 and 5 we describe the flow event structure and flow net interpretations for CCS. Finally, in Section 6, the various semantics are brought together, and we show how they relate to each other. This is also where the event system, our auxiliary model, is introduced and studied.

## 2. CCS

### 2.1. *Terms and Transitions*

We start by introducing the syntax of CCS. We assume some familiarity with this calculus, referring the reader to Milner's work [40–42] and to [33] for a general introduction to CCS and related algebraic languages. We should point out here that we shall not consider the relabelling operator, although it would not introduce any difficulty.

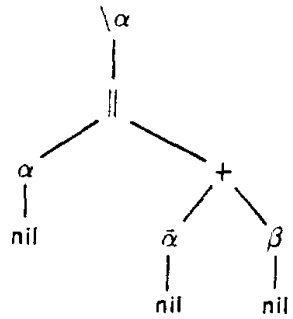
As in [40], we let $\Delta$ be a fixed set of *names*. We use $\alpha, \beta, \ldots$ to stand for names. We assume a set $\bar{\Delta}$ of *co-names* (complementary names), disjoint from $\Delta$ and in bijection with it: the co-name of $\alpha$ is $\bar{\alpha}$, while its name is $\mathbf{nm}(\bar{\alpha}) = \mathbf{nm}(\alpha) = \alpha$. Then $\Lambda = \Delta \cup \bar{\Delta}$ is the set of *labels*. We shall use $\lambda$ to range over $\Lambda$, and extend the bijection so that $\bar{\bar{\lambda}} = \lambda$. As usual the set $A$ of CCS *actions* is $A = \Lambda \cup \{\tau\}$, where $\tau$ is a new symbol, not in $\Lambda$; by convention the name of $\tau$ is $\tau$. We use $a, b, c, \ldots$ to range over $A$.

To define the terms of the language, we assume a denumerable set of process variables, disjoint from $A$. We use $x, y, z, \ldots$ to range over these variables. The set of CCS terms we will study is then given by the grammar

$$p ::= \mathbf{nil} \mid x \mid \mu x.p \mid a.p \mid (p \parallel p') \mid (p + p') \mid p\backslash\alpha.$$

We shall use $p, q, r, \ldots$ to range over terms. We shall call FCCS the calculus of "finite" terms, or more accurately of *fixpoint-free* terms, that are written without using process variables or the fixpoint construct $\mu x.p$. As is standard, the fixpoint construct binds the defined process variable, and substituting $q$ for $x$ in $p$, resulting in $p[q/x]$, may require renaming the bound variables of $p$ to avoid captures.

Although we shall not elaborate formally on this representation, we should point out that the terms of FCCS may be viewed as finite *trees*, with parallel composition and sum as binary node constructors, action and restriction as unary ones, with a parameter in $A$ and $\Delta$ respectively, and **nil** as a constant. For instance, the term $r = (\alpha.\mathbf{nil} \parallel (\bar{\alpha}.\mathbf{nil} + \beta.\mathbf{nil}))\backslash\alpha$ may be identified with the syntactic tree



Our three semantics for CCS will be strongly based on such syntactic representation for terms. We recall first the standard operational semantics of CCS. This is given by means of *inference rules*, allowing one to prove *transitions* of the form $p \xrightarrow{a} p'$ for closed terms $p$. The transitions of a term $p$ are exactly those which can be proved in the following system of rules:

| | |
|---|---|
| *action* | $\vdash a.p \xrightarrow{a} p$ |
| *parallel composition* 1 | $p \xrightarrow{a} p' \vdash (p \parallel q) \xrightarrow{a} (p' \parallel q)$ |
| *parallel composition* 2 | $q \xrightarrow{b} q' \vdash (p \parallel q) \xrightarrow{b} (p \parallel q')$ |
| *communication* | $p \xrightarrow{\lambda} p', q \xrightarrow{\bar{\lambda}} q' \vdash (p \parallel q) \xrightarrow{\tau} (p' \parallel q')$ |
| *sum* 1 | $p \xrightarrow{a} p' \vdash (p+q) \xrightarrow{a} p'$ |
| *sum* 2 | $q \xrightarrow{b} q' \vdash (p+q) \xrightarrow{b} q'$ |
| *restriction* | $p \xrightarrow{a} p', \mathbf{nm}(a) \neq \alpha \vdash (p\backslash\alpha) \xrightarrow{a} (p'\backslash\alpha)$ |
| *fixpoint* | $p[\mu x.p/x] \xrightarrow{a} p' \vdash \mu x.p \xrightarrow{a} p'$ |

This set of rules is usually regarded as defining an *interleaving semantics* for CCS. Indeed, if we except the possibility of communication, the above rules for parallel composition describe $\parallel$ as an interleaving operator, allowing the components to move in any order, but no together. The reader is referred to [40, 42] for a full account of CCS and the related interleaving theory.

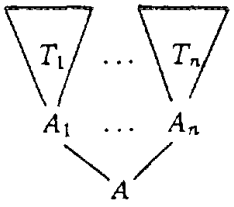
Note that the semantics of a term—the set of transitions which can be referred for it—contains no trace of the inference mechanism itself. We want now to show that, if we keep track of the proofs of transitions, we can extract much more information from the same set of rules. In particular, we will be able to derive a *non-interleaving semantics* for the language, without having to depart from its basic operational semantics.

## 2.2. *The Proved Transition System*

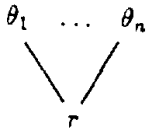
We shall now introduce our *proved transition system* for CCS, where transitions are labelled by their proofs. Let us first formalize the idea of *proof* in the inference system of CCS. In general, in an inference system one has rules of the form

$$r : \frac{A_1, \ldots, A_n}{A} \qquad (*)$$

to deduce an assertion from a finite set of assertions. Such rules generate proof trees of the form




Here $T_1, ..., T_n$ are proof of trees for $A_1, ..., A_n$, while the whole tree is a proof for $A$. Alternatively, one may represent the structure of a proof as a tree $\theta$ whose nodes are labelled by the *rules* which have been used in the derivation. It is not difficult to see that such a tree is isomorphic to the proof tree itself. For example, if $\theta_1, ..., \theta_n$ represent the (rule labelled) trees for $A_1, ..., A_n$ and the last step of derivation consists in applying rule $r$ with premisses $A_1, ..., A_n$, the whole proof may be represented by the tree




Since each rule has a fixed finite number of premisses, any tree of this kind may be denoted by a term $r(\theta_1, ..., \theta_n)$, which we call a *proof term*—or simply a proof—in the following. This is the kind of notation we will use for proofs of transitions in CCS.

Our next step will be to decorate assertions with their proofs, and consider *proved assertions* $\theta : A$, where $\theta$ is a proof of $A$. To manipulate proved assertions, we will introduce, for each rule $(*)$ of the inference system, a decorated rule

$$\frac{x_1 : A_1, ..., x_n : A_n}{r(x_1, ..., x_n) : A} \qquad (**)$$

Such rules will build up proofs of assertions as the inference process goes on: whenever a rule is applied, the name $r$ of the rule is recorded in the resulting proof term.

In the inference system of CCS, assertions are transitions of the form $p \xrightarrow{a} p'$. The proof terms $\theta$ will be built with the following symbols, one for each inference rule of CCS—except for the meta-rule for fixpoints:

$a$ for the action rule (note that there is a separate rule for each action $a$)

$\|_0$ for the rule inferring a transition at the left of a parallel composition

$\|_1$ for the rule inferring a transition at the right of a parallel composition

$(\ ,\ )$ for the communication rule

$+_0$ for the rule inferring a transition at the left of a sum

$+_1$ for the rule inferring a transition at the right of a sum

$\backslash_\alpha$ for the rule for restriction (one for each name $\alpha$)

As we said, these rule names will be used as proof constructors. Each constructor takes as many parameters as are the hypotheses of the corresponding rule. For example the constructor $a$ takes no parameters, while $(\ ,\ )$ takes two. The syntax for proofs of CCS transitions is thus given by the grammar, where $a \in Act$ and $\alpha \in \Delta$:

$$\theta ::= a \mid \|_0(\theta) \mid \|_1(\theta) \mid (\theta, \theta') \mid +_0(\theta) \mid +_1(\theta) \mid \backslash_\alpha(\theta).$$

We shall denote by $\Theta$ the set of proof terms. We will mostly use the notations $\|_i \theta$, $+_i \theta$, $\backslash_\alpha \theta$ instead of $\|_i(\theta)$, $+_i(\theta)$, $\backslash_\alpha(\theta)$. Our proof terms are similar to the "synchronisation terms" of Degano *et al.* [17], apart from the fact that we also record the choices. Note that although we call them proof terms, the $\theta$'s do not always represent proper proofs; for instance, $\backslash_\alpha \alpha$ and $(\alpha, \beta)$ do not correspond to any CCS transition. However, this will not cause any problem, since such proof of terms will no actually appear in a transition. We shall use a *labelling* mapping $\ell : \Theta \to Act$, assigning to a proof $\theta$ the action of which $\theta$ is an occurence:

$$\ell(a) = a$$

$$\ell(\|_i \theta) = \ell(\theta) = \ell(+_i \theta) = \ell(\backslash_\alpha \theta)$$

$$\ell((\theta, \theta')) = \tau.$$

Again note that defining $\ell(\alpha, \beta) = \tau$ or $\ell(\backslash_\alpha \alpha) = \alpha$ does not cause any problem. We shall now bring together proofs and transitions. Usually one denotes $\theta : A$ the fact that $\theta$ is a proof of the assertion $A$. Here we will use, instead of $\theta : p \xrightarrow{a} p'$, the notation $p \xrightarrow{a,\theta} p'$, which is more convenient for dealing with sequences of transitions. Note that for such a transition $p \xrightarrow{a,\theta} p'$, we will always have $a = \ell(\theta)$ and thus we may omit the action $a$. We will then adopt the simpler notation $p \xrightarrow{\theta} p'$, to be interpreted follows: $\theta$ is a proof of the fact that $p$ performs the action $\ell(\theta)$ and becomes $p'$ in doing so. We will call $p \xrightarrow{\theta} p'$ a *proved transition* for $p$. These transitions are given by the following rules:

A. $\vdash a.p \xrightarrow{a} p$

P0. $p \xrightarrow{\theta} p' \vdash (p \parallel q) \xrightarrow{\|_0 \theta} (p' \parallel q)$

P1. $q \xrightarrow{\theta} q' \vdash (p \parallel q) \xrightarrow{\|_1 \theta} (p \parallel q')$

C. $p \xrightarrow{\theta} p', q \xrightarrow{\theta'} q', \overline{\ell(\theta)} = \ell(\theta') \vdash (p \parallel q) \xrightarrow{(\theta, \theta')} (p' \parallel q')$

S0. $p \xrightarrow{\theta} p' \vdash (p + q) \xrightarrow{+_0 \theta} p'$

S1. $q \xrightarrow{\theta} q' \vdash (p + q) \xrightarrow{+_1 \theta} q'$

R. $p \xrightarrow{\theta} p', \mathbf{nm}(\ell(\theta)) \neq \alpha \vdash (p \backslash \alpha) \xrightarrow{\backslash_\alpha \theta} (p' \backslash \alpha)$

FIX. $p[\mu x.\, p/x] \xrightarrow{\theta} p' \vdash \mu x.\, p \xrightarrow{\theta} p'$

It should be clear that if we drop the proof terms—and retain their labels—we obtain the original rules of CCS. Hence the usual semantics of CCS is still explictly present in the proved transition system.

Let us see an example. Take again the term $r = (\alpha.\mathbf{nil} \parallel (\bar{\alpha}.\mathbf{nil} + \beta.\mathbf{nil}))\backslash\alpha$. Let us derive the proved transition corresponding to the communication on $\alpha, \bar{\alpha}$. To illustrate our technique we picture the whole proof tree for transition:

$$\dfrac{\dfrac{\dfrac{}{\alpha.\mathbf{nil} \xrightarrow{\alpha} \mathbf{nil}} \quad \dfrac{\dfrac{}{\bar{\alpha}.\mathbf{nil} \xrightarrow{\bar{\alpha}} \mathbf{nil}}}{(\bar{\alpha}.\mathbf{nil} + \beta.\mathbf{nil}) \xrightarrow{+_0 \bar{\alpha}} \mathbf{nil}}}{(\alpha.\mathbf{nil} \parallel (\bar{\alpha}.\mathbf{nil} + \beta.\mathbf{nil})) \xrightarrow{(\alpha,\, +_0 \bar{\alpha})} (\mathbf{nil} \parallel \mathbf{nil})}}{(\alpha.\mathbf{nil} \parallel (\bar{\alpha}.\mathbf{nil} + \beta.\mathbf{nil}))\backslash\alpha \xrightarrow{\backslash_\alpha(\alpha,\, +_0 \bar{\alpha})} (\mathbf{nil} \parallel \mathbf{nil})\backslash\alpha}$$

One could also see that if we let $(a^\omega \parallel b^\omega) = (\mu x.a.x \parallel \mu y.b.y)$ then the following are proved transitions:

$$(a^\omega \parallel b^\omega) \xrightarrow{\parallel_0 a} (a^\omega \parallel b^\omega) \qquad \text{and} \qquad (a^\omega \parallel b^\omega) \xrightarrow{\parallel_1 b} (a^\omega \parallel b^\omega).$$

Decorating the transitions with their proofs provides us with "maximal" concrete information. As a matter of fact, our proof terms are closely related to syntactic trees. If we look back at the syntactic tree for the term $r$, we may notice that the proof $\theta$ of a transition specifies a path in the tree. In the simple case were $\ell(\theta) = a$, this path leads to the subterm, $a.q$ which performs the action $a$. However, if the action is a communication as in the example above, the proof will be a path to a pair of complementary subterms $\lambda.q$ and $\bar{\lambda}.q'$.

To sum up, the proof of a transition $\xrightarrow{a}$ for a process $p$ is an exact indication of how we get the action $a$ from $p$. As a matter of fact, the proof of a transition identifies uniquely that transition: the new system of proved transitions is *deterministic*, that is,

$$p \xrightarrow{\theta} p' \quad \text{and} \quad p \xrightarrow{\theta} p'' \Rightarrow p' = p''.$$

Now the concrete information contained in proofs may be weakened in various ways to obtain more abstract semantics.

Here we shall use this concrete information to define a *concurrency relation* on transitions. This will enable us to define an equivalence by permutations on sequences of transitions, and thereby retrieve a partial order semantics for CCS. A similar but somewhat dual approach was taken in [17] by Degano *et al.*, who use a concrete transition system for CCS to extract a causality relation on transitions; this allows them to reconstruct a partial order transition from any sequence of concrete transitions.

## 3. Permutation of Transitions

We shall now introduce a notion of *concurrency* on proved transitions. Roughly speaking, two transitions are concurrent if they occur on different sides of a parallel composition, whereas they are in *conflict* if they occur on different sides of a sum. However some complications arise from communication, which may introduce new conflicts. Typically, two communications will be in conflict if they share one component. Conversely, they will be concurrent if they are pairwise concurent—i.e., they have concurrent components.

The relation of concurrency on proved transitions is induced from a relation of concurrency on proof terms, in notation $\theta \frown \theta'$, which we define now. The concurrency relation $\frown$ on proof terms is the least symmetric relation that satisfies the following clauses (for any $\theta, \theta', \theta'' \in \Theta$):

$$\text{(A1)} \qquad \|_0\,\theta \frown \|_1\,\theta'$$

$$\text{(A2)} \qquad \theta \frown \theta' \quad \Rightarrow \quad \begin{cases} \|_0\,\theta \frown (\theta', \theta'') \\ \|_1\,\theta \frown (\theta'', \theta') \end{cases}$$

$$\text{(A3)} \qquad \theta \frown \theta' \quad \Rightarrow \quad \begin{cases} \|_i\,\theta \frown \|_i\,\theta' \\ +_i\,\theta \frown +_i\,\theta' \\ \backslash_\alpha\,\theta \frown \backslash_\alpha\,\theta' \end{cases}$$

$$\text{(A4)} \qquad \theta_0 \frown \theta_0' \quad \text{and} \quad \theta_1 \frown \theta_1' \quad \Rightarrow \quad (\theta_0, \theta_1) \frown (\theta_0', \theta_1')$$

The essential rule is (A1). Rules (A3) and (A4) express the compatibility of $\frown$ with the various proof constructors. Rule (A2) also states a kind of compatibility of $\frown$ w.r.t. the constructors $\|_i$, since a proof $(\theta', \theta'')$ stands for the co-occurrence of $\|_0\,\theta'$ and $\|_1\,\theta''$. We may now define the concurrency relation on proved transitions.

DEFINITION (Concurrent Transitions). Let $t_0 = p \xrightarrow{\theta_0} p_0$ and $t_1 = p \xrightarrow{\theta_1} p_1$ be two proved transitions for the same CCS term $p$. The transitions are concurrent, in notation $t_0 \frown t_1$, if and only if $\theta_0 \frown \theta_1$.

Note that by definition the concurrency relation between transitions is symmetric and irreflexive: it is easy to check that $\theta \frown \theta \Rightarrow \theta \neq \theta'$. Let us see a simple example: two possible initial transitions of the term $(a^\omega \parallel b^\omega)$ are concurrent, that is,

$$(a^\omega \parallel b^\omega) \xrightarrow{\|_0 a} (a^\omega \parallel b^\omega) \frown (a^\omega \parallel b^\omega) \xrightarrow{\|_1 b} (a^\omega \parallel b^\omega).$$

To further illustrate the application of clauses (A1)–(A4), we examine the relations between some transitions of the term $p = (\alpha \parallel \alpha) \parallel \bar{\alpha}$ (for simplicity we shall omit the trailing **nil**'s in CCS terms). The two $\alpha$-transitions are concurrent because $\|_0 \|_0 \alpha \frown \|_0 \|_1 \alpha$, by (A1) and (A3). The first $\alpha$-transition and the communication on $\alpha, \bar{\alpha}$ of the remaining two components are also concurrent, because $\|_0 \|_0 \alpha \frown (\|_1 \alpha, \bar{\alpha})$ is an instance of (A2), with $\theta = \|_0 \alpha$ and $\theta' = \|_1 \alpha$. On the other hand $\|_0 \|_0 \alpha \not\frown (\|_0 \alpha, \bar{\alpha})$, since $\|_0 \alpha \not\frown \|_0 \alpha$ and thus (A2) does not apply.

The complementary relation of conflict could be formalized in a similar way (this will be done in the more general setting of event transitions; see Sections 4 and 6). For example we have a conflict between the two communications, since the two transitions

$$p \xrightarrow{(\|_0 \alpha, \bar{\alpha})} (\mathbf{nil} \parallel \alpha) \parallel \mathbf{nil}, \qquad p \xrightarrow{(\|_1 \alpha, \bar{\alpha})} (\alpha \parallel \mathbf{nil}) \parallel \mathbf{nil}$$

share the same "sub-transition" $\|_1 \bar{\alpha}$. Another case of conflict occurs in $r = (\alpha \| (\bar{\alpha} + \beta))\backslash\alpha$, considered earlier. Here the two transitions

$$r \xrightarrow{\backslash\alpha \|_1 + _1 \beta} (\alpha \parallel \mathbf{nil})\backslash\alpha, \qquad r \xrightarrow{\backslash\alpha (\alpha, + _0 \bar{\alpha})} (\mathbf{nil} \parallel \mathbf{nil})\backslash\alpha$$

are in conflict because they made two different choices at the subterm $(\bar{\alpha} + \beta)$.

The intuition about concurrent transitions is that they are *compatible* and may be executed in any order without affecting the result. However, the proof of a transition may be affected by the occurrence transition, because of non-deterministic choices: when a parallel composition is placed in a sum-context, the choice may be made by one of the parallel components, and does not have to be solved again by the other component. This phenomenon, known as "symmetric confusion", will be made clearer by an example below. Hence we need to define the *residual* $t/t'$ of a proved transition $t$ by a concurrent one $t'$, namely what is left of the transition $t$ after $t'$. We define first the residual $\theta/\theta'$ of a proof term by a

concurrent one. For any concurrent proofs $\theta, \theta'$, the residual of $\theta$ after $\theta'$, noted $\theta/\theta'$, is defined by

$$i \neq j \quad \Rightarrow \quad \|_i \theta / \|_j \theta' = \|_i \theta$$

$$\theta \frown \theta' \quad \Rightarrow \quad \begin{cases} \|_0 \theta/(\theta', \theta'') = \|_0 (\theta/\theta') & \text{and} \quad (\theta', \theta'')/\|_0 \theta = (\theta'/\theta, \theta'') \\ \|_1 \theta/(\theta'', \theta') = \|_1 (\theta/\theta') & \text{and} \quad (\theta'', \theta')/\|_1 \theta = (\theta'', \theta'/\theta) \end{cases}$$

$$\theta \frown \theta' \quad \Rightarrow \quad \begin{cases} \|_i \theta / \|_i \theta' = \|_i (\theta/\theta') \\ +_i \theta / +_i \theta' = \theta/\theta' \\ \backslash_\alpha \theta / \backslash_\alpha \theta' = \backslash_\alpha (\theta/\theta') \end{cases}$$

$$\theta_0 \frown \theta_0' \quad \text{and} \quad \theta_1 \frown \theta_1' \Rightarrow (\theta_0, \theta_1)/(\theta_0', \theta_1') = (\theta_0/\theta_0', \theta_1/\theta_1').$$

Obviously the labels of proof are preserved by residuals, that is to say $\ell(\theta) = \ell(\theta/\theta')$.

Let us look at an example, which shows how residuals are affected by choices. The term $p = ((a \parallel b) + c)$ has the following concurrent proved transitions:

$$p \xrightarrow{+_0 \|_0 a} (\mathbf{nil} \parallel b), \qquad p \xrightarrow{+_0 \|_1 b} (a \parallel \mathbf{nil}).$$

Thus the proof of the $b$-transition from $p$ is $+_0 \|_1 b$. On the other hand, once the $a$-transition has occurred, the proof of the $b$-transition becomes $(+_0 \|_1 b)/(+_0 \|_0 a) = \|_1 b$; i.e., we have

$$p \xrightarrow{+_0 \|_0 a} (\mathbf{nil} \parallel b) \xrightarrow{\|_1 b} (\mathbf{nil} \parallel \mathbf{nil}).$$

This shows a difference between our framework and Mazurkiewicz theory of traces. In a trace, two independent actions may always be commuted as they stand, since the residual of one action after the other is the action itself. Intuitively, this is because actions in a trace are essentially events (see later) rather than initial occurrences.

Note that it is indeed necessary to record choices in our proof terms: without the constructors $+_i$ we would not be able to define the concurrency relation on our proofs. For consider the term

$$(a \parallel b) + (a \parallel c).$$

If we did not record the $+_i$ in our proofs, we would not be able to distinguish the two $a$-transitions and thus we would not know, by just looking at their proofs, which is concurrent with the $b$-transition and which is concurrent with the $c$-transition.

We turn now to the main property of the concurrency relation. The following result, also known as the diamond property or the parallel moves

property, states a "conditional Church–Rosser property", namely that whenever two transitions are concurrent then they are confluent. This result is much simpler in CCS than in $\lambda$-calculus or term rewriting systems, since here the proof of a transition cannot be duplicated or deleted by a concurrent one.

LEMMA (the Diamond Lemma). *Let* $t_0 = p \xrightarrow{\theta_0} p_0$ *and* $t_1 = p \xrightarrow{\theta_1} p_1$ *be two proved transitions such that* $\theta_0 \frown \theta_1$. *Then there exists a unique term* $\bar{p}$ *such that* $p_0 \xrightarrow{\theta_1/\theta_0} \bar{p}$ *and* $p_1 \xrightarrow{\theta_0/\theta_1} \bar{p}$.

*Proof.* By induction on the definition of $\frown$. In all cases the uniqueness of $\bar{p}$ follows from the determinacy of the proved transition system, i.e., from the property

$$\left(p \xrightarrow{\theta} p' \;\&\; p \xrightarrow{\theta} p''\right) \Rightarrow p' = p''.$$

We consider the most significant cases for $\theta_0 \frown \theta_1$, leaving the others to the reader.

(i) Basic case: $\theta_0 = \|_0\, \theta_0'$ and $\theta_1 = \|_1\, \theta_1'$. In this case $p = q_0 \| q_1$ with $q_0 \xrightarrow{\theta_0'} q_0'$ and $q_1 \xrightarrow{\theta_1'} q_1'$, hence $p_0 = q_0' \| q_1$ and $p_1 = q_0 \| q_1'$. Then applying rules P1 and P0 we obtain

$$p_0 \xrightarrow{\|_1\, \theta_1'} q_0' \| q_1' \qquad \text{and} \qquad p_1 \xrightarrow{\|_0\, \theta_0'} q_0' \| q_1',$$

which are the required closing transitions, since by the definition of residual we have $\theta_1/\theta_0 = \|_1\, \theta_1'/\|_0\, \theta_0' = \|_1\, \theta_0' = \theta_1$ and $\theta_0/\theta_1 = \|_0\, \theta_0'/\|_1\, \theta_1' = \|_0\, \theta_0' = \theta_0$.

(ii) $\theta_0 = \|_0\, \theta_0'$ and $\theta_1 = (\theta_1', \theta_1'')$, with $\theta_0' \frown \theta_1'$. Then $p = q_0 \| q_1$ with $q_0 \xrightarrow{\theta_0'} q_0'$, $q_0 \xrightarrow{\theta_1'} q_0''$, $q_1 \xrightarrow{\theta_1''} q_1'$, whence $p_0 = q_0' \| q_1$ and $p_1 = q_0'' \| q_1'$. Now since $\theta_0' \frown \theta_1'$, by induction there exists $\bar{q}_0$ such that $q_0' \xrightarrow{\theta_1'/\theta_0'} \bar{q}_0$ and $q_0'' \xrightarrow{\theta_0'/\theta_1'} \bar{q}_0$. Then applying rules C and P0 we obtain

$$q_0' \| q_1 \xrightarrow{(\theta_1'/\theta_0',\, \theta_1'')} \bar{q}_0 \| q_1' \qquad \text{and} \qquad q_0'' \| q_1' \xrightarrow{\|_0\, (\theta_0'/\theta_1')} \bar{q}_0 \| q_1',$$

which are the required transitions, since the residuals are $\theta_1/\theta_0 = (\theta_1', \theta_1'')/\|_0\, \theta_0' = (\theta_1'/\theta_0', \theta_1'')$ and $\theta_0/\theta_1 = \|_0\, \theta_0'/(\theta_1', \theta_1'') = \|_0\, (\theta_0'/\theta_1')$.

(iii) $\theta_0 = +_0\, \theta_0'$ and $\theta_1 = +_0\, \theta_1'$, with $\theta_0' \frown \theta_1'$. Then $p = q_0 + q_1$ with $q_0 \xrightarrow{\theta_0'} q_0'$ and $q_0 \xrightarrow{\theta_1'} q_0''$, and thus $p_0 = q_0'$ and $p_1 = q_0''$. By induction we have for $\theta_0'$, $\theta_1'$ the transitions

$$q_0' \xrightarrow{\theta_1'/\theta_0'} \bar{q}_0 \qquad \text{and} \qquad q_0'' \xrightarrow{\theta_0'/\theta_1'} \bar{q}_0,$$

which are also closing transitions for $\theta_0$, $\theta_1$, since $\theta_1/\theta_0 = +_0 \theta_1'/+_0 \theta_0' = \theta_1'/\theta_0'$ and $\theta_0/\theta_1 = +_0 \theta_0'/+_0 \theta_1' = \theta_0'/\theta_1'$. ∎

This property is in fact much stronger than confluence—note for instance that the transitions of $ab+ba$ are confluent: it says that any proved transition survives a concurrent one. We can then adopt the standard terminology of [3, 5, 35, 37]: the transition $p_0 \xrightarrow{\theta_1/\theta_0} \bar{p}$ (with notations of the diamond lemma) is the *residual* of $t_1$ by $t_0$, denoted $t_1/t_0$, and similarly we have $t_0/t_1 = p_1 \xrightarrow{\theta_0/\theta_1} \bar{p}$.

We are now ready to define the equivalence by permutations, first defined by Lévy and Berry, see [37, 3], on sequences of transitions of CCS terms. Each term $p$ determines a set $\mathcal{T}(p)$ of *sequences* of proved transitions of the form

$$p \xrightarrow{\theta_1} p_1 \cdots p_{n-1} \xrightarrow{\theta_n} p_n$$

which may equivalently be presented as sequences of steps:

$$t_1 \cdots t_n \qquad \text{where} \qquad t_i = p_{i-1} \xrightarrow{\theta_i} p_i,\ p_0 = p, \text{ and } 1 \leqslant i \leqslant n.$$

Intuitively two sequences of proved transitions of $\mathcal{T}(p)$ are equivalent if they are the same up to permutations of concurrent steps. We give next the formal definition of permutation equivalence on $\mathcal{T}(p)$. Let $ss'$ denote the concatenation of $s \in \mathcal{T}(p)$ and $s' \in \mathcal{T}(q)$, which is only defined if $s$ ends at $q$.

DEFINITION (Permutation Equivalence). Let $p$ be a CCS term. The equivalence by permutations on $\mathcal{T}(p)$ is the least equivalence $\simeq$ such that

$$s_0 t_0 (t_1/t_0)\, s_1 \simeq s_0 t_1 (t_0/t_1)\, s_1,$$

provided that $t_0 \frown t_1$ and that concatenation is defined.

The same notion of equivalence of proved transitions, put in a categorical setting, is used by Ferrari and Montanari in [23]. These authors argue that our formulation, presented in [8], remains at the level of transition systems, not exploiting an algebraic framework for defining the permutation equivalence. However, it is an immediate observation that the permutation equivalence is the congruence (on the arrows of the category generated by the proved transitions system) generated by the equations

$$t_0; (t_1/t_0) = t_1; (t_0/t_1) \qquad \text{where} \qquad t_0 \frown t_1.$$

An example of equivalent sequences of transitions is

$$(a.p \parallel b.q) + c.r \xrightarrow{+_0 \parallel_0 a} (p \parallel b.q) \xrightarrow{\parallel_1 b} (p \parallel q)$$

$$(a.p \parallel b.q) + c.r \xrightarrow{+_0 \parallel_1 b} (a.p \parallel q) \xrightarrow{\parallel_0 a} (p \parallel q).$$

Here one can commute two steps. There is another kind of sequence of transitions where this is not possible, because a step is *caused*, or created, by a previous one. The typical example is obviously

$$a.b \xrightarrow{a} b \xrightarrow{b} \mathbf{nil}.$$

Finally, we define the *partial order of computations* of $p$. This is the quotient $\mathscr{C}(p) = \mathscr{T}(p)/\simeq$, ordered by the prefix ordering up to permutations:

DEFINITION (Permutations Semantics). The set $\mathscr{C}(p)$ of computations of the CCS term $p$ is the set $\mathscr{T}(p)/\simeq$ of equivalence classes of sequences of proved transitions of $p$. The computations of $p$ are ordered by

$$[\![s]\!] \sqsubseteq [\![s']\!] \Leftrightarrow_{\text{def}} \exists s''.\, s' \simeq ss''.$$

The domain computations of $p$ is $\mathscr{D}_{\text{per}}(p) = (\mathscr{C}(p), \sqsubseteq)$.

In Section 6 we will show that this poset is isomorphic both to the domain of configurations of an event structure and, for FCCS terms, to the domain of computations of a Petri net. The first result was already stated in [8].

To conclude this section, we relate the permutation semantics to the semantics by means of *pomset transitions* which was introduced in [6] for a subset of CCS. The equivalence class $[\![s]\!]$ of a sequence

$$s = p \xrightarrow{\theta_1} \cdots \xrightarrow{\theta_n} p'$$

may be represented as a one step transition $p \xrightarrow{P} p'$, where $P$ is a pomset (partially ordered multiset [48]) of actions of $A$—that is, an isomorphism class of posets labelled in $A$. Let us formalize this idea: we shall write $s \sim_\zeta s'$ if $s'$ results from $s$ by the transposition of the steps $i$ and $i+1$, and $\zeta$ is the corresponding transposition of $\{1, ..., n\}$, where $n$ is the length of $s$ (obviously $\simeq$ preserves the length of sequences). So $\zeta(i) = i+1$ and $\zeta(i+1) = i$. It should be clear that $s' \simeq s$ if and only if there is a sequence $\zeta_1, ..., \zeta_k$ of such transpositions from $s$ to $s'$. Let us denote this fact by $s \sim_{\zeta_1, ..., \zeta_k} s'$. Then the equivalence class of $s = p \xrightarrow{\theta_1} \cdots \xrightarrow{\theta_n} p'$ determines a

transition $p \xrightarrow{P} p'$, where $P = (E, l, \leqslant)$ is the labelled posed defined (up to the naming of events) by

$$\begin{cases} E = \{e_1, \ldots, e_n\} \\ l(e_i) = \ell(\theta_i) \\ e_i \leqslant e_j \Leftrightarrow \forall s' : s' \sim_{\zeta_1, \ldots, \zeta_k} s \Rightarrow \eta(i) \leqslant \eta(j) \qquad \text{where} \quad \eta = \zeta_k \circ \cdots \circ \zeta_1. \end{cases}$$

Note that $P$ is defined up to isomorphism, since the events $e_i$ are taken arbitrarily. This construction is close to that of *dependency graph* corresponding to a trace—as defined by Mazurkiewicz in [38]. A similar definition is given in [17] for concurrent histories and in [26] for Petri nets. Let us see an example. The equivalence class of the sequence

$$(a.p \parallel b.c.q) \xrightarrow{\|_0 a} (p \parallel b.c.q) \xrightarrow{\|_1 b} (p \parallel c.q) \xrightarrow{\|_1 c} (p \parallel q)$$

may be represented as a transition whose label is a pomset consisting of events $e_1, e_2$ and $e_3$ labelled $a, b$ and $c$ respectively, where $e_2$ precedes $e_3$ and $e_1$ is incomparable with $e_2$ and $e_3$, that is:

$$(a.p \parallel b.c.q) \xrightarrow{\left\{\begin{smallmatrix} a & b \\ & | \\ & c \end{smallmatrix}\right\}} (p \parallel q).$$

## 4. Flow Event Structure Semantics

In this section we present an event structure semantics for CCS, which will be later related, in Section 6, to the semantics by permutations. Terms of CCS will be interpreted as *flow event structures*: this semantics was first proposed in [8] and, in a slightly different form, in [15]. The flow event structure semantics may be seen as a variation of Winskel's interpretation of CCS by means of *stable event structures* [58], and agrees with it in a rather strong sense: it yields the same families of configurations for any term. We recall some general points about event structures, for the readers not familiar with this model.

Event structures (e.s.) were introduced by Nielsen *et al.* in [43] as a model for computational processes. These were first order structures, essentially sets of events with two binary relations of *causality* and *conflict* between them, satisfying particular constraints. These structures, also called *prime event structures*, were shown to be connected both with a class of Petri nets—called by the authors [43] occurrence nets—and with a class of domains: their spaces of computations, i.e., configurations, are finitary prime algebraic coherent domains. In fact prime event structures are just equivalent—if more concrete—representations for their domains of

configurations. In particular they can be retrieved from such domains by picking up particular configurations—the complete prime elements of the domain.

Because they are essentially models of computations, i.e., unfoldings of processes, prime event structures are too rigid for interpreting in a simple way process operators like the parallel product or data type constructions like exponentiation. This is why in subsequent works [56, 58, 59] Winskel turned to a more general class of event structures, stable event structures, for which such constructions may be defined in a natural way—and coincide with categorical constructs.

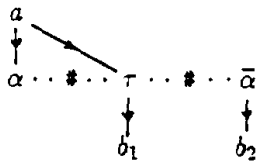
However, by moving to more general structures, one loses some of the suggestiveness of the model. Stable e.s. are described at a more abstract level than prime e.s.: the causality relation on events is replaced by a second order *enabling* relation for events, specifying when a set of events is a possible set of causes for a given event. As a consequence, one loses the graphical representation of processes as sets of events with two kinds of arcs between them, representing causality and conflict, which was a pleasant feature of prime event structures.

Flow event structures were proposed in [8] as an intermediate between prime event structures and stable event structures, allowing both a graphical representation and an easy definition of process operators. In fact, flow event structures are a direct generalisation of prime event structures, where the conflict relation is not inherited and the partial ordering of causality is replaced by a local *flow* relation on events, representing immediate causality. Moreover, there is no requirement on the relation between flow and conflict. With these new structures it is fairly easy to interpret CCS terms in a "natural" way.

Let us be more precise about the meaning of "natural" here. We have somehow suggested that a semantics by means of stable or flow event structures may be more natural than one by prime event structures. On the other hand, interpretations of CCS or related languages have been defined by means of prime, e.s. by Goltz and Loogen [32], by Degano *et al.* [19], and more recently by Vaandrager [55].

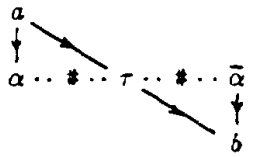
Our point is that in Winskel's stable e.s. semantics, as well as in the flow e.s. semantics, events have a clear operational meaning: they are just occurrences of actions (or pairs of occurrences, for a communication) in the syntactic tree obtained by unfolding the CCS term. For instance, in the term $(a\alpha \mid \bar{\alpha}b)$, the action $b$ gives rise to a unique event, which can be caused in two incompatible ways, either by the action $\bar{\alpha}$ alone, or by the communication action. Stable and flow event structures account for such conflicting causes, while prime event structures—constrained by the principle of conflict heredity—require as many copies of an event as there are conflicting sets of causes for it. As a matter of fact, events in a prime

event structure are *histories* of an occurrence in the syntactic tree. For example, the prime event structure interpretation of $(a\alpha \mid \bar{\alpha}b)$ is




where there are two events $b_1$ and $b_2$ representing the two possible histories of the same occurrence $b$. Using prime event structures to interpret other simple terms may be quite laborious—try for instance $(\alpha a\beta \mid \bar{\beta}b\bar{\alpha})$ where causality cycles arise. Not surprisingly, the simplest—and most elegant—way to define an operation on prime e.s. is to define it first on the configurations, which are themselves histories, and then recover the corresponding prime e.s. (see [55, 58]).

This is the reason that, to our view, stable and flow e.s. are more suited than prime e.s. for modelling process operators. Flow e.s. are preferred here since they allow—just like prime e.s.—a graphical representation of processes. For instance, the flow event structure interpretation of the term $(a\alpha \mid \bar{\alpha}b)$ will be




where the arrows represent now an immediate causality. Such causality is in general *not* transitive. Here for example $a$ causes $\tau$, and $\tau$ causes $b$, but $a$ does not cause $b$: indeed $a$ and $b$ may occur independently, e.g., in the computation $\{a, \bar{\alpha}, b\}$ where the communication does not take place. We will see later in this section the flow e.s. interpretation of $(\alpha a\beta \mid \bar{\beta}b\bar{\alpha})$, together with various other examples.

In the rest of this section, we shall only be concerned with flow event structures. For more about prime and stable e.s. we refer the reader to [43] and to Winskel's papers [56, 58, 59]. We proceed now with the formal definitions.

DEFINITION (Flow Event Structure). A flow event stucture is a structure $S = (E, \#, \prec)$, where

(i) $E$ is a denumerable set of events

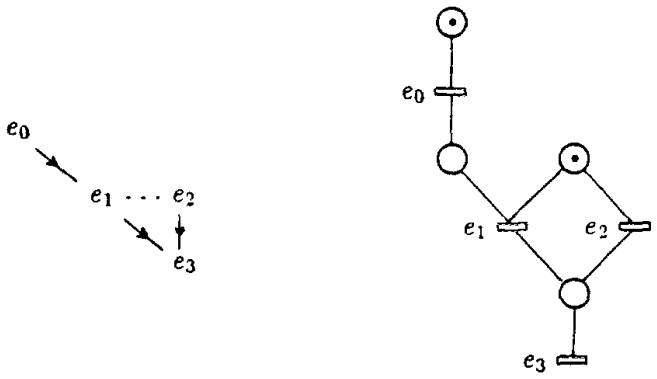
(ii) $\# \subseteq E \times E$ is a symmetric relation, the conflict relation

(iii) $\prec \subseteq E \times E$ is an irreflexive relation, the flow relation.

For those familiar with prime event structures, it will be clear that any prime event structure $S = (E, \#, \leqslant)$ may be regarded as a flow event structure, with $\prec$ given by the strict ordering $<$, or by its covering relation, if it generates the order. Note that the flow relation is not required to be transitive, nor acyclic. Intuitively, the flow relation presents a possible *immediate causality*. A simple way of understanding the flow relation is by analogy with Petri nets: a flow between two events in an event structure corresponds to the presence of a condition between the events in a net. This point is illustrated by the example below.

Note also that the conflict relation is not assumed to be irreflexive. This means that we allow *self-conflicting* or *inconsistent* events, that is events $e \in E$ such that $e \# e$. Such events will be used for defining the operation of restriction, and it may be shown that they are indeed convenient to obtain correct constructions on flow event structures, see [15].

As we said, the first order flow event structures allow a graphical representation, with two kinds of arcs between events. In this representation we shall draw $e \prec e'$ as a directed arc $e \rightarrowtail e'$, and $\#$ as a dotted line. Self-conflicts will be represented by dotted circles around events. The following is an example of flow event structure, together with a "corresponding" Petri net:




This example, with typically arises when CCS communication is modelled, exhibits both a confluence after conflict and a case where the flow $\prec$ is essentially not transitive: the events $e_0$ and $e_3$ are indeed causally related if $e_1$ occurs, but they are independent if $e_2$ occurs. In other words, $e_0$ and $e_3$ are in a different relation depending on the computation where they are considered.

We shall now formalise this notion of computation or *configuration* for flow event structures. A configuration is a set of events occurring at some stage of evolution of a process. Since flow event structures are rather general, the definition of configuration is slightly more elaborated than for prime event structures. Let $\boldsymbol{Con}_S$ be the set of conflict-free, or consistent, sets of events: $X \in \boldsymbol{Con}_s$ iff $\forall e, e' \in X, \neg(e \# e')$. Obviously, an event $e$ is

inconsistent, i.e., $e \# e$, if and only if $e \notin X$ for any $X \in \mathbf{Con}_S$. For a subset $X$ of $E$, let $\prec_X$ be the restriction of the flow relation to $X$ and let $\leqslant_X =_{\text{def}} \prec_{X^*}$ be the preordering generated by $\prec_X$. Here we shall only deal with *finite* configurations, which are enough to determine the whole domain associated with an event structure, see [10]. We recall the definition:

DEFINITION (Configurations). Let $S = (E, \#, \prec)$ be a flow event structure. A (finite) configuration of $S$ is a finite subset $X$ of $E$ such that:

(i) $X$ is a conflict-free: $X \in \mathbf{Con}_S$

(ii) $X$ is left-closed up to conflicts: $e' \prec e \in X \,\&\, e' \notin X \Rightarrow \exists e'' \in X. e' \# e'' \prec e$

(iii) $X$ has no causality cycles: the relation $\leqslant_X$ is an ordering.

The first conditions are essentially the same as for prime event structures: condition (ii) is adapted to account for the more general—non-hereditary—conflict relation. It states that any event appears in a configuration with a "complete" set of causes. Condition (iii) ensures that any event in a configuration is actually reachable at some stage of computation. Note that an inconsistent event cannot appear in a configuration. The set of (finite) configurations of a flow event structure $S$ will be denoted by $\mathscr{F}(S)$, and its *domain of configurations*, the poset of these configurations ordered by inclusion, by $\mathscr{D}(S) =_{\text{def}} (\mathscr{F}(S), \subseteq)$. It is shown in [10] that for any flow e.s. $S$ the poset $\mathscr{D}(S)$ is a finitary coherent, prime algebraic, and finitary domain, as for prime and stable event structures; see [43, 56, 59].

Let us turn now to the interpretation of CCS by means of flow event structures. We shall directly define from the syntactical materials a structure $\mathscr{S}(p) = (\mathscr{E}(p), \prec, \#)$ for each term $p$. This definition will be compared later with that given by Winskel [56, 58], where one defines a construction on event structures for each CCS operator and interprets FCCS by a morphism into the event structure algebra. We start by defining a global set of events $\mathscr{E}$, from which $\mathscr{E}(p)$ will be extracted. Events are occurrences of possibly guarded actions in a term: to specify them syntactically we just have to extend the syntax for proof terms, given in the previous section, to allow a proof to pass through a guard $a.p$. We will denote by $\hat{a}.e$ the occurrence of $e$ after a guard $a$. The set $\mathscr{E}$ of *events* has thus the following syntax:

$$e ::= a \mid \hat{a}.e \mid \|_0 e \mid \|_1 e \mid (e, e') \mid +_0 e \mid +_1 e \mid \backslash_\alpha e.$$

For instance, the occurrence of the action $b$ in $a.b.\mathbf{nil}$ will be $\hat{a}.b$. The labelling $\ell$ of proof terms is extended to events in the obvious way: $\ell(\hat{a}.e) = \ell(e)$. Let us now define the notions of conflict and flow on

events. The *conflict* relation $e \# e'$ is the least symmetric relation which satisfies the following clauses, where we denote by $\sharp$ the reflexive closure of $\#$:

(B1) $i \neq j \Rightarrow +_i e \# +_j e'$

(B2) $e \sharp e' \Rightarrow \begin{cases} \|_0 e \# (e', e) \\ \|_1 e \# (e'', e) \end{cases}$

(B3) $e \# e \Rightarrow \begin{cases} \|_i e \# \|_i e' \\ +_i e \# +_i e' \\ \backslash_\alpha e \# \backslash_\alpha e' \\ \hat{a}.e \# \hat{a}.e' \end{cases}$

(B4) $\mathbf{nm}(\ell(e)) = \alpha \Rightarrow \backslash_\alpha e \# \backslash_\alpha e$

(B5) $e_0 \sharp e_0'$ or $e_1 \sharp e_1'$ and $(e_0, e_1) \neq (e_0', e_1') \Rightarrow (e_0, e_1) \# (e_0', e_1')$

$e_0 \# e_0$ or $e_1 \# e_1 \Rightarrow (e_0, e_1) \# (e_0, e_1)$.

Note the way we interpret restriction: if the event $e$ is an occurrence of the action $\alpha$ or $\bar{\alpha}$ then $\backslash_\alpha e$ is self-conflicting. Note also the formal similarity with the definition of *concurrency* on proofs given in the previous section. In fact, the concurrency relation could be defined on events too by extending in the obvious way the definition for proofs. Indeed, concurrency on events will be considered later, in Section 6.1.

We define now the *flow* relation on events. In the structure $\mathscr{S}(p)$, the flow represents a possible immediate precedence. Quite obviously the relation $e \prec e'$ is brought in by the action construct $a.p$—loosely speaking $a \prec \hat{a}.e$. More precisely, $\prec$ is the least relation on $\mathscr{E}$ satisfying the following clauses:

(C1) $a \prec \hat{a}.\theta$ where $\theta$ is any proof term

(C2) $e \prec e' \Rightarrow \begin{cases} (e, e'') \prec \|_0 e' \\ (e'', e) \prec \|_1 e' \end{cases}$ and $\begin{cases} \|_0 e \prec (e', e'') \\ \|_1 e \prec (e'', e') \end{cases}$

(C3) $e \prec e' \Rightarrow \begin{cases} (e, e_1) \prec (e', e_1') \\ (e_0, e) \prec (e_0', e') \end{cases}$

(C4) $e \prec e' \Rightarrow \begin{cases} \|_i e \prec \|_i e' \\ +_i e \prec +_i e' \\ \backslash_\alpha e \prec \backslash_\alpha e' \\ \hat{a}.e \prec \hat{a}.e'. \end{cases}$

The relation $\prec$ is irreflexive. Note on the other hand that it is not transitive: for instance, if $e_0 \prec e_0'$ and $e_1 \prec e_1'$, then $\|_0 e_0 \prec (e_0', e_1)$ and

$(e_0', e_1) \prec \|_1 e_1'$, but we do not have $\|_0 e_0 \prec \|_1 e_1'$. Let us see some examples: in the term $r = (\alpha.\alpha \parallel \bar{\alpha})$ we have

$$(\alpha, \bar{\alpha}) \prec (\hat{\alpha}.\alpha, \bar{\alpha})$$

$$(\alpha, \bar{\alpha}) \# (\hat{\alpha}.\alpha, \bar{\alpha})$$

This shows that $\#$ and $\prec$ are not necessarily disjoint. The following example shows that the transitive closure of $\prec$ is not disjoint from $\smile$ (extended to events in the obvious way, see Section 6 for a formal definition): in term $q = (a.\alpha \parallel \bar{\alpha}.b)$ we have

$$\|_0 a \prec (\hat{a}.\alpha, \bar{\alpha}) \prec \|_1 \hat{\bar{\alpha}} \qquad \text{and} \qquad \|_0 a \smile \|_1 \hat{\bar{\alpha}}.b.$$

Note also that $\prec$ is not asymmetric; for instance, in the term $(\alpha.\beta \parallel \bar{\beta}.\bar{\alpha})$ we have

$$(\alpha, \hat{\bar{\beta}}\bar{\alpha}) \prec (\hat{\alpha}\beta, \bar{\beta}) \prec (\alpha, \hat{\bar{\beta}}\bar{\alpha}).$$

To define the structure $\mathcal{S}(p)$ it just remains to define its set of events $\mathcal{E}(p) \subseteq \mathcal{E}$. Then the flow and conflict relations in $\mathcal{S}(p) = (\mathcal{E}(p), \prec, \#)$ will simply be the restrictions to $\mathcal{E}(p)$ of the corresponding relations on $\mathcal{E}$. The set $\mathcal{E}(p)$ of events of $p$ is defined inductively; that is, $\mathcal{E}(p)$ is the least subset of $\mathcal{E}$ satisfying

(E1) $a \in \mathcal{E}(a.p)$

if $e \in \mathcal{E}(p)$ then $\hat{a}.e \in \mathcal{E}(a.p)$

(E2) if $e \in \mathcal{E}(p_i)$ then $\|_i e \in \mathcal{E}(p_0 \parallel p_1)$

if $e \in \mathcal{E}(p_0)$ and $e' \in \mathcal{E}(p_1)$ and $\ell(e) = \overline{\ell(e')}$, then $(e, e') \in \mathcal{E}(p_0 \parallel p_1)$

(E3) if $e \in \mathcal{E}(p_i)$ then $+_i e \in \mathcal{E}(p_0 + p_1)$

(E4) if $e \in \mathcal{E}(p)$ then $\backslash_\alpha e \in \mathcal{E}(p\backslash\alpha)$

(E5) if $e \in \mathcal{E}(p[\mu x.p/x])$ then $e \in \mathcal{E}(\mu x.p)$.

*Remark.* Usually to interpret the CCS constructs one defines, as in Winskel's semantics [58] (see also [8]), set-theoretical constructions on event structures. For instance, one uses disjoint union of sets of events,

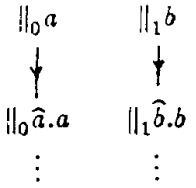
$$E_0 \uplus E_1 = \{(i, e_i) \mid e_i \in E_i, i = 0, 1\}$$

to interpret the sum and a product of these sets,

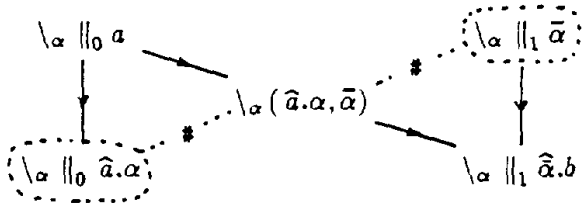
$$E_0 \times_* E_1 = (E_0 \times \{*\}) \cup (\{*\} \times E_1) \cup \{(e_0, e_1) \mid e_i \in E_i \,\&\, \ell(e_0) = \overline{\ell(e_1)}\},$$

to interpret parallel composition. It should be clear that this "denotational" or "algebraic" approach yields structures which are isomorphic to the ones we obtain using an "operational" or syntactic approach. In fact, event names like $(e, *)$ or $(0, e)$ are just notational variants for $\|_0 e$ and $+_0 e$. Note, however, that in the algebraic approach, one must use *labelled* event structures, since the labels are not extracted from the names of the events. Moreover, interpreting the fixpoint construct is quite tedious in the algebraic approach.

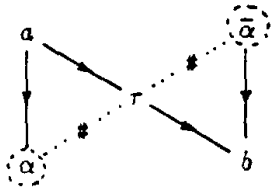
Let us see some examples, illustrating our definition of $\mathcal{S}(p)$. Note that $\mathcal{E}(\mathbf{nil}) = \varnothing = \mathcal{E}(\mu x.x)$ and $\mathcal{S}(\mu x.p) = \mathcal{S}(p[\mu x.p/x])$. For instance the flow event structure $\mathcal{S}(a^\omega \parallel b^\omega)$ may be drawn as follows:



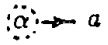

Similarly, if $r = (a.\alpha \parallel \bar{\alpha}.b)\backslash\alpha$, then $\mathcal{S}(r)$ may be drawn as follows:




Clearly $\backslash_\alpha \|_0 \hat{a}.\alpha$ and $\backslash_\alpha \|_1 \hat{\bar{\alpha}}$ cannot occur in a configuration since they are inconsistent. From now on we will not write the full names of events in drawings, but only their labels. Then the structure $\mathcal{S}(r)$ will be drawn simply as

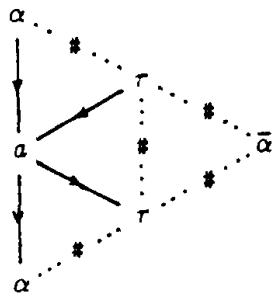


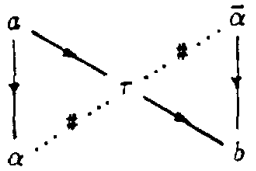

The interpretation of $\alpha.a.\mathbf{nil}\backslash\alpha$ is



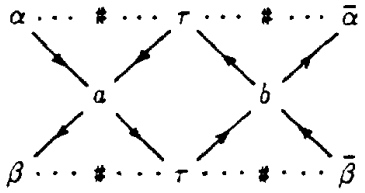
This structure has only the trivial configuration $\varnothing$. The structure $\mathscr{S}((\alpha.\beta \parallel \bar{\alpha}))$ may be drawn as follows:


This example shows that $\mathscr{S}(p)$ may contain a substructure $\nabla'$ (see [6]). The interpretations of $(\alpha.a.\alpha \parallel \bar{\alpha})$ and $(a.\alpha \parallel \bar{\alpha}.b)$ may be drawn respectively




The second structure contains a substructure $N$ and a substructure $\nabla$, see [6]. An example, suggested by M. Nielsen, shows that $\prec^*$ is not an ordering: if we interpret $(\alpha.a.\beta \parallel \bar{\beta}.b.\bar{\alpha})$ we get




Note that if $r = (\alpha.a.\beta \parallel \bar{\beta}.b.\bar{\alpha})\backslash\alpha\backslash\beta$ then the structure $\mathscr{S}(r)$ has no proper configuration $X$, i.e., such that $X \neq \varnothing$. The same holds for the interpretation of $(\alpha.\beta \parallel \bar{\beta}.\bar{\alpha})\backslash\alpha\backslash\beta$. One can also see that in the interpretation of $((\alpha.\beta \parallel \bar{\alpha}) \parallel \alpha.\bar{\beta})\backslash\alpha\backslash\beta$, there is no configuration containing the $\beta$ communication.

We are now ready to give our second semantics for CCS:

DEFINITION (Event Structure Semantics). The set of event structure computations of the CCS term $p$ is the set $\mathscr{F}(\mathscr{S}(p))$ of configurations of the flow event structure $\mathscr{S}(p)$. The domain of event computations of $p$ is $\mathscr{D}_{es}(p) = \mathscr{D}(\mathscr{S}(p)) = (\mathscr{F}(\mathscr{S}(p)), \subseteq)$.

We will show that this flow event structure semantics of CCS has an *operational meaning*, in that it corresponds to the "truly concurrent"

semantics defined in the previous section. More precisely, we will show that for any term $p$ the domain $\mathscr{D}_{es}(p)$ is *isomorphic* to the domain $\mathscr{D}_{per}(p)$ of equivalence classes of proved transitions, as was defined in the previous section.

## 5. Flow Net Semantics

In this section we give an interpretation of FCCS terms as *safe flow nets.* The reason for the restriction to fixpoint-free terms will be explained later. We assume some elementary knowledge of Petri nets: the reader is referred to [25, 49] for a general introduction to nets.

Flow nets are a new class of nets introduced in [10], strictly more general than the *occurrence nets* considered in [43, 57, 29]: e.g., flow nets may have cycles in the flow relation. However, they are still "semantically acyclic." It is shown in [10] that there is a very close correspondence between flow nets and flow events structures, the model we considered in the previous section. The reader may want to look at the definition of flow net in [10], but this is not required fo reading through this section: the flow net interpretation for FCCS presented here is essentially the standard Petri net semantics (see [57, 26, 30, 31, 20, 45]), with the pecularity that events and places are defined themselves structurally, using the syntax of terms.

The idea of defining the places of a net syntactically is taken from [20]. Maintaining a syntax on events and states will allow us to compare more easily our flow net semantics with the other CCS semantics considered earlier on. We will use the following definition of (marked) net.

Definition (Nets). A net is a structure $N=(B, E, \Phi, \mu_N)$, where:

(i) $B$ is the set of conditions, or places

(ii) $E$ is the set of events, denumerable and disjoint from $B$

(iii) $\Phi \subseteq (B\times E)\cup(E\times B)$ is the flow relation, satisfying the two conditions:
the set $\{b \mid \exists e, e'.(e, b)\in\Phi \,\&\, (b, e')\in\Phi\}$ is denumerable
$\forall e\in E\, \exists b\in B.(b, e)\in\Phi$

(iv) $\mu_N : B \to \mathbb{N}$ is the initial marking.

A place $b$ is called *initial* if it is initially marked, that is, $\mu_N(b)\geqslant 1$. It is often convenient to represent the flow relation as a mapping $\phi : (B\times E)\cup(E\times B) \to \{0, 1\}$, where $\phi(x)=1 \Leftrightarrow x\in\Phi$. We will use the notation $b \overset{e}{\mapsto} b'$ to mean $\phi(b, e)=1=\phi(e, b')$, or also $b \overset{e}{\mapsto} b'$ in $\Phi_N$ when

we want to explicitly refer to the net $N$. The sets of *preconditions* and *post-conditions* of an event $e$ are given by

$$\mathbf{pre}(e) = \{b \mid \phi(b, e) = 1\}$$

$$\mathbf{post}(e) = \{b \mid \phi(e, b) = 1\}.$$

Similarly we may define the sets of pre-events and post-events of a place $b$, respectively $\mathbf{pre}(b)$ and $\mathbf{post}(b)$. We then say that a place $b$ is a precondition if it has at least one post-event; $\mathbf{post}(b) \neq \varnothing$, and a post-condition if it has at least one pre-event; $\mathbf{pre}(b) \neq \varnothing$.

We shall now give an interpretation of FCCS terms as (marked) nets. The nets thus obtained will be shown to be one-safe, as well as flow nets in the sense of [10]. When interpreting FCCS, or similar algebraic languages, one may use the syntax of terms to specify places and events of the corresponding nets. We take the syntax of *events* to be the same as for flow event structures:

$$e ::= a \mid \hat{a}.e \mid \|_0 e \mid \|_1 e \mid (e, e') \mid +_0 e \mid +_1 e \mid \backslash_\alpha e.$$

In a similar style, the syntax of *places* is given by the grammar, where $p$ is any FCCS term:

$$b ::= \mathbf{nil} \mid a.p \mid \hat{a}.b \mid \|_0 b \mid \|_1 b \mid (b + b') \mid +_0 b \mid +_1 b \mid \backslash_\alpha b.$$

This syntax is similar to that of "grapes" of [17, 20], except for the terms $\hat{a}.b$ and $+_i b$. We shall comment in detail on the meaning of such events and places as we go along interesting FCCS terms. Let us just recall that events are built starting from an occurrence $a$, or two occurrences $\alpha$ and $\bar{\alpha}$ in the case of communication. Thus an event records an occurrence of an action $a$, or $\tau$, together with its "past," or pair of pasts. On the other hand, places—which are local states—record both past and "future." These points will be further illustrated by examples.

Note also that events and places are disjoint—as required by the definition of net—because the algebras are built on different constants. Therefore, in this section, although we will mostly omit the trailing **nil**'s in FCCS terms, we will avoid doing so in the names of places.

We now proceed to define the marked net $\mathcal{N}(p) = (B_p, E_p, \Phi_p, \mu_p)$ associated with an FCCS term $p$. In the definition we will use the following conventions: since all events in $\mathcal{N}(p)$ will have at least one precondition and one postcondition, the flow relation $\Phi_p$ will be specified using the notation $b \overset{e}{\mapsto} b'$ in $\Phi_p$; also, since in all constructions the inital marking will be safe, i.e., $\forall b.\mu_p(b) \leqslant 1$, we will specify $\mu_p$ as a subset of $B_p$.

The definition of $\mathcal{N}(p)$ is now given by structural induction on the FCCS term $p$.

(i) The net $\mathcal{N}(\mathbf{nil})$ consists of just one place **nil**, initially marked:

$$B_{\mathbf{nil}} = \mu_{\mathbf{nil}} = \{\mathbf{nil}\}$$
$$E_{\mathbf{nil}} = \Phi_{\mathbf{nil}} = \varnothing.$$

(ii) *Prefixing* $\mathcal{N}(a.p)$ adds an event $a$ with a precondition $a.p$ in front of net $\mathcal{N}(p)$, while shifting all the other events and conditions $\hat{a}.$. For a set $X$, we shall write $\hat{a}.X$ for $\{\hat{a}.x \mid x \in X\}$. Then $\mathcal{N}(a.p)$ is given by

$$B_{a.p} = \{a.p\} \cup \hat{a}.B_p$$
$$\mu_{a.p} = \{a.p\}$$
$$E_{a.p} = \{a\} \cup \hat{a}.E_p.$$

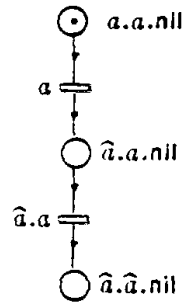
$\Phi_{a.p}$ is the least relation such that

$$b \in \mu_p \Rightarrow a.p \overset{a}{\mapsto} \hat{a}.b \text{ in } \Phi_{a.p}$$
$$b \overset{e}{\mapsto} b' \text{ in } \Phi_p \Rightarrow \hat{a}.b \overset{\hat{a}.e}{\mapsto} \hat{a}.b' \text{ in } \Phi_{a.p}.$$

As for flow event structures, events not involving the constructors $\hat{a}.$—what we called proof terms earlier on—represent immediate occurrences of an action in a process, i.e., initial occurrences in its synchronisation tree; the constructor $\hat{a}.$ allows one to pass a guard $a$ and thus to specify events at any depth in the tree.

Similar remarks apply to places. A place of the form $a.p$ is an initial place, more precisely the initial place of a guarded term; for a general term, we shall see that all places built without the constructors $\hat{a}.$ are initial. Note that the initial event of $\mathcal{N}(a.p)$ is taken to be simply the label $a$. This is the only case where an event is confused with its label.

EXAMPLE 1. The interpretation of the term $a.a.\mathbf{nil}$ is the net




(iii) *Parallel composition* $\mathcal{N}(p \parallel q)$ puts side by side its components $\mathcal{N}(p)$ and $\mathcal{N}(q)$ while shifting their events and places by $\|_0$ and $\|_1$ respectively. It may also introduce communication events, which are built with

the constructor $(e, e')$. On the other hand there are no special place constructors for communication, since the preconditions and postconditions of a communication are just those of its components. For a set $X$, we write $\|_i X$ for $\{\|_i x \mid x \in X\}$. Then $\mathcal{N}(p \| q)$ is given by

$$B_{p\|q} = \|_0 B_p \cup \|_1 B_q$$

$$\mu_{p\|q} = \|_0 \mu_p \cup \|_1 \mu_q,$$

$$E_{p\|q} = \|_0 E_p \cup \|_1 E_q \cup \{(e, e') \mid e \in E_p \,\&\, e' \in E_q \,\&\, \overline{\ell(e)} = \ell(e')\}.$$

$\Phi_{p\|q}$ is the least relations such that

$$b \overset{e}{\longmapsto} b' \text{ in } \Phi_p \Rightarrow \|_0 b \overset{\|_0 e}{\longmapsto} \|_0 b' \text{ in } \Phi_{p\|q}$$

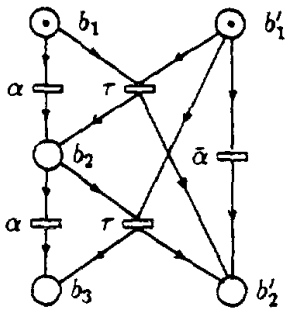
$$b \overset{e}{\longmapsto} b' \text{ in } \Phi_q \Rightarrow \|_1 b \overset{\|_1 e}{\longmapsto} \|_1 b' \text{ in } \Phi_{p\|q},$$

moreover, for any $e \in E_p$, $e' \in E_q$ s.t. $\overline{\ell(e)} = \ell(e')$,

$$b \overset{e}{\longmapsto} b' \text{ in } \Phi_p \Rightarrow \|_0 b \overset{(e,e')}{\longmapsto} \|_0 b' \text{ in } \Phi_{p\|q}$$

$$b \overset{e'}{\longmapsto} b' \text{ in } \Phi_q \Rightarrow \|_1 b \overset{(e,e')}{\longmapsto} \|_1 b' \text{ in } \Phi_{p\|q}.$$

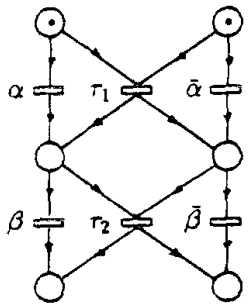
EXAMPLE 2. The interpretation of $(\alpha.\alpha \| \bar{\alpha})$ is, representing for simplicity the events by their labels,




Here the places are $b_1 = \|_0 \alpha.\alpha.\mathbf{nil}$, $b_2 = \|_0 \hat{\alpha}.\alpha.\mathbf{nil}$, $b_3 = \|_0 \hat{\alpha}.\hat{\alpha}.\mathbf{nil}$, $b'_1 = \|_1 \bar{\alpha}.\mathbf{nil}$, $b'_2 = \|_1 \hat{\bar{\alpha}}.\mathbf{nil}$, while the events are $\|_0 \alpha$, $\|_0 \hat{\alpha}.\alpha$, $(\alpha, \bar{\alpha})$, $(\hat{\alpha}.\alpha, \bar{\alpha})$, $\|_1 \bar{\alpha}$.

From now, on we shall omit the names of places and events in the diagrams, showing only the labels of events—possibly with an index when the same label occurs more than once.

EXAMPLE 3. The interpretation of the term $(\alpha.\beta \parallel \bar{\alpha}.\bar{\beta})$ will be drawn as




Here the places are

$$b_1 = \|_0\, \alpha.\beta.\mathbf{nil} \qquad b_2 = \|_0\, \hat{\alpha}.\beta.\mathbf{nil} \qquad b_3 = \|_0\, \hat{\alpha}.\hat{\beta}.\mathbf{nil} \qquad \text{and}$$
$$b'_1 = \|_1\, \bar{\alpha}.\bar{\beta}.\mathbf{nil} \qquad b'_2 = \|_1\, \hat{\bar{\alpha}}.\bar{\beta}.\mathbf{nil} \qquad b'_3 = \|_1\, \hat{\bar{\alpha}}.\hat{\bar{\beta}}.\mathbf{nil},$$

and the communication events are

$$\tau_1 = (\alpha, \bar{\alpha}) \qquad \text{and}$$
$$\tau_2 = (\hat{\alpha}.\beta, \hat{\bar{\alpha}}.\bar{\beta}).$$

Note that the "past" of $\tau_2$ consists of the two events labelled $\alpha$ and $\bar{\alpha}$, regardless of the way these events have happened, independently or simultaneously as $\tau_1$. In fact the past recorded in an event is its history projected on the local components, which is unique and statically determined, while in general an event may have different *execution* histories, due to communication.

(iv) *Nondeterminism* $\mathcal{N}(p+q)$ does not create new events. It shifts the names of events and non-initial places of its components respecitvely by $+_0$ and $+_1$. The constructor $(b+b')$ is used to build the Cartesian product of the *initial* places of the two nets. Note that unlike $(e, e')$ the constructor $(b+b')$ may be iterated any number of times. If $X$, $Y$ are sets of places, we write $(X+Y)$ for $\{(b+b') \mid b \in X, b' \in Y\}$ and $+_i X$ for $\{+_i x \mid x \in X\}$:

$$B_{p+q} = (\mu_p + \mu_q) \cup +_0 (B_p - \mu_p) \cup +_1 (B_q - \mu_q)$$
$$\mu_{p+q} = (\mu_p + \mu_q)$$
$$E_{p+q} = +_0 E_p \cup +_1 E_q.$$

$\Phi_{p+q}$ is the least relation such that

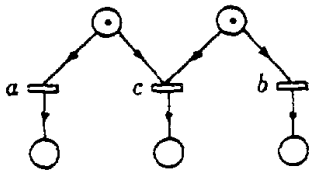
$$b \xmapsto{e} b' \text{ in } \Phi_p, \text{ with } b \notin \mu_p \Rightarrow +_0 b \xmapsto{+_0 e} +_0 b' \text{ in } \Phi_{p+q}$$

$$b \xmapsto{e} b' \text{ in } \Phi_q, \text{ with } b \notin \mu_q \Rightarrow +_1 b \xmapsto{+_1 e} +_1 b' \text{ in } \Phi_{p+q}$$

$b \overset{e}{\longmapsto} b'$ **in** $\Phi_p$, with $b \in \mu_p$ and $b'' \in \mu_q \Rightarrow (b + b'') \overset{+_0 e}{\longmapsto} +_0 b'$ **in** $\Phi_{p+q}$

$b \overset{e}{\longmapsto} b'$ **in** $\Phi_q$, with $b \in \mu_q$ and $b'' \in \mu_p \Rightarrow (b'' + b) \overset{+_1 e}{\longmapsto} +_1 b'$ **in** $\Phi_{p+q}$.

EXAMPLE 4 (Symmetric Confusion). The interpretation of the term $(a \parallel b) + c$ is




where the initial places are $(\|_0 a.\mathbf{nil} + c.\mathbf{nil})$ and $(\|_1 b.\mathbf{nil} + c.\mathbf{nil})$.

*Remark.* We may now explain why, for simplicity, we restrict ourselves to terms written without recursion. It would not be too difficult to interpret a term like $(\alpha \parallel (\mu x.(\bar{\alpha} \parallel x)))$ as an infinite flow net, since any relevant piece of information (concerning the places, events, flow, initial marking) may be found in some unfolding of this term. This is not the case for $\mu x.(a + x)$, which requires introducing an "infinite initial choice place" that is a common precondition for all events $+_1 \cdots +_1 +_0 a$. Since we do not want to enter here into technical developments about limits of flow nets, we do not deal with the flow net interpretation of such infinite processes.

(v) *Restriction* $\mathcal{N}(p \backslash \alpha)$ takes away from $\mathcal{N}(p)$ the subset of events bearing a given label $\alpha$ or its complement $\bar{\alpha}$, while maintaining places unchanged. All events are shifted by $\backslash_\alpha$, and similarly for the places—although this is not really needed:

$$B_{p \backslash \alpha} = \backslash_\alpha B_p$$

$$\mu_{p \backslash \alpha} = \backslash_\alpha \mu_p$$

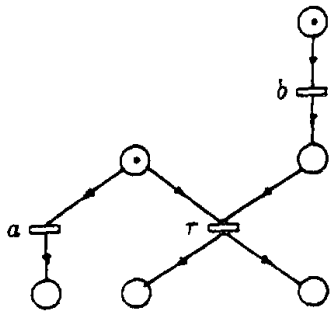
$$E_{p \backslash \alpha} = \backslash_\alpha (E_p - E_\alpha), \text{ where } E_\alpha = \{e \in E_p \mid \mathbf{nm}(\ell(e)) = \alpha\}.$$

$\Phi_{p \backslash \alpha}$ is given by

$$b \overset{e}{\longmapsto} b' \text{ in } \Phi_p, \text{ with } e \notin E_\alpha \Rightarrow \backslash_\alpha b \overset{\backslash_\alpha e}{\longmapsto} \backslash_\alpha b' \text{ in } \Phi_{p \backslash \alpha}.$$

Recall that $b \overset{e}{\longmapsto} b'$ is a purely static relation, which should not be confused with the transition relation on markings describing the behavior of the net—which is given below.

EXAMPLE 5 (Asymmetric Confusion). the interpretation of the term $((a+\alpha)\| b.\bar{\alpha})\backslash_\alpha$ is




This concludes the definition of our net semantics for FCCS. We shall now turn to the *behaviour* of such nets. It is well known that a net determines a transition system on its markings. Let us briefly recall the definition.

A marking of the net $N=(B, E, \Phi, \mu_N)$ is any mapping $\mu : B \to \mathbb{N}$. If $\mu(b)>0$ we say that the condition $b$ holds at $\mu$. A marking $\mu$ *enables* an event $e$ if all the preconditions of $e$ hold in $\mu$, or more formally: $\forall b \in B : \phi(b, e) \leqslant \mu(b)$. Then the labelled transition system on the markings of net $N$ is defined as follows:

$$\mu \xrightarrow{e} \mu' \Leftrightarrow_{\text{def}} \forall b \in B : \phi(b, e) \leqslant \mu(b) \quad \text{and} \quad \mu'(b) = \mu(b) - \phi(b, e) + \phi(e, b).$$

It should be obvious that this transition system is deterministic: if $\mu$ enables $e$ then the next marking $\mu'$ is uniquely determined. Given a net $N=(B, E, \Phi, \mu_N)$, a sequence of transitions

$$\mu_N = \mu_0 \xrightarrow{e_1} \mu_1 \cdots \mu_{n-1} \xrightarrow{e_n} \mu_n \cdots$$

is called a *firing sequence* for $N$. A marking $\mu$ is called *safe* if $\forall b : \mu(b) \leqslant 1$. A marked net is safe if all its reachable markings are safe.

Now the nets obtained as interpretations of FCCS terms may be shown to satisfy some simple structural properties. Let $\mathcal{N}(p)=(B_p, E_p, \Phi_p, \mu_p)$ be the net corresponding to some term $p$. Then it is easy to check that $\mathcal{N}(p)$ satisfies the following:

*Property* 5.1. The initial marking $\mu_p$ is safe: $\forall b \in B_p : \mu_p(b) \leqslant 1$.

*Property* 5.2. There are no self-loops: $e \in E_p \Rightarrow \mathbf{pre}(e) \cap \mathbf{post}(e) = \varnothing$.

*Property* 5.3. A postcondition is not initially marked:

$$\forall b \in B_p : \mathbf{pre}(b) \neq \varnothing \Rightarrow \mu_p(b) = 0.$$

Looking at the definition of $\mathcal{N}(p)$, it is easy to see that the only operator introducing backward branching for places is parallel composition, when it connects a communication to the postconditions of its components. But then the communication is also connected—in a symmetric way—to the preconditions of its components (by definition of $\mathcal{N}(p)$, any event has at least one precondition). As a consequence we have the following:

*Property* 5.4. If two events share a postcondition they also share a precondition:

$$\mathbf{post}(e) \cap \mathbf{post}(e') \neq \varnothing \Rightarrow \mathbf{pre}(e) \cap \mathbf{pre}(e') \neq \varnothing.$$

Note that two communications are brought to share a postcondition whenever they have a common component. In this case the two communications will also share the preconditions of their common component. This implies in particular that there is no fixed bound to the number of pre-events (and post-events) of a place. Consider, for example, the net for $(\alpha \parallel \bar{\alpha} \parallel \cdots \parallel \bar{\alpha})$: here the postcondition of the event labelled $\alpha$ is a postcondition of all the possible communications on $\alpha, \bar{\alpha}$.

The above properties may be used to show that for any $p$ the net $\mathcal{N}(p)$ is a *safe flow* net. The proof of this result is based on a characterisation of safe flow nets given in [10] (Lemma 3.2.), which we recall here:

*Characterisation of Safe Flow Nets* (from [10]). A net $= (B, E, \Phi, \mu_N)$ is a safe flow net if and only if for any condition $b$ and for any firing sequence $\mu_0 = \mu_N \xrightarrow{e_1} \mu_1 \cdots \mu_{n-1} \xrightarrow{e_n} \mu_n$ the following property is satisfied: $\mu_0(b) + \sum_{1 \leqslant i \leqslant n} \phi(e_i, b) \leqslant 1$.

To prove that $\mathcal{N}(p)$ satisfies this property we will use the following lemma:

LEMMA 5.5. *Take* $\mathcal{N}(p) = (B_p, E_p, \Phi_p, \mu_p)$. *Let* $\mu_0 = \mu_p$. *For any finite firing sequence of* $\mathcal{N}(p)$,

$$\mu_0 \xrightarrow{e_1} \mu_1 \cdots \mu_{n-1} \xrightarrow{e_n} \mu_n,$$

*the markings* $\mu_0, \ldots, \mu_n$ *are safe and if* $b \in \mathbf{post}(e_n)$ *then* $b$ *is not marked at any of* $\mu_0, \ldots, \mu_{n-1}$. *That is,*

$$\forall b, \forall i \leqslant n: \mu_i(b) \leqslant 1 \quad \textit{and} \quad \phi(e_i, b) = 1 \Rightarrow \sum_{0 \leqslant j < i} \mu_j(b) = 0. \qquad (*)$$

*Proof.* By contradiction. Suppose $(*)$ false and let $k$ be the smallest index falsifying it. This means that there exists a place $b$ such that either $\mu_k(b) > 1$ or $\phi(e_k, b) = 1$ and $\sum_{0 \leqslant j < k} \mu_j(b) \neq 0$.

We show that the first case implies the second. Then we show that the second case leads to a contradiction.

*First Case.* $\mu_k(b) > 1$. Then $k \neq 0$ by Property 5.1, and $\mu_{k-1}(b) \leqslant 1$ implies $b \in \mathbf{post}(e_k)$. In fact it can only be $\mu_k(b) = 2$ and $\mu_{k-1}(b) = 1$. But then we fall in the second case, since $\phi(e_k, b) = 1$ and $\sum_{0 \leqslant j < k} \mu_j(b) \geqslant \mu_{k-1}(b) = 1$.

*Second Case.* Let $\phi(e_k, b) = 1$ and $\sum_{0 \leqslant j < k} \mu_j(b) \neq 0$. This means that $b$ is already marked before the firing of $e_k$. Let $\mu_i$ be the first marking such that $\mu_i(b) = 1$. Then $0 < i < k$: note that $i \neq 0$ by Property 5.3, since $b \in \mathbf{post}(e_k)$. Hence $b \in \mathbf{post}(e_i)$. Thus $e_i$ and $e_k$ have a postcondition in common. But then, by Property 5.4, $e_i$ and $e_k$ have also a precondition in common, say $b'$. Now $\mu_{i-1}$ is safe by hypothesis, hence $\mu_{i-1}(b') = 1$, whence by Property 5.2 $\mu_i(b') = 0$. Then in order for $b'$ to be marked again at $\mu_{k-1}$, it must be the postcondition of some event $e_j$, $i < j < k$. But this contradicts the hypothesis that for $j < k$ no postcondition of $e_j$ is marked at any of $\mu_0, \ldots, \mu_{j-1}$. ∎

PROPOSITION 5.6. *For any* FCCS *term p, the net* $\mathcal{N}(p)$ *is a safe flow net.*

*Proof.* We want to show that for any place $\mu_0 \xrightarrow{e_1} \mu_1 \cdots \mu_{n-1} \xrightarrow{e_n} \mu_n$ the following is satisfied:

$$\mu_0(b) + \sum_{1 \leqslant i \leqslant n} \phi(e_i, b) \leqslant 1. \qquad (**)$$

If the place $b$ is initially marked, then $(**)$ is easily derived by observing that $\mu_0(b) = 1$, since $\mu_0$ is safe (Property 5.1), and then $\sum_{1 \leqslant i \leqslant n} \phi(e_i, b) = 0$ by Property 5.3.

Otherwise $b$ is not initially marked. Then if no event $e_i$ has $b$ as a postcondition, $(**)$ is trivially satisfied. Suppose now that $b$ is the postcondition of some event $e_i$. Let $j$ be the greatest index in the sequence such that $b$ is a postcondition of $e_j$. We then know that $\sum_{1 \leqslant i < j} \mu_i(b) = 0$ by the previous lemma, whence we may also deduce that $\sum_{1 \leqslant i < j} \phi(e_i, b) = 0$. Because of the way we chose $j$ we have $\sum_{j < i \leqslant n} \phi(e_i, b) = 0$, whence we conclude

$$\sum_{1 \leqslant i \leqslant n} \phi(e_i, b) = \sum_{1 \leqslant i \leqslant n,\, i \neq j} \phi(e_i, b) + \phi(e_j, b) = 1. \quad ∎$$

We may define now our third semantics for FCCS:

DEFINITION (Flow Net Semantics). A net computation of the FCCS term $p$ is a set of events $X = \{e_1, \ldots, e_n\}$ occurring in a firing sequence $\mu_0 = \mu_p \xrightarrow{e_1} \mu_1 \cdots \mu_{n-1} \xrightarrow{e_n} \mu_n$ of $\mathcal{N}(p)$. The set of these computations in $\mathscr{C}_{\text{net}}(p)$, and the domain of net computations of $p$ is $\mathscr{D}_{\text{net}}(p) = (\mathscr{C}_{\text{net}}(p), \subseteq)$.

An important property of flow nets, as well as of the safe nets of [26], is that for any firing sequence one can deduce a partial order of *causality* directly from the structure of the net: the (immediate) causality is given by the presence of a condition between two events (cf. [10]). As a consequence, any computation may be regarded here, as in our permutation semantics, as a partially ordered set of events.

## 6. Equivalence of the Three Semantics

In this section we undertake to compare our three interpretations for CCS, restricting ourselves to FCCS when dealing with the flow net semantics. We will show that the three models—proved transition system, flow event structures, and flow nets—are all *abstractly* equivalent, in that they yield isomorphic domains of computations for any (fixpoint-free) CCS term. For the flow models—flow event structures and flow nets—we will establish a stronger equivalence result, namely that they determine the same families of computations for any term. In other words, we will establish the following:

THEOREM. *For any* FCCS *term* $p$ *the flow event structure* $\mathcal{S}(p)$ *and the net* $\mathcal{N}(p)$ *have the same computations, that is,* $\mathcal{D}_{\text{es}}(p) = \mathcal{D}_{\text{net}}(p)$. *Moreover, for any* CCS *term* $p$, *the domains* $\mathcal{D}_{\text{es}}(p)$ *and* $\mathcal{D}_{\text{per}}(p)$ *are isomorphic.*

The reader may have noted the analogy between the two flow models: they both give global representations for terms, modelling occurrences of actions in a term as events—and using the same syntactic names for these events. Compared to them, the proved transition system stands a little aside: it appears as a purely operational model where terms are described step by step, as generating initial occurrences of actions.

In fact, although we know that in the three models computations may be seen as posets of occurrences, it is not immediately clear how to relate the proved transition system to the other two models, which appear somehow more "denotational." Consider for example the flow net semantics. One simple way to relate it to the proved transition system would be show a correspondence, for any term $p$, between the proved transitions of $p$ and the transitions on markings in the net $\mathcal{N}(p)$. An immediate problem, detected by Degano *et al.* [17] and Olderog [45], is that a reachable marking in the net is not necessarily the marking $\mu_p$ determined by a term, the typical example being

$$\mu((a \parallel b) + c) \xrightarrow{+_0 \|_0 a} \{+_0 \|_0 \hat{a}.\mathbf{nil} + c.\mathbf{nil})\}.$$

Here the resulting marking is somehow "mixed," since the part $+_0 \|_0 \hat{a}.\mathbf{nil}$ indicates that a choice has been passed, while in the part $(\|_1 b.\mathbf{nil} + c.\mathbf{nil})$ the choice is still present.

Here we will get over this problem by generalising the proved transition system so as to make it record the *history* of previous occurrences. We then obtain a very concrete transition system, which records CCS transitions while keeping track of the whole structure of a term. This system is called the *event transition system* or simply the *event system*, because its labels are now occurrences with a "past"—and thus general events. Then we will get a fourth semantics for CCS by defining a permutation equivalence on the event system.

The event system for a (fixpoint-free) term $p$ may be easily related to the transition system on markings of $\mathcal{N}(p)$: it turns out that the two transition systems are *isomorphic*. In particular the same labels—i.e., the same events—occur along their transition sequences. Moreover, we shall see that the prefix ordering of event transition sequences, up to permutations, coincides with the inclusion of the generated sets of events. Therefore the event system semantics and the flow net semantics determine the same families of computations. Similarly, the event system for $p$ bears a direct correspondence to the flow event structure interpretation of $p$: the computations of the event system coincide, as sets of events, with the configurations of the event structure. Here again, the two models give rise to the same families of computations. A consequence of these two results is that the three event-based models—flow event structures, flow nets, and event system—yield the same families of computations, and thus are *concretely* equivalent.

We also have to show that the event system is equivalent to the original proved transition system. The first step of comparison will be to establish a bijective correspondence between proved and event transition sequences. We shall then show that two proved transitions sequences are permutation equivalent if and only if the same is true of the corresponding event transition sequences. As a result, we will know that the domains of computations have the "same elements." It is then easy to show that the domains also have the same orderings, and thus are isomorphic. In other words, the proved transition system and the event system will be shown to be abstractly equivalent. As a consequence, the proved system is also abstractly equivalent to the two flow models. This will establish our result.

Some similar results are worth mentioning here: Best and Devillers have established the correspondence between the equivalence by permutations for firing sequences and processes of Petri nets [4]—see also the work by Degano *et al.* [21]. Also, the relation between trace semantics and a (prime) event structure semantics for a basic class of nets has been presented by Thiagarajan in [54].

6.1. *Event System*

The intermediate transition system is defined on a new set of terms representing partially executed CCS processes, which we shall call *marked terms*. The syntax of marked terms is as follows,

$$\xi ::= p \mid \hat{a}.\xi \mid (\xi \parallel \xi') \mid (\xi \hat{+}_0 p) \mid (p \hat{+}_1 \xi) \mid \xi\backslash\alpha,$$

where $p$ is any (closed) CCS term. Note that for CCS terms this grammar is somewhat ambiguous: for instance, $(p \parallel q)$ and $p\backslash\alpha$ may be regarded as marked terms. This will be used in the proofs of the results concerning marked terms.

Marked terms keep track of the *dynamic* operators of a process as well as the static ones: the construct $\hat{a}.\xi$ means that a guard $a.p$ has been passed, while $(\xi \hat{+}_0 q)$ and $(p \hat{+}_1 \xi)$ mean that a choice has been made, respectively to the left and to the right of a sum. Thus marked terms keep the whole structure of a term while recording an execution. In particular it will always be possible starting from a marked term, to reconstruct the CCS term from which it is derived.

We now define transitions of the form $\xi \xrightarrow{e}\!\!\!\!\rightarrow \xi'$ on marked terms. These transitions will be labelled by events, rather than mere proof terms, having exactly the same syntactic names as in flow event structures and flow nets. For this reason the corresponding transition system is called an *event transition system*—or simply *event system*. The inference rules for event transitions are given in the following table:

A0′. $\quad \vdash a.p \xrightarrow{a}\!\!\!\!\rightarrow \hat{a}.p$

A1′. $\quad \xi \xrightarrow{e}\!\!\!\!\rightarrow \xi' \vdash \hat{a}.\xi \xrightarrow{\hat{a}.e}\!\!\!\!\rightarrow \hat{a}.\xi'$

P0′. $\quad \xi \xrightarrow{e}\!\!\!\!\rightarrow \xi' \vdash (\xi \parallel \xi'') \xrightarrow{\|_0 e}\!\!\!\!\rightarrow (\xi' \parallel \xi'')$

P1′. $\quad \xi \xrightarrow{e}\!\!\!\!\rightarrow \xi' \vdash (\xi'' \parallel \xi) \xrightarrow{\|_1 e}\!\!\!\!\rightarrow (\xi'' \parallel \xi')$

C′. $\quad \xi_0 \xrightarrow{e}\!\!\!\!\rightarrow \xi_0', \xi_1 \xrightarrow{e'}\!\!\!\!\rightarrow \xi_1', \overline{\ell(e)} = \ell(e') \vdash (\xi_0 \parallel \xi_1) \xrightarrow{(e,e')}\!\!\!\!\rightarrow (\xi_0' \parallel \xi_1')$

S0′. $\quad p \xrightarrow{e}\!\!\!\!\rightarrow \xi \vdash (p+q) \xrightarrow{+_0 e}\!\!\!\!\rightarrow (\xi \hat{+}_0 q)$

S1′. $\quad q \xrightarrow{e}\!\!\!\!\rightarrow \xi \vdash (p+q) \xrightarrow{+_1 e}\!\!\!\!\rightarrow (p \hat{+}_1 \xi)$

ST0′. $\quad \xi \xrightarrow{e}\!\!\!\!\rightarrow \xi' \vdash (\xi \hat{+}_0 q) \xrightarrow{+_0 e}\!\!\!\!\rightarrow (\xi' \hat{+}_0 q)$

ST1′. $\quad \xi \xrightarrow{e}\!\!\!\!\rightarrow \xi' \vdash (p \hat{+}_1 \xi) \xrightarrow{+_1 e}\!\!\!\!\rightarrow (p \hat{+}_1 \xi')$

R′. $\quad \xi \xrightarrow{e}\!\!\!\!\rightarrow \xi', \mathbf{nm}(\ell(e)) \neq \alpha \vdash \xi\backslash\alpha \xrightarrow{\backslash\alpha e}\!\!\!\!\rightarrow \xi'\backslash\alpha$

FIX′. $\quad p[\mu x.p/x] \xrightarrow{e}\!\!\!\!\rightarrow \xi \vdash \mu x.p \xrightarrow{e}\!\!\!\!\rightarrow \xi$

The interesting rules for event transitions are A0′, S0′, S1′, which introduce the "past" constructors $\hat{a}.$, $\hat{+}_0$, $\hat{+}_1$. Rules A1′, ST0′, ST1′, allow terms to move under these constructors, or, putting it another way, transmit the "past" to future occurrences. Note that some marked terms, like $(p \hat{+}_0 q)$ or $(\hat{\alpha}.p)\backslash\alpha$, are never obtained as the target of some event transition. In what follows we shall mostly consider marked terms that are generated via event transitions starting from some CCS term. Then marked terms represent CCS processes at some state, just as marked nets represent nets at some state. In fact, when we come to relate the event system to flow nets, we shall see that marked terms correspond to *markings* in a net: more precisely, the event system will be shown to by the syntactic counterpart of flow nets, an event transition corresponding exactly to a transition between to marking in a net. The relation between event systems and flow nets will be established in Section 6.4. Let us see now some examples of event transitions.

EXAMPLE 1. The term $p = (a.b) + c$ has the following event transition sequence:

$$(a.b) + c \xrightarrow{+_0 a} \hat{a}.b \hat{+}_0 c \xrightarrow{+_0 \hat{a}.b} \hat{a}.\hat{b} \hat{+}_0 c.$$

Note that no transition can be deduced at the right of the past operator $\hat{+}_0$.

EXAMPLE 2. The term $p = (a \parallel b) + c$ has the following event transition sequence:

$$(a \parallel b) + c \xrightarrow{+_0 \parallel_0 a} (\hat{a} \parallel b) \hat{+}_0 c \xrightarrow{+_0 \parallel_1 b} (\hat{a} \parallel \hat{b}) \hat{+}_0 c.$$

Note that the two actions $a$ and $b$ give rise to symmetric events, even though action $b$ occurs after the choice has been made. Indeed, this will be shown to be a fundamental property of the event system: by looking at the names of events only, one will never be able to tell which of two concurrent event has happened first. This is also an essential difference w.r.t. the proved transition system, where one could have different sets of labels, i.e., proofs, along equivalent sequences.

Now the reader should be easily convinced that the event system is *deterministic*; i.e.,

$$\xi \xrightarrow{e} \xi' \qquad \text{and} \qquad \xi \xrightarrow{e} \xi'' \Rightarrow \xi' = \xi''.$$

The system has also the further property that any event occurring in some event transition sequence for a term $p$ corresponds uniquely to an occurrence, or a pair of occurrences for a communication, in the syntactic tree of $p$.

EXAMPLE 3. Take the term $p = (\alpha.b \parallel \bar{\alpha})$. In the two event transition sequences for $p$

$$(\alpha.b \parallel \bar{\alpha}) \xrightarrow{(\alpha,\bar{\alpha})} (\hat{\alpha}.b \parallel \hat{\alpha}) \xrightarrow{\|_0 \hat{\alpha}.b} (\hat{\alpha}\hat{b}. \parallel \hat{\bar{\alpha}})$$

$$(\alpha.b \parallel \bar{\alpha}) \xrightarrow{\|_0 \alpha} (\hat{\alpha}.b \parallel \bar{\alpha}) \xrightarrow{\|_0 \hat{\alpha}.b} (\hat{\alpha}\hat{b}. \parallel \bar{\alpha}),$$

the occurrence of $b$ is represented by the same event $\|_0 \hat{\alpha}.b$, whether it follows the communication $(\alpha, \bar{\alpha})$ or the simple event $\|_0 \alpha$.

In fact, events record the "syntactic history" of an occurrence, rather than a particular execution history. As we said, the event system is very intensional. From any state (marked term) $\xi$ of the system one may trace back the original CCS term from which $\xi$ arises, which we call the *ancestor* of $\xi$. The ancestor $\varrho(\xi)$ is defined on marked terms as follows:

$$\begin{aligned}
\varrho(p) &= p \qquad \text{for any CCS term } p \\
\varrho(\hat{a}.\xi) &= a.\varrho(\xi) \\
\varrho(\xi \parallel \xi') &= (\varrho(\xi) \parallel \varrho(\xi')) \\
\varrho(\xi \hat{+}_0 q) &= (\varrho(\xi) + q) \\
\varrho(p \hat{+}_1 \xi) &= (p + \varrho(\xi)) \\
\varrho(\xi \backslash \alpha) &= \varrho(\xi) \backslash \alpha.
\end{aligned}$$

It should be clear that if $\xi$ is the target of some event transition sequence starting from $p$ then $p$ is the ancestor of $\xi$, since $\xi \xrightarrow{e} \xi'$ implies that $\varrho(\xi') = \varrho(\xi)$. Note, however, that a marked term is not necessarily reachable from its ancestor: this is the case, for instance, for $(\hat{\alpha}.\xi)\backslash\alpha$.

We want now to show that the event transition system is equivalent to the proved transition system, in the sense that it yields the same domain of computations for any term $p$. We recall that the computations in the proved transition system are equivalence classes of sequences of transitions, ordered by the prefix ordering (cf. Section 3). We shall define the notion of *event computation* exactly in the same way. To this end, we have to introduce the notions of concurrency and permutation equivalence on the event system. The *concurrency* relation on events, denoted $\frown$ as usual, is the least symmetric relation that satisfies the following clauses, for any $e, e', e'' \in \mathscr{E}$:

$$\text{(A1)} \quad \|_0 e \frown \|_1 e'$$

$$\text{(A2)} \quad e \frown e' \quad \Rightarrow \quad \begin{cases} \|_0 e \frown (e', e'') \\ \|_1 e \frown (e'', e') \end{cases}$$

$$\text{(A3)} \quad e \frown e' \quad \Rightarrow \quad \begin{cases} \|_i e \frown \|_i e' \\ +_i e \frown +_i e' \\ \backslash_\alpha e \frown \backslash_\alpha e' \end{cases}$$

(A4) $e_0 \smile e'_0$ and $e_1 \smile e'_1 \Rightarrow (e_0, e_1) \smile (e'_0, e'_1)$

(A5) $e \smile e' \Rightarrow \hat{a}.e \smile \hat{a}.e'$.

The definition is thus formally identical to that of concurrency on proof terms, except for clause (A5), which allows concurrency to pass through a guard. On the other hand, the notion of residual is much simpler for events than for proofs, the residual of an event $e$ by a concurrent event being just the event $e$ itself. The diamond lemma for concurrent event transitions has then the following form:

LEMMA 6.1 (Diamond Lemma for Event Transitions). *Let* $t_0 = \xi \xrightarrow{e_0} \xi_0$ and $t_1 = \xi \xrightarrow{e_1} \xi_1$ *be two event transitions such that* $e_0 \smile e_1$. *Then there exists a unique term* $\bar{\xi}$ *such that* $\xi_0 \xrightarrow{e_1} \bar{\xi}$ *and* $\xi_1 \xrightarrow{e_0} \bar{\xi}$.

*Proof.* The uniqueness of $\bar{\xi}$ follows from the determinacy of the event system. We show now the existence of $\bar{\xi}$, by induction on the definition of $\smile$. We examine only two cases, the ones which make the difference with the diamond lemma for proved transitions.

(i) $e_0 = +_0 e'_0$ and $e_1 = +_0 e'_1$, with $e'_0 \smile e'_1$. Then we have two possibilities:

(a) $\xi = q_0 + q_1$ with $q_0 \xrightarrow{e'_0} \xi'_0$ and $q_0 \xrightarrow{e'_1} \xi'_1$, and thus $\xi_0 = \xi'_0 \hat{+}_0 q_1$ and $\xi_1 = \xi'_1 \hat{+}_0 q_1$. By induction we have for $e'_0, e'_1$ the closing transitions

$$\xi'_0 \xrightarrow{e'_1} \bar{\xi}' \qquad \text{and} \qquad \xi'_1 \xrightarrow{e'_0} \bar{\xi}'.$$

Then, applying rule ST0′, we get

$$\xi'_0 \hat{+}_0 q_1 \xrightarrow{+_0 e'_1} \bar{\xi}' \hat{+}_0 q_1 \qquad \text{and} \qquad \xi'_1 \hat{+}_0 q_1 \xrightarrow{+_0 e'_0} \bar{\xi}' \hat{+}_0 q_1,$$

which are the required closing transitions for $e_0, e_1$.

(b) $\xi = \xi' \hat{+}_0 q$ with $\xi' \xrightarrow{e'_0} \xi'_0$ and $\xi' \xrightarrow{e'_1} \xi'_1$, and thus $\xi_0 = \xi'_0 \hat{+}_0 q$ and $\xi_1 = \xi'_1 \hat{+}_0 q$. By induction we have for $e'_0, e'_1$ the transitions

$$\xi'_0 \xrightarrow{e'_1} \bar{\xi}' \qquad \text{and} \qquad \xi'_1 \xrightarrow{e'_0} \bar{\xi}'.$$

Then, applying rule ST0′ again, we obtain

$$\xi'_0 \hat{+}_0 q \xrightarrow{+_0 e'_1} \bar{\xi}' \hat{+}_0 q \qquad \text{and} \qquad \xi'_1 \hat{+}_0 q \xrightarrow{+_0 e'_0} \bar{\xi}' \hat{+}_0 q,$$

which are the required closing transitions.

(ii) $e_0 = \hat{a}.e_0'$, $e_1 = \hat{a}.e_1'$, with $e_0' \smile e_1'$. Then $\xi = \hat{a}.\xi'$ with $\xi' \xrightarrow{e_0'} \xi_0'$, $\xi' \xrightarrow{e_1'} \xi_1'$, and $\xi_0 = \hat{a}.\xi_0'$, $\xi_1 = \hat{a}.\xi_1'$. By induction we have for $e_0', e_1'$ the transitions

$$\xi_0' \xrightarrow{e_1'} \bar{\xi}' \qquad \text{and} \qquad \xi_1' \xrightarrow{e_0'} \bar{\xi}'.$$

Then, applying rule A1', we get

$$\hat{a}.\xi_0' \xrightarrow{\hat{a}.e_1'} \hat{a}.\bar{\xi}' \qquad \text{and} \qquad \hat{a}.\xi_1' \xrightarrow{\hat{a}.e_0'} \hat{a}.\bar{\xi}',$$

which are the required closing transitions for $e_0, e_1$. ∎

We may now define permutation equivalence on event sequences, in the same way as we did for proved transitions. We will use now $\sigma, \sigma', \sigma_i, \ldots$ to denote event sequences. Let $\sigma\sigma'$ denote the concatenation of $\sigma$ and $\sigma'$, which is only defined if the source of $\sigma'$ is the target of $\sigma$. Permutation equivalence is as usual the congruence generated by the diamond lemma:

DEFINITION (Permutation Equivalence on Event Sequences). Let $\xi$ be a marked term. The equivalence by permutations on event transitions of $\xi$ is the least equivalence $\simeq$ *s.t.*

$$\sigma_0 t_0 (t_1/t_0)\, \sigma_1 \simeq \sigma_0 t_1 (t_0/t_1)\, \sigma_1,$$

provided that $t_0 \smile t_1$, and that concatenation is defined.

Note that there are events which are "syntactically" concurrent, according to our definition, and yet can never be permuted because there is no event transition sequence where they are adjacent. Let us see an example.

EXAMPLE 4. In the term $r = (a.\alpha \parallel \bar{\alpha}.b)\backslash\alpha$ (see Section 4 for its event structure representation), the events $\backslash_\alpha \|_0 a$ and $\backslash_\alpha \|_1 \hat{\bar{\alpha}}.b$ are concurrent. However, they can never be permuted, since in the unique event sequence where they both occur they are not consecutive:

$$(a.\alpha \parallel \bar{\alpha}.b)\backslash\alpha \xrightarrow{\backslash_\alpha \|_0 a} (\hat{a}.\alpha \parallel \bar{\alpha}.b)\backslash\alpha \xrightarrow{\backslash_\alpha (\hat{a}.\alpha,\, \bar{\alpha})} (\hat{a}.\alpha \parallel \hat{\bar{\alpha}}.b)\backslash\alpha$$
$$\xrightarrow{\backslash_\alpha \|_1 \hat{\bar{\alpha}}.b} (\hat{a}.\alpha \parallel \hat{\bar{\alpha}}\hat{b}.)\backslash\alpha.$$

Note that this is indeed desirable since the two events are intuitively causally related in this computation.

Finally we may introduce our last semantics for CCS terms: this is the permutation semantics with respect to the event system. In fact we define this semantics for marked terms, denoting by $\mathscr{T}_{ev}(\xi)$ the set of sequences of event transitions of $\xi$:

DEFINITION (Event System Semantics). The set $\mathscr{C}_{ev}(\xi)$ of event computations of the marked term $\xi$ is the set $\mathscr{T}_{ev}(\xi)/\simeq$ of equivalence classes of sequences of event transitions of $\xi$. The event computations of $\xi$ are ordered by

$$[\![\sigma]\!] \sqsubseteq [\![\sigma']\!] \Leftrightarrow_{\text{def}} \exists \sigma''.\, \sigma' \simeq \sigma\sigma''.$$

The domain of event computations of $\xi$ is $\mathscr{D}_{ev}(\xi) = (\mathscr{C}_{ev}(\xi), \sqsubseteq)$.

For any sequence $\sigma \in \mathscr{T}_{ev}(\xi)$, that is,

$$\sigma = \xi \xrightarrow{e_1} \xi_1 \cdots \xi_{n-1} \xrightarrow{e_n} \xi_n,$$

let us denote by $\mathrm{Ev}(\sigma)$ the set of events occurring in $\sigma$, that is, $\mathrm{Ev}(\sigma) = \{e_1, \ldots, e_n\}$. Then it should be obvious that the following holds:

*Remark* 6.2. Let $\sigma_0, \sigma_1$ be event transition sequences for some marked term $\xi$. Then

$$[\![\sigma_0]\!] \sqsubseteq [\![\sigma_1]\!] \Rightarrow \mathrm{Ev}(\sigma_0) \subseteq \mathrm{Ev}(\sigma_1).$$

6.2. *Correspondence between Event System and Proved Transition System*

We show in this section that event and proved transitions systems determine isomorphic domains of computations. We start by giving constructions between the two kinds of transition sequences. Looking at the table of inference rules, it should be clear that event transitions $\xi \xrightarrow{e} \xi'$ contain in some sense the proved transitions $p \xrightarrow{\theta} p'$. We recall that in the proved system labels are proofs, that is events built without the constructors $\hat{a}.$. Intuitively, it should be possible to reduce or "flatten" any event transition $\xi \xrightarrow{e} \xi'$ to a proved transition of the form $p \xrightarrow{\theta} p'$, by just forgetting the history information recorded in $\xi, e, \xi'$.

Let us now formalise how an event transition $\xi \xrightarrow{e}$ may be *flattened* to a proved transition. First a marked term $\xi$ may be reduced to a CCS term $\varphi(\xi)$ with the same "future," by just dropping the past operators in $\xi$. In particular, if the marked term $\xi$ is the target of some event transition sequence of a CCS term $p$, then $\varphi(\xi)$ should be exactly what we would have obtained by applying ordinary CCS transitions to the term $p$. We define $\varphi(\xi)$ inductively on marked terms as follows:

$$\begin{aligned}
\varphi(p) &= p \qquad \text{for any CCS term } p \\
\varphi(\hat{a}.\xi) &= \varphi(\xi) \\
\varphi(\xi \parallel \xi') &= (\varphi(\xi) \parallel \varphi(\xi')) \\
\varphi(\xi \hat{+}_0 q) &= \varphi(\xi) \\
\varphi(p \hat{+}_1 \xi) &= \varphi(\xi) \\
\varphi(\xi \backslash \alpha) &= \varphi(\xi) \backslash \alpha.
\end{aligned}$$

Note that, although the grammar for marked terms is ambiguous, the term $\varphi(\xi)$ is always well defined. Looking back at the rules for event transitions, we may remark that if $p \xrightarrow{e}\!\!\!\!\rightarrow \xi'$, where $p$ is a CCS term, then $e$ is a proof term (i.e., $e \in \Theta$). Then for an event transition of the form $p \xrightarrow{\theta}\!\!\!\!\rightarrow \xi'$ the flattening is straightforward:

LEMMA 6.3. *For any CCS term* $p$: $p \xrightarrow{\theta}\!\!\!\!\rightarrow \xi \Rightarrow p \xrightarrow{\theta}\!\!\!\!\rightarrow \varphi(\xi)$.

*Proof.* By induction on the proof of $p \xrightarrow{\theta}\!\!\!\!\rightarrow \xi$. Note that rules A1′, ST0′, ST1′ do not apply to CCS terms. Let us examine some of the other rules.

(A0′) Then the transition is $a.p' \xrightarrow{a}\!\!\!\!\rightarrow \hat{a}.p'$, and by the Rule A for proved transitions we get $a.p' \xrightarrow{a}\!\!\!\!\rightarrow p'$, where $p' = \varphi(\hat{a}.p')$.

(S0′) Here the transition is $(p+q) \xrightarrow{+_0\theta}\!\!\!\!\rightarrow (\xi' \hat{+}_0 q)$ with $p \xrightarrow{\theta}\!\!\!\!\rightarrow \xi'$. By induction $p \xrightarrow{\theta}\!\!\!\!\rightarrow \varphi(\xi')$; therefore by S0 we have $(p+q) \xrightarrow{+_0\theta}\!\!\!\!\rightarrow \varphi(\xi')$, which is the required transition, since $\varphi(\xi') = \varphi(\xi' \hat{+}_0 q)$. ∎

For a general transition $\xi \xrightarrow{e}\!\!\!\!\rightarrow \xi'$ we need a second function $\psi$ to reduce the event $e$ to a proof term. However, $\psi$ cannot be simply a function of $e$, as shown by the example:

EXAMPLE 2 (Continued). We saw already that $p = (a \parallel b) + c$ has the event transition sequence

$$p \xrightarrow{+_0 \|_0 a}\!\!\!\!\rightarrow (\hat{a} \parallel b) \hat{+}_0 c \xrightarrow{+_0 \|_1 b}\!\!\!\!\rightarrow (\hat{a} \parallel \hat{b}) \hat{+}_0 c.$$

Now the corresponding proved sequence should be

$$s = p \xrightarrow{+_0 \|_0 a}\!\!\!\!\rightarrow (\mathbf{nil} \parallel b) \xrightarrow{\|_1 b}\!\!\!\!\rightarrow (\mathbf{nil} \parallel \mathbf{nil}).$$

We can see that in order to obtain the proved sequences $s$, we should retain the constructor $+_0$ when flattening the first event, whereas we should drop it in the second event. Intuitively, this is because in the first event the $+_0$ represents a choice occurring during the current transition, while in the second event the $+_0$ refers to a choice that occurred sometime in the past and was passed on by a term $(\xi \hat{+}_0 q)$. Now, to be able to distinguish the two cases, we need to look at the structure of the marked term performing the event.

Hence $\psi$ must be a function of $\xi$ as well as $e$. In fact, we shall specify $\psi$ as a partial function, which will be defined in particular for pairs $(\xi, e)$ such that $\xi \xrightarrow{e}\!\!\!\!\rightarrow \xi'$ for some $\xi'$. The definition of the proof term $\psi(\xi, e)$ corresponding to the event $e$ is as follows:

$$\psi(p, \theta) = \theta \qquad \text{for any CCS term } p \text{ and } \theta \in \Theta$$

$$\psi(\hat{a}.\xi, \hat{a}.e) = \psi(\xi, e)$$

$$\psi((\xi_0 \parallel \xi_1), \|_i e) = \|_i \psi(\xi_i, e)$$

$$\psi((\xi \parallel \xi'), (e, e')) = (\psi(\xi, e), \psi(\xi', e'))$$

$$\psi(\xi \hat{+}_0 q, +_0 e) = \psi(\xi, e)$$

$$\psi(p \hat{+}_1 \xi, e) = \psi(\xi, e)$$

$$\psi(\xi \backslash \alpha, \backslash_\alpha e) = \backslash_\alpha \psi(\xi, e).$$

*Remark.* Note that $\psi(\xi, e)$ is well-defined: if $\xi$ is a CCS term $p$, we have $e \in \Theta$ and $\psi(p, e) = e$. Moreover, if $\psi(\xi, e) = \psi(\xi, e')$ then $e = e'$.

Using functions $\varphi$ and $\psi$ we may now define the *flattening* $t^\vee$ of an event transition $t$:

LEMMA 6.4 (Transition Flattening). *Given an event transition* $t = \xi \xrightarrow{e}\!\!\!\!\rightarrow \xi'$, *let us define the flattening* $t^\vee$ *of* $t$ *by* $t^\vee = \varphi(\xi) \xrightarrow{\psi(\xi, e)}\!\!\!\!\rightarrow \varphi(\xi')$. *Then* $t^\vee$ *is a proved transition.*

By induction on the proof of the transition $t$. If $t$ is a transition of a CCS term we have the result by the previous lemma. So we do not need to consider rules A0′, S0′, S1′. We examine some of the other cases.

(A1′) If $t$ is $\hat{a}.\xi \xrightarrow{\hat{a}.e}\!\!\!\!\rightarrow \hat{a}.\xi'$ with $\xi \xrightarrow{e}\!\!\!\!\rightarrow \xi'$, by induction the flattening $\varphi(\xi) \xrightarrow{\psi(\xi, e)} \varphi(\xi')$ is a proved transition. Then

$$\varphi(\hat{a}.\xi) \xrightarrow{\psi(\hat{a}.\xi, \hat{a}.e)} \varphi(\hat{a}.\xi')$$

gives the same proved transition, since $\varphi(\hat{a}.\xi) = \varphi(\xi)$, $\psi(\hat{a}.\xi, \hat{a}.e) = \psi(\xi, e)$, and $\varphi(\hat{a}.\xi') = \varphi(\xi')$.

(C′) If $t$ is $(\xi_0 \parallel \xi_1) \xrightarrow{(e_0, e_1)}\!\!\!\!\rightarrow (\xi_0' \parallel \xi_1')$ with $\xi_0 \xrightarrow{e_0}\!\!\!\!\rightarrow \xi_0'$ and $\xi_1 \xrightarrow{e_1}\!\!\!\!\rightarrow \xi_1'$, then by induction, for $i = 0, 1$, $\varphi(\xi_i) \xrightarrow{\psi(\xi_i, e_i)} \varphi(\xi_i')$ is a proved transition. Therefore the flattening

$$\varphi(\xi_0 \parallel \xi_1) \xrightarrow{\psi((\xi_0 \parallel \xi_1), (e_0, e_1))} \varphi(\xi_0' \parallel \xi_1')$$

is also a proved transition, since

$$\varphi(\xi_0 \parallel \xi_1) = (\varphi(\xi_0) \parallel \varphi(\xi_1))$$

$$\psi((\xi_0 \parallel \xi_1), (e_0, e_1)) = (\psi(\xi_0, e_0), \psi(\xi_1, e_1))$$

$$\varphi(\xi_0' \parallel \xi_1') = (\varphi(\xi_0') \parallel \varphi(\xi_1'))$$

and by rule C we deduce the proved transition

$$(\varphi(\xi_0)\| \varphi(\xi_1)) \xrightarrow{(\psi(\xi_0, e_0), \psi(\xi_1, e_1))} (\varphi(\xi_0')\| \varphi(\xi_1')).$$

(ST0′) If $t$ is $(\xi \hat{+}_0 q) \xrightarrow{+_0 e} (\xi' \hat{+}_0 q)$ with $\xi \xrightarrow{e} \xi'$, by induction $\varphi(\xi) \xrightarrow{\psi(\xi, e)} \varphi(\xi')$ is a proved transition. Then the flattening

$$\varphi(\xi \hat{+}_0 q) \xrightarrow{\psi(\xi \hat{+}_0 q, +_0 e)} \varphi(\xi' \hat{+}_0 q)$$

yields the same proved transition, since $\varphi(\xi \hat{+}_0 q) = \varphi(\xi)$, $\psi(\xi \hat{+}_0 q, +_0 e) = \psi(\xi, e)$, and $\varphi(\xi' \hat{+}_0 q) = \varphi(\xi')$. ∎

It is straightforward to extend this flattening to sequences of event transitions:

PROPOSTION 6.5 (Sequence Flattening). *Let* $\sigma = \xi_0 \xrightarrow{e_1} \xi_1 \cdots \xi_{n-1} \xrightarrow{e_n} \xi_n$ *be an event transition sequence for* $\xi_0$. *Then*

$$\sigma^{\vee} = \varphi(\xi_0) \xrightarrow{\psi(\xi_0, e_1)} \varphi(\xi_1) \cdots \varphi(\xi_{n-1}) \xrightarrow{\psi(\xi_{n-1}, e_n)} \varphi(\xi_n)$$

*is a proved sequence for* $\varphi(\xi_0)$. *We call* $\sigma^{\vee}$ *the flattening of* $\sigma$.

We want now to show the converse, namely that each proved sequence $s$ of a term $p$ may be *lifted* to an event transition sequence $s^{\wedge}$ for $p$. In fact, it will be conventient to define the lifting of a proved sequence of $p$ starting from any $\xi$ such that $\varphi(\xi) = p$. The first step is lifting a proved transition of $p$ to an event transition of $p$.

LEMMA 6.6. *For any* CCS *term and proved transition* $p \xrightarrow{\theta} p'$ *there exists a unique marked term* $\xi$ *such that* $p \xrightarrow{\theta} \xi$ *is an event transition. Moreover* $\varphi(\xi) = p'$.

*Proof.* The uniqueness of $\xi$ results from the fact that the event system is deterministic. The existence of $\xi$, and the fact that $\varphi(\xi) = p'$, is proved by induction on the inference of the transition $p \xrightarrow{\theta} p'$. Let us just see some cases:

(i) If the transition is inferred using the rule A, then $p = a.p'$ and $\theta = a$. We let $\xi = \hat{a}.p'$. Then $p \xrightarrow{\theta}$ by A0′, and obviously $\varphi(\xi) = p'$.

(ii) If $p = p_0 + p_1$ and $\theta = +_0 \theta'$ with $p_0 \xrightarrow{\theta'} p'$, then by induction there exists $\xi'$ such that $p_0 \xrightarrow{\theta'} \xi'$ with $\varphi(\xi') = p'$. Then we let $\xi = \xi' \hat{+}_0 p_1$. We have $p \xrightarrow{\theta} \xi$ by S0′, and obviously $\varphi(\xi) = p'$.

(iii) If $p = \mu x.q$ and $q[\mu x.q/x] \xrightarrow{\theta} p'$ then we use the induction hypothesis and FIX′ to conclude. ∎

Now we show how to lift a proved transition $t$ of $p$ to an event transition $t^{\wedge}(\xi)$ when $\varphi(\xi)=p$:

LEMMA 6.7 (Transition Lifting). *Let $p$ be a* CCS *term and $t=p \xrightarrow{\theta} p'$ be a proved transition of $p$. Then for any marked term $\xi$ such that $\varphi(\xi)=p$, there exists a unique event transition $\xi \xrightarrow{e} \xi'$ such that $\psi(\xi, e)=\theta$ and $\varphi(\xi')=p'$. We call this transition, denoted $t^{\wedge}(\xi)$, the lifting of $t$ from $\xi$.*

*Proof.* For the uniqueness of the lifting $t^{\wedge}(\xi)$, assume $\xi \xrightarrow{e} \xi'$ and $\xi \xrightarrow{e'} \xi''$ with $\psi(\xi, e)=\theta=\psi(\xi, e')$ and $\varphi(\xi')=p'=\varphi(\xi'')$. Then, by the remark above, we have $e=e'$, and thus also $\xi'=\xi''$, since the event system is deterministic.

We show now the existence of $t^{\wedge}(\xi)$ by induction on the structure of $\xi$. We only examine some cases.

(i) If $\xi$ is a CCS term $p$, then the lifting $t^{\wedge}(\xi)$ is given by the previous lemma.

(ii) If $\xi=\hat{a}.\xi'$ then $\varphi(\xi')=\varphi(\xi)=p$, and by induction $\xi' \xrightarrow{e'} \xi''$, with $\psi(\xi', e')=\theta$ and $\varphi(\xi'')=p'$. Then by rule A1′ we obtain $\hat{a}.\xi' \xrightarrow{\hat{a}.e'} \hat{a}.\xi''$, satisfying $\psi(\hat{a}.\xi', e')=\psi(\xi', e')=\theta$ and $\varphi(\hat{a}.\xi'')=\varphi(\xi'')=p'$.

(iii) If $\xi=\xi_0 \hat{+}_0 q$ then $\varphi(\xi_0)=\varphi(\xi)=p$, whence by induction $\xi_0 \xrightarrow{e_0} \xi_0'$, with $\psi(\xi_0, e_0)=\theta$ and $\varphi(\xi_0')=p'$. By rule ST0′ we then have $\xi \xrightarrow{e} \xi'$, with $e=+_0 e_0$ and $\xi'=\xi_0' \hat{+}_0 q$ satisfying $\psi(\xi, e)=\psi(\xi_0, e_0)=\theta$ and $\varphi(\xi')=\varphi(\xi_0')=p'$.

(iv) If $\xi=(\xi_0 \parallel \xi_1)$ then we have $p=(p_0 \parallel p_1)$, with $\varphi(\xi_0)=p_0$, $\varphi(\xi_1)=p_1$. There are three possibilities for the transition $p \xrightarrow{\theta} p'$. We consider the cases $\theta=\|_0 \theta_0$ and $\theta=(\theta_0, \theta_1)$.

(a) If $\theta=\|_0 \Theta_0$ with $p_0 \xrightarrow{\theta_0} p_0'$, by induction $\xi_0 \xrightarrow{\theta_0} \xi_0'$, with $\psi(\xi_0, e_0)=\theta_0$ and $\varphi(\xi_0')=p_0'$. Now by P0′ we have $(\xi_0 \parallel \xi_1) \xrightarrow{\|_0 e_0} (\xi_0' \parallel \xi_1)$, with

$$\psi((\xi_0 \parallel \xi_1), \|_0 e_0)=\|_0(\psi(\xi_0, e_0))=\|_0 \theta_0=\theta$$

$$\varphi(\xi_0' \parallel \xi_1)=(\varphi(\xi_0') \parallel \varphi(\xi_1))=(p_0' \parallel p_1).$$

(b) If $\theta=(\theta_0, \theta_1)$ with $p_0 \xrightarrow{\theta_0} p_0'$ and $p_1 \xrightarrow{\theta_1} p_1'$, then by induction $\xi_0 \xrightarrow{\theta_0} \xi_0'$, $\xi_1 \xrightarrow{\theta_1} \xi_1'$, with $\psi(\xi_i, e_i)=\theta_i$ and $\varphi(\xi_i')=p_i'$. Then by rule C we get $(\xi_0 \parallel \xi_1) \xrightarrow{(e_0, e_1)} (\xi_0' \parallel \xi_1')$. Here

$$\psi((\xi_0 \parallel \xi_1), (e_0, e_1))=(\psi(\xi_0, e_0), \psi(\xi_1, e_1))=(\theta_0, \theta_1)$$

and $\varphi(\xi_0' \parallel \xi_1')=(p_0' \parallel p_1')$. ∎

Lifting a proved transition sequence is slightly more involved than flattening an event transition sequence: the lifting of a step $p_i \xrightarrow{e_i} p_{i+1}$ of a proved transition sequence depends on the particular $\xi_i$ obtained by lifting the segment $p_0 \xrightarrow{\theta_1} p_1 \cdots p_{i-1} \xrightarrow{\theta_i} p_i$ of the sequence.

PROPOSITION 6.8 (Sequence Lifting). *Let $s = p_0 \xrightarrow{\theta_1} p_1 \cdots p_{n-1} \xrightarrow{\theta_n} p_n$ be a proved sequence for the term $p_0$, and of $\xi_0$ be a marked term such that $\varphi(\xi_0) = p_0$. Then there exists a unique event transition sequence $s^\wedge(\xi_0) = \xi_0 \xrightarrow{e_1} \xi_1 \cdots \xi_{n-1} \xrightarrow{e_n} \xi_n$ such that $\psi(\xi_{i-1}, e_i) = \theta_i$ and $\varphi(\xi_i) = p_i$ for $i = 1, \ldots, n$. The event transition sequence $s^\wedge(\xi_0)$ is called the lifting of $s$ from $\xi_0$.*

*Proof.* The sequence $s^\wedge(\xi_0)$ is constructed by induction on the length $n$ of $s$. The case $n = 1$ is covered by the previous lemma. Otherwise let $s = p_0 \xrightarrow{\theta_1} p_1 \cdots p_n \xrightarrow{\theta_{n+1}} p_{n+1}$. By induction the prefix $\bar{s} = p_0 \xrightarrow{\theta_1} p_1 \cdots p_{n-1} \xrightarrow{\theta_n} p_n$ may be lifted to an event transition sequence $\bar{s}^\wedge(\xi_0) = \xi_0 \xrightarrow{e_1} \xi_1 \cdots \xi_{n-1} \xrightarrow{e_n} \xi_n$. Since $\varphi(\xi_n) = p_n$, by the previous lemma the proved transition $t = p_n \xrightarrow{\theta_{n+1}} p_{n+1}$ may be lifted to an event transition $t^\wedge(\xi_n) = \xi_n \xrightarrow{e_{n+1}} \xi_{n+1}$. Then $s^\wedge(\xi_0) = \bar{s}^\wedge(\xi_0) \cdot t^\wedge(\xi_n)$ is the lifting of $s$. ∎

We may now establish the correspondence between the two permutation semantics:

THEOREM 6.9. *For any marked term $\xi$ the domains of computation $\mathscr{D}_{\text{ev}}(\xi) = \mathscr{T}_{\text{ev}}(\xi)/\simeq, \sqsubseteq)$ and $\mathscr{D}_{\text{per}}(\varphi(\xi)) = (\mathscr{T}(\varphi(\xi))/\simeq, \sqsubseteq)$ are isomorphic.*

*Proof.* Clearly sequence flattening and sequence lifting provide inverse bijections between $\mathscr{T}_{\text{ev}}(\xi)$ and $\mathscr{T}(\varphi(\xi))$. Moreover, these mappings are compatible with sequence concatenation; that is,

$$(\sigma_0 \sigma_1)^\vee = \sigma_0^\vee \, \sigma_1^\vee$$

and

$$(s_0 s_1)^\vee(\xi) = s_0^\wedge(\xi) \, s_1^\wedge(\xi'), \qquad \text{where } \xi' \text{ is the target of } s_0^\wedge(\xi).$$

It is easy to show that proof diamonds, relating two-steps proved transition sequences, are lifted to event diamonds, and conversely event diamonds are flattened to proof diamonds; therefore

$$\sigma_0 \simeq \sigma_1 \Leftrightarrow \sigma_0^\vee \simeq \sigma_1^\vee ;$$

hence

$$[\![\sigma_0]\!] \sqsubseteq [\![\sigma_1]\!] \Leftrightarrow [\![\sigma_0^\vee]\!] \sqsubseteq [\![\sigma_1^\vee]\!].$$

This shows that the domains of computations are isomorphic. ∎

To conclude this section we prove some properties of event computations that will be used in comparing the event system semantics and the flow event structure semantics. First we show that the event system is an *asynchronous system* in the sense of [1, 51, 36]; that is, it satisfies the property that any two concurrent events that may be executed in some order from a given state may also be executed from the same state in the reverse order.

PROPOSITION 6.10 (The Event System Is an Asynchronous System). *If* $\xi \xrightarrow{e} \xi' \xrightarrow{e'} \xi''$ *and* $e \frown e'$, *then* $\exists \xi'''$ *such that* $\xi \xrightarrow{e'} \xi''' \xrightarrow{e} \xi''$.

*Proof.* The proof is by induction on the definition of $\frown$ on events.

(i) Basic case: $e = \|_0 e_0$ and $e' = \|_1 e_1$. In this case $\xi$ may be unfolded into $(\xi_0 \| \xi_1)$ such that $(\xi_0 \| \xi_1) \xrightarrow{\|_0 e_0} \xi'$ is inferred from $\xi_0 \xrightarrow{e_0} \xi_0'$ by rule P0', with $\xi' = (\xi_0' \| \xi_1)$. Similarly $(\xi_0' \| \xi_1) \xrightarrow{\|_1 e_1} \xi''$ is inferred from $\xi_1 \xrightarrow{e_1} \xi_1'$ by rule P1', with $\xi'' = (\xi_0' \| \xi_1')$. But then, using the same rules in the reverse order, we deduce $(\xi_0 \| \xi_1) \xrightarrow{\|_1 e_1} (\xi_0 \| \xi_1') \xrightarrow{\|_0 e_0} (\xi_0' \| \xi_1')$, hence the result, with $\xi''' = (\xi_0 \| \xi_1')$.

(ii) If $e = \|_0 e_0$ and $e' = (e_0', e_1)$, with $e_0 \frown e_0'$, then again $\xi$ may be unfolded into $(\xi_0 \| \xi_1)$ such that $(\xi_0 \| \xi_1) \xrightarrow{\|_0 e_0} \xi'$ is inferred from $\xi_0 \xrightarrow{e_0} \xi_0'$ by rule P0', with $\xi' = (\xi_0' \| \xi_1)$. On the other hand $(\xi_0' \| \xi_1) \xrightarrow{(e_0', e_1)} \xi''$ is inferred from $\xi_0' \xrightarrow{e_0'} \xi_0''$ and $\xi_1 \xrightarrow{e_1} \xi_1'$ by rule C', with $\xi'' = (\xi_0'' \| \xi_1')$. But now, since $\xi_0 \xrightarrow{e_0} \xi_0' \xrightarrow{e_0'} \xi_0''$, by induction $\exists \xi_0'$ such that $\xi_0 \xrightarrow{e_0'} \xi_0''' \xrightarrow{e_0} \xi_0''$. Then applying rules P0' and C' to $\xi = (\xi_0 \| \xi_1)$ in the reverse order, we obtain $(\xi_0 \| \xi_1) \xrightarrow{(e_0', e_1)} (\xi_0''' \| \xi_1') \xrightarrow{\|_0 e_0} (\xi_0'' \| \xi_1')$, hence the result, with $\xi''' = (\xi_0'' \| \xi_1')$.

The other cases are treated similarly. ∎

Now we show that two consecutive events in a sequence are concurrent if the first one is not a cause of the second:

LEMMA 6.11. *If* $\xi \xrightarrow{e} \xi' \xrightarrow{e'} \xi''$ *and* $e \not\prec e'$, *then* $e \frown e'$.

*Proof.* By induction on the inference of $\xi \xrightarrow{e} \xi'$. Since $e \not\prec e'$ the case of A0' is excluded. We only examine some cases:

(i) If the transition is proved using A1' then $\xi = \hat{a}.\xi_0$ and $e = \hat{a}.e_0$ with $\xi_0 \xrightarrow{e_0} \xi_0'$ and $\xi' = \hat{a}.\xi_0'$. Then $\xi' \xrightarrow{e'} \xi''$ must be proved using A1', therefore $e' = \hat{a}.e_0'$. We have $e \not\prec e' \Rightarrow e_0 \not\prec e_0'$; hence by the induction hypothesis $e_0 \frown e_0'$; therefore $e \frown e'$.

(ii) If the transition is proved by means of P0' then $\xi = (\xi_0 \| \xi_1)$, $e = \|_0 e_0$ with $\xi_0 \xrightarrow{e_0} \xi_0'$ and $\xi' = (\xi_0' \| \xi_1)$. There are three cases:

(a) If $\xi' \xrightarrow{e'} \xi'$ is proved using P1' then $e' = \|_1 e_1$; therefore $e \frown e'$ by definition of the concurrency relation.

(b) If $\xi' \xrightarrow{e'} \xi'$ is proved using P0′ then $e' = \|_0 e_1$, and $\|_0 e_0 \prec \|_0 e_1 \Leftrightarrow e_0 \prec e_1$, therefore by induction $e_0 \smile e_1$, and thus also $e \smile e'$.

(c) If $\xi' \xrightarrow{e'} \xi'$ is proved using C′ then $e' = (e'_0, e_1)$ with $\xi'_0 \xrightarrow{e'_0} \xi''_0$. Then $e \not\prec e' \Leftrightarrow e_0 \not\prec e'_0$, therefore by induction $e_0 \smile e'_0$, and thus also $e \smile e'_0$, and thus also $e \smile e'$.

(iii) If $\xi \xrightarrow{e} \xi'$ is proved by means of FIX′ then $\xi = \mu x.p$ with $p[\mu x.p/x] \xrightarrow{e} \xi'$, and we use the induction hypothesis to conclude.

The remaining cases are left to the reader. ∎

Finally we prove that the event transition sequences of a compound term $(\xi_0 \| \xi_1)$ may be decomposed into sequences of the components. Let us define an *event sequences* of $\xi$ to be the sequence $u = e_1 \cdots e_n$ of events of an event transition sequence $\sigma = \xi \xrightarrow{e_1} \xi_1 \cdots \xi_{n-1} \xrightarrow{e_n} \xi_n$. Define then $\mathscr{S}_{\text{ev}}(\xi)$, the *set of event sequences* of $\xi$; that is,

$$e_1 \cdots e_n \in \mathscr{S}_{\text{ev}}(\xi) \Leftrightarrow \exists \xi_1, \ldots, \xi_n.\ \xi \xrightarrow{e_1} \xi_1 \cdots \xi_{n-1} \xrightarrow{e_n} \xi_n.$$

Note that $\mathscr{S}_{\text{ev}}(\xi)$ always includes the empty sequence $\varepsilon$. Define now, for a marked term of the form $\xi = (\xi_0 \| \xi_1)$, the *projections* $\pi_0(u)$, $\pi_1(u)$ of an event sequence $u$ of $\xi$ as follows:

$$\pi_0(\varepsilon) = \varepsilon$$

$$\pi_0(eu) = \begin{cases} \pi_0(u), & \text{if } e = \|_1 e' \\ e'\pi_0(u), & \text{if } e = \|_0 e' \text{ or } e = (e', e'') \end{cases}$$

and $\pi_1(u)$ is defined similarly. We may now prove the announced result:

LEMMA 6.12. *Let $\xi_0, \xi_1$ be marked terms. Then $u \in \mathscr{S}_{\text{ev}}(\xi_0 \| \xi_1)$ iff $\forall i = 0, 1 : \pi_i(u) \in \mathscr{S}_{\text{ev}}(\xi_i)$.*

*Proof.* Let $u$ be a sequence such that $\pi_i(u) \in \mathscr{S}_{\text{ev}}(\xi_i)$ for $i = 0, 1$. We show that $u \in \mathscr{S}_{\text{ev}}(\xi_0 \| \xi_1)$ by induction on the length of $u$. The statement is obvious if $u = \varepsilon$. Now suppose that $u = e \cdot v$. There are then three possibilities for $e$:

(i) If $e = \|_0 e_0$ then $\exists \xi'_0$ such that $\xi_0 \xrightarrow{e_0} \xi'_0$ and $\pi_0(v)$ is an event sequence of $\xi'_0$. From this we deduce $\xi_0 \| \xi_1 \xrightarrow{\|_0 e_0} \xi'_0 \| \xi_1$, and since $\pi_0(v) \in \mathscr{S}_{\text{ev}}(\xi'_0)$, $\pi_1(v) = \pi_1(u) \in \mathscr{S}_{\text{ev}}(\xi_1)$, by induction we have $v \in \mathscr{S}_{\text{ev}}(\xi'_0 \| \xi_1)$. It follows that $e \cdot v \in \mathscr{S}_{\text{ev}}(\xi_0 \| \xi_1)$.

(ii) The case where $e = \|_1 e_1$ is symmetric to the previous one.

(iii) If $e = (e_0, e_1)$ then $\exists \xi'_0, \xi'_1$ such that $\xi_i \xrightarrow{e_i} \xi'_i$ and $\pi_i(v)$ is an event sequence of $\xi'_i$, for $i = 0, 1$. From this we deduce that $\xi_0 \| \xi_1 \xrightarrow{(e_0, e_1)} \xi'_0 \| \xi'_1$. Then since $\pi_i(v) \in \mathscr{S}_{\text{ev}}(\xi'_i)$, by induction we have $v \in \mathscr{S}_{\text{ev}}(\xi'_0 \| \xi'_1)$, hence $e \cdot v \in \mathscr{S}_{\text{ev}}(\xi_0 \| \xi_1)$.

Let now $u \in \mathscr{S}_{ev}(\xi_0 \parallel \xi_1)$. We show, again by induction on the length of $u$, that $\pi_i(u) \in \mathscr{S}_{ev}(\xi_i)$. If $u = \varepsilon$ the statement is obvious. Let then $u = e \cdot v$, and $\xi_0 \parallel \xi_1 \xrightarrow{e} \xi'$ be the first step of the sequence. Again, we examine the three possibilities for $e$:

(i) If $e = \|_0 e_0$ then $\exists \xi_0'$ such that $\xi_0 \xrightarrow{e_0} \xi_0'$ and $\xi' = \xi_0' \parallel \xi_1$. Then, since $v \in \mathscr{S}_{ev}(\xi_0' \parallel \xi_1)$, by induction $\pi_0(v) \in \mathscr{S}_{ev}(\xi_0')$ and $\pi_1(v) \in \mathscr{S}_{ev}(\xi_1)$. It follows that $\pi_0(e \cdot v) = e_0 \cdot \pi_0(v) \in \mathscr{S}_{ev}(\xi_0)$ and $\pi_1(e \cdot v) = \pi_1(v) \in \mathscr{S}_{ev}(\xi_1)$.

(ii) The case of $e = \|_1 e_1$ is symmetric to the previous one.

(iii) If $e = (e_0, e_1)$ then $\exists \xi_0', \xi_1'$ such that $\xi_i \xrightarrow{e_i} \xi_i'$ for $i = 0, 1$ and $\xi' = \xi_0' \parallel \xi_1'$. By induction $\pi_i(v) \in \mathscr{S}_{ev}(\xi_i')$; therefore we have $\pi_i(e \cdot v) = \pi_i((e_0, e_1) \cdot v) = e_i \cdot \pi_i(v) \in \mathscr{S}_{ev}(\xi_i)$. ∎

6.3. *Correspondence between Event System and Flow Event Structures*

In this section we establish the *concrete equivalence* between the event system and the flow models. We start by considering the flow event structure model. We will show that the *computations* of $p$ in the event system are exactly the *configurations* of the flow event structure $\mathscr{S}(p)$. To this end we shall use the following characterisation for configurations from [10], where we recall that $\sharp$ is the reflexive closure of the conflict relation $\#$:

DEFINITION (Proving Sequences). Given a flow event structure $S = (E, \#, \prec)$, a proving sequence in $S$ is a sequence $e_1 \cdots e_n \cdots$ of distinct non-conflicting events (i.e., $i \neq j \Rightarrow e_i \neq e_j$ and $\neg(e_i \# e_j)$ for all $i, j$) satisfying

$$\forall i \, \forall e : e \prec e_i \Rightarrow \exists j < i . \, e \sharp e_j \prec e_i .$$

Note that any prefix of a proving sequence is itself a proving sequence. From [10] we have the following characterisation of configurations of flow event structures in terms of proving sequences.

PROPOSITION (from [10]). *Given a flow event structure $S = (E, \#, \prec)$, a subset $X$ of $E$ is a configuration of $S$ if and only if it can be enumerated as $\{e_1, ..., e_n, ...\}$, where $e_1 \cdots e_n \cdots$ is a proving sequence.*

We want to show that the *event sequences* of a CCS term $p$ in the event transition system coincide with the *proving sequences* of the flow event structure $\mathscr{S}(p)$. To do this we first give a characterisation of the proving sequences of the product of flow event structures, similar to Winskel's characterisation for the configurations of the product of stable event structures [58]. This characterisation only holds for flow event structures satisfying a structural property called axiom $\Delta$, which is known to be satisfied by all structures obtained from CCS terms [15]. Informally, this

property rules out certain *triangles* from flow event structures, or more precisely, it requires that these triangles be saturated by other events in a precise way. Let us recall from [15] the definition of property $\Delta$. Formally, a triangle is a structure with three distinct events $e_0, e_1, e_2$ such that $e_0 \# e_1 \prec e_2$ and $e_0$ is not related to $e_2$ by any of the relations $\prec$, $\succ$, $\#$. For two distinct events $e, e'$ we will write $eRe'$ for ($e \# e'$ or $e \prec e'$ or $e \succ e'$). Then axiom $\Delta$ is the following:

*Axiom* $\Delta$.

$$e_0 \# e_1 \prec e_2 \,\&\, \neg(e_0 R e_2) \Rightarrow \exists e_3 . (e_1 \# e_3 \prec e_2) \,\&\, (\forall e \# e_3 : e_1 \sharp e R e_2).$$

It has been shown in [15] that this property is satisfied by all flow event structures obtained from CCS terms. For such structures, the property may be justified intuitively as follows. A triangle $e_0 \# e_1 \prec e_2$ is necessarily created by a parallel composition: Intuitively, $e_1$ is a communication, and $e_0$ is a component of this communication which is not related to $e_2$. Hence $e_1$ must inherit its causality relation to $e_2$ from its other component, call it $e_3$. This event $e_3$ is precisely the one which is required by axiom $\Delta$. Moreover, if some other event $e$ is in conflict with $e_3$, there are two possibilities:

— Either $e$ is another communication using $e_3$ as a component, in which case it will be in conflict with $e_1$ and inherit from $e_3$ the flow relation to $e_2$.

— Or the conflict between $e$ and $e_3$ was created by the operator $+$, and thus either $e \prec e_2$ or $e \# e_2$, i.e., in any case $eRe_2$. Moreover $e_1$ inherits from $e_3$ the conflict to $e$.

PROPOSITION (from [15]). *For any* CCS *term* $p$ *the flow event structure* $\mathcal{S}(p)$ *satisfies the axiom* $\Delta$.

Let us then proceed with our characterisation of proving sequences for the product of flow event structures, or more precisely for $\mathcal{S}(p_0 \parallel p_1)$. Here we use the *projections* $\pi_0(u)$ and $\pi_1(u)$ of a sequence $u$ of events, as defined in the previous section (see Lemma 6.12). *Par abus de langage* we shall also regard these projections as partially defined on events of the product; that is, $\pi_0(\|_0 e) = e = \pi_0(e, e')$, and similarly for $\pi_1$.

LEMMA 6.13. *Let* $p_0$ *and* $p_1$ *be two* CCS *terms. Then* $u$ *is a proving sequence of* $\mathcal{S}(p_0 \parallel p_1)$ *if and only if its projections* $\pi_0(u)$, $\pi_1(u)$ *are proving sequences of* $\mathcal{S}(p_0)$ *and* $\mathcal{S}(p_1)$ *respectively.*

*Proof.* Let $u = e_1 \cdots e_n$ be a sequence of events of $\mathcal{S}(p_0 \parallel p_1)$ such that $\pi_0(u)$, $\pi_1(u)$ are proving sequences of $\mathcal{S}(p_0)$ and $\mathcal{S}(p_1)$. We denote by $X$ the set of events occurring in $u$. These are of the form $\|_i(e)$ or $(e, e')$.

We first show that $u$ is a sequence of distinct, non-conflicting events. The events of $u$ are all distinct since so are the events of $\pi_0(u)$, $\pi_1(u)$. Now let $e, e' \in X$, $e \neq e'$. Since $\neg(\pi_k(e) \sharp \pi_k(e'))$ for $k = 0, 1$, we have also $\neg(e \# e')$, by definition of the conflict relation on $\mathscr{E}$. Moreover, for any $e \in X$, since $\pi_k(e)$ is not self-conflicting for $k = 0, 1$, then neither is $e$. Therefore $X$ is conflict-free.

We show now that $u$ is consistently left-closed. Suppose $e \prec e_i$; then $\pi_k(e) \prec \pi_k(e_i)$ for $k = 0$ or $k = 1$. Since $\pi_k(u)$ is a proving sequence there exists $j < i$ such that $\pi_k(e) \sharp \pi_k(e_j) \prec \pi_k(e_i)$. Therefore we have $e \sharp e_j \prec e_i$.

Conversely, let $u = e_1 \cdots e_n$ be a proving sequence of $\mathscr{S}(p_0 \parallel p_1)$. Let us show that $\pi_0(u)$ is a proving sequence of $\mathscr{S}(p_0)$. The events of $\pi_0(u)$ are all distinct since otherwise $u$ would not be conflict-free (or would contain the same event twice). Similarly, $\pi_0(u)$ is conflict-free since otherwise $u$ would not be either.

Let now $\pi_0(u) = x_1 \cdots x_m$, and $x \prec x_i$, for some $x$ not occurring in $\pi_0(u)$. We have to find in the sequence $x_1 \cdots x_{i-1}$ a cause for $x_i$ which is in conflict with $x$. We have $x_i = \pi_0(e_j)$, for some $e_j \in X$. Then $\|_0 x \prec e_j$, and $\|_0 x \notin X$, since $x$ does not occur in $\pi_0(u)$. Therefore, since $u$ is a proving sequence, $\exists h < j$ s.t. $\|_0 x \# e_h \prec e_j$. The event $e_h$ cannot be of the form $\|_1 x'$. So let $x' = \pi_0(e_h)$; then, since $x \neq x'$ because $x$ does not occur in $\pi_0(u)$, we have $x \# x'$. Now $e_h \prec e_j$ is deduced either from $x' = \pi_0(e_h) \prec \pi_0(e_j) = x_i$ or from $\pi_1(e_h) \prec \pi_1(e_j)$. In the first case $x'$ is the required cause for $x_i$ and we have finished.

Otherwise we have $x' \not\prec x_i$, and from $e_h, e_j \in X$ and $e_h \prec e_j$ it follows that $\neg(x' \# x_i)$ and $x_i \not\prec x'$. Then, since $x' \# x \prec x_i$, we may apply axiom $\Delta$ to deduce that $\exists x''. x \# x'' \prec x_i$. From this we deduce also $\|_0 x \# \|_0 x'' \prec e_j$. There are again two cases:

(a) If $\|_0 x''$ occurs in $e_1 \cdots e_{j-1}$, then $x''$ occurs in $x_1 \cdots x_{i-1}$ and $x''$ is the required cause for $x_i$.

(b) Otherwise $\|_0 x''$ does not occur in $e_1 \cdots e_{j-1}$. But then, since $e_1 \cdots e_j$ is a proving sequence (because it is a prefix of the proving sequence $u$), there exists $l < j$ s.t. $\|_0 x'' \# e_l \prec e_j$. The event $e_l$ cannot be of the form $\|_1 \bar{x}$. So let $\bar{x} = \pi_0(e_l)$. Then $\|_0 x'' \# e_l$ is deduced from $x'' \sharp \bar{x}$. If $x'' = \bar{x}$ then $x''$ occurs in $x_1 \cdots x_i$, and we have finished. Otherwise $x'' \# \bar{x}$. Then, by axiom $\Delta$ again, we have that $x \# \bar{x}$ and ($\bar{x} \# x_i$ or $\bar{x} \prec x_i$ or $x_i \prec \bar{x}$). Now it cannot be $\bar{x} \# x_i$ since $e_l$, $e_j$ both occur in $u$, and it cannot be $x_i \prec \bar{x}$ because $e_l \prec e_j$. Thus $\bar{x} \prec x_i$, and this ends the proof. ∎

Now Lemma 6.12 in Section 6.2 gives us a similar characterisation for the event sequences of $(p_0 \parallel p_1)$ in the event system. Using these two lemmas, we may show that:

PROPOSITION 6.14. *For any term $p$, a sequence $u = e_1 \cdots e_n$ is an event sequence of $p$ in the event system if and only if it is a proving sequence of the flow event structure $\mathcal{S}(p)$.*

*Proof.* We show that any event sequence $u = e_1 \cdots e_n$ of $p$ is a proving sequence of $\mathcal{S}(p)$, by induction on $n$. This is trivial for $n = 0$. For $n > 0$, we know that $p \xrightarrow{e_1} \xi$ for some $\xi$. Then we proceed by induction on the proof of this transition:

(i) If this transition is an instance of A0′ then $p = a.q$ and there exists an event sequence $v = e'_1 \cdots e'_{n-1}$ of $q$, such that $u = a \cdot \hat{a}v$, that is $u = a \cdot \hat{a}e'_1 \cdots \hat{a}e'_{n-1}$. By induction $v$ is a proving sequence of $\mathcal{S}(q)$. Then it is easy to see that $u$ is a proving sequence of $\mathcal{S}(p)$, since $a \neq \hat{a}e'_i$, $u$ is conflict-free, because $(e_i \#_p e_j \Leftrightarrow i, j > 1 \,\&\, e'_{i-1} \#_q e'_{j-1})$, and $u$ is consistently left-closed, because $(e \prec_p e_j) \Leftrightarrow (e = e_1 = a \,\&\, e_j = \hat{a}e'_1)$ or $(e = \hat{a}e' \,\&\, e' \prec_q e'_{j-1})$.

(ii) If the transition is inferred using S0′ or S1′ then $p = q + r$ and $u$ is obtained either from an event sequence $u_0 = e'_1 \cdots e'_n$ of $q$ such that $u = +_0 u_0$, or form an event sequence $u_1 = e''_1 \cdots e''_n$ of $r$ such that $u = +_1 u_1$. Suppose $u = +_0 u_0 = +_0 e'_1 \cdots +_0 e'_n$. By induction $u_0$ is a proving sequence of $\mathcal{S}(q)$. Then $u$ is a proving sequence of $\mathcal{S}(p)$, since for any $e_i = +_0 e'_i$, $e_j = +_0 e'_j$ occuring in $u$ we have $(e_i \#_p e_j \Leftrightarrow e'_i \#_q e'_j)$ and, for $e \in \mathcal{S}(p)$, $(e \prec_p e_j \Leftrightarrow e = +_0 e'_i \,\&\, e'_i \prec_q e'_j)$.

(iii) If the transition is proved using P0′, P1′ or C′ then $p = (p_0 \parallel p_1)$. Since $u$ is an event sequence of $p$, by Lemma 6.12 $\pi_0(u)$, $\pi_1(u)$ are event sequences of $p_0, p_1$ respectively. Then by induction $\pi_0(u)$ and $\pi_1(u)$ are proving sequences of $\mathcal{S}(p_0)$, $\mathcal{S}(p_1)$. It follows by Lemma 6.13 that $u$ is a proving sequence of $\mathcal{S}(p_0 \parallel p_1)$.

(iv) If the transition results from R′ then $p = q\backslash\alpha$ and there exists an event sequence $v = e'_1 \cdots e'_n$ of $q$ such that $\forall i. \ell(e'_i) \neq \alpha, \bar{\alpha}$ and $u = \backslash_\alpha v = \backslash_\alpha e'_1 \cdots \backslash_\alpha e'_n$. By induction $v$ is a proving sequence of $\mathcal{S}(q)$. Then $u$ is conflict-free because $(\backslash_\alpha e' \#_p \backslash_\alpha e'') \Leftrightarrow (e' \#_q e'')$ or $(e' = e'' \,\&\, \ell(e') = \alpha, \bar{\alpha})$, and $u$ is consistently left-closed because whenever $\backslash_\alpha e' \prec_p \backslash_\alpha e'_i$ then $e' \prec_q e'_i$ and thus, by consistent left-closedness of $v$, $(\exists j < i. e' \sharp_q e'_j \prec_q e'_i)$; since $\ell(e'_j) \neq \alpha, \bar{\alpha}$, this implies that $(\backslash_\alpha e' \sharp_q \backslash_\alpha e_j \prec_p \backslash_\alpha e_i)$.

(v) Finally, if the transition is inferred by means of FIX′, then $p = q[\mu x.q/x]$, and the result follows by induction since $u$ is an event sequence of $q[\mu x.q/x]$, with a shorter proof of the initial step, and $\mathcal{S}(p) = \mathcal{S}(q[\mu x.q/x])$.

Conversely, we show that a proving sequence $u = e_1 \cdots e_n$ of $\mathcal{S}(p)$ is an event sequence of $p$, by induction on $n$. This is trivial for $n = 0$. For $n > 0$, we proceed by induction on the definition of $e_1 \in \mathcal{E}(p)$:

(i) If $e_1 = a$ and $p = a.q$ then $u$ is of the form $u = a \cdot \hat{a}v = a \cdot \hat{a}e'_1 \cdots \hat{a}e'_{n-1}$, for some proving sequence $v = e'_1 \cdots e'_{n-1}$ of $\mathscr{S}(q)$. By induction, $v$ is an event sequence of $q$. Now, since $p \xrightarrow{a} \hat{a}.q$, it is clear that $a \cdot \hat{a}v$ is an event sequence of $p$.

(ii) We cannot have $e_1 = \hat{a}.e'$ with $p = a.q$ and $e' \in \mathscr{E}(q)$, since $a \prec \hat{a}.e'$ and $a \in \mathscr{E}(p)$, contradicting the fact that $u$ is a proving sequence of this structure.

(iii) If $e_1 = \|_1 x_i$ or $e = (x_0, x_1)$ with $p = (p_0 \| p_1)$ and $x_i \in \mathscr{E}(p_i)$ then by Lemma 6.13 $\pi_0(u)$ and $\pi_1(u)$ are proving sequences of $\mathscr{S}(p_0)$, $\mathscr{S}(p_1)$. By induction $\pi_0(u)$ and $\pi_1(u)$ are event sequences of $p_0, p_1$. Then by Lemma 6.12 it follows that $u$ is an event sequence of $p$.

(iv) If $e_1 = +_i x_i$ with $p = p_0 + p_1$ and $x_i \in \mathscr{E}(p_i)$ then $u$ is obtained either from a proving sequence $u_0$ of $\mathscr{S}(p_0)$ or from a proving sequence $u_1$ of $\mathscr{S}(p_1)$. Suppose, e.g., that $u = +_0 u_0 = +_0 e'_1 \cdots +_0 e'_n$, where $u_0$ is a proving sequence of $\mathscr{S}(p_0)$. By induction $u_0$ is an event sequence of $p_0$; i.e., there exists $\xi_1, \ldots \xi_n$ such that $p_0 \xrightarrow{e'_1} \xi_1 \cdots \xrightarrow{e'_n} \xi_n$. From this we deduce that $p \xrightarrow{+_0 e'_1} \xi_1 \hat{+}_0 p_1 \cdots \xrightarrow{+_0 e'_n} \xi_n \hat{+}_0 p_1$; that is, $u$ is an event sequence of $p$.

(v) If $e_1 = \backslash_\alpha e'$ with $p = q \backslash \alpha$ and $e' \in \mathscr{E}(q)$ then there exists a proving sequence $v$ of $\mathscr{S}(q)$, $v = e'_1 \cdots e'_n$, such that $\forall i. \ell(e'_i) \neq \alpha, \bar{\alpha}$ and $u = \backslash_\alpha v = \backslash_\alpha e'_1 \cdots \backslash_\alpha e'_n$. By induction, $v$ is an event sequence of $q$; i.e., there exist $\xi_1, \ldots \xi_n$ such that $q \xrightarrow{e'_1} \xi_1 \cdots \xrightarrow{e'_n} \xi_n$. Then, since $\forall i. \ell(e'_i) \neq \alpha, \bar{\alpha}$, we have also $q \backslash \alpha \xrightarrow{\backslash_\alpha e'_1} \xi_1 \backslash \alpha \cdots \xrightarrow{\backslash_\alpha e'_n} \xi_n \backslash \alpha$.

(vi) Finally if $p = \mu x.q$ and $e \in \mathscr{E}(q[\mu x.q/x])$ then $u$ is a proving sequence of $\mathscr{S}(q[\mu x.q/x])$ since $\mathscr{S}(q[\mu x.q/x]) = \mathscr{S}(\mu x.q)$, therefore by induction $u$ is an event sequence of $q[\mu x.q/x]$, hence also (using FIX') of $p$. ∎

We want now to show that the domain $\mathscr{D}_{es}(p) = (\mathscr{F}(\mathscr{S}(p)), \subseteq)$ of event computations of the CCS term $p$ is isomorphic to the domain $\mathscr{D}_{ev}(p) = (\mathscr{T}_{ev}(p)/\simeq, \sqsubseteq)$ of event computations of $p$. The previous proposition shows that the mapping Ev provides a surjection from $\mathscr{T}_{ev}(p)$ to $\mathscr{F}(\mathscr{S}(p))$. Then to prove the announced isomorphism, it just remains to show:

LEMMA 6.15. *Let $\xi$ be a marked term such that there exists an event transition sequence from a* CCS *term $p$ to $\xi$. Then for any event transition sequences $\sigma_0$ and $\sigma_1$ of $\xi$, we have*

$$[\![\sigma_0]\!] \sqsubseteq [\![\sigma_1]\!] \Leftrightarrow \mathrm{Ev}(\sigma_0) \subseteq \mathrm{Ev}(\sigma_1).$$

*Proof.* In one direction this is Remark 6.2. Conversely, let $\sigma_0 = \xi \xrightarrow{e_1} \xi_1 \cdots \xi_{n-1} \xrightarrow{e_n} \xi_n$ and $\sigma_1 = \xi \xrightarrow{e'_1} \xi'_1 \cdots \xi'_{m-1} \xrightarrow{e'_m} \xi'_m$ be such that

$\{e_1, ..., e_n\} \subseteq \{e'_1, ..., e'_m\}$. We prove that there exists $\sigma$ such that $\sigma_1 \simeq \sigma_0 \sigma$ by induction on $n$:

(i) This is trivial for $n = 0$.

(ii) If $n > 0$ then there is a unique $i$ such that $e_1 = e'_i$. We proceed by induction on $i$:

(a) If $i = 1$ then $\xi_1 = \xi'_1$, $\{e_2, ..., e_n\} \subseteq \{e'_2, ..., e'_m\}$, and we use the induction hypothesis (on the length of the sequence starting from $\xi_1$) to conclude.

(b) Otherwise, let $c_1 \cdots c_k$ be an event sequence of $p$ such that $p \xrightarrow{c_1} \cdots \xrightarrow{c_k} \xi$. Then by the previous proposition we know that $c_1 \cdots c_k \cdot e_1 \cdots e_n$ and $c_1 \cdots c_k \cdot e'_1 \cdots e'_m$ are both proving sequences of the flow event structure $\mathscr{S}(p)$. Let us show that this implies $e'_{i-1} \not\prec e'_i$, where $e'_i = e_1$. Assume the contrary: then, since $c_1 \cdots c_k \cdot e_1$ is a proving sequence of $\mathscr{S}(p)$, there should exist $j$ such that $e'_{i-1} \# c_j \prec e_1 = e'_i$. But this is impossible: if $e'_{i-1} = c_j$ then $c_j$ would appear twice in the proving sequence $c_1 \cdots c_k \cdot e'_1 \cdots e'_m$, and if $e'_{i-1} \# c_j$ then this sequence would not be conflict-free. Therefore we have $e'_{i-1} \smile e'_i$ by Lemma 6.11. Then, by the Proposition 6.10, we know that we may permute the events $e'_{i-1}$ and $e'_i$. The resulting sequence $\sigma'_1$ is such that $\sigma'_1 \simeq \sigma_1$, and by induction hypothesis (on the index where $e_1$ occurs in $\sigma'_1$) there exists $\sigma$ such that $\sigma'_1 \simeq \sigma_0 \sigma$. ∎

Finally, we have shown:

THEOREM 6.16. *For any* CCS *term* $p$ *the domains of computations* $\mathscr{D}_{ev}(p) = (\mathscr{T}_{ev}(p)/\simeq, \sqsubseteq)$ *and* $\mathscr{D}_{es}(p) = (\mathscr{F}(\mathscr{S}(p)), \subseteq)$ *are isomorphic. Moreover* $X \in \mathscr{F}(\mathscr{S}(p))$ *if and only if there exists* $\sigma \in \mathscr{T}_{ev}(p)$ *such that* $X = \mathrm{Ev}(\sigma)$.

An alternative way of proving this result could have been to define a transition system on flow event structures, and relate it to the event transition system in the same way as we will do for flow nets. In the appendix we shall introduce such a transition system on flow event structures, and give a sketch of the alternative proof.

### 6.4. *Correspondence between Event System and Flow Nets*

Let us turn now to the relation between the event system and the flow net interpretations for a fixpoint-free term $p$. Again, the correspondence is a rather direct one: we will show that for each FCCS term $p$, the transition system on markings of the net $\mathscr{N}(p)$ is *isomorphic* to the event transition system generated by $p$. More precisely, we shall show that a sequence of events is firable in the net $\mathscr{N}(p)$ if and only if it is an event sequence of $p$ (see the definition at the end of Section 6.2). From the previous theorem, this will also imply that a sequence of events is firable in the net $\mathscr{N}(p)$ if

and only if it is a configuration, or more precisely a proving sequence, of the flow event structure $\mathscr{S}(p)$.

Recall the definition of ancestor $\varrho(\xi)$ from section 6.1.† According to our intuition, if $\xi$ has ancestor $\varrho(\xi)=p$, then $\xi$ is a partially executed version of $p$. Then we expect $\xi$ to correspond to some marking in the net $\mathscr{N}(p)$. In fact, since a marked term $\xi$ records both the original term—its ancestor $\varrho(\xi)$—and a state of execution for it, it is rather straightforward to associate with $\xi$ a marking in the net $\mathscr{N}(\varrho(\xi))$.

The marking $\mu(\xi)$ associated with marked terms is defined inductively starting from the marking $\mu(p)=\mu_p$ of FCCS terms, where $\mu_p$ is the initial marking of the net $\mathscr{N}(p)$ as given in the previous section. It will be convenient to distinguish two parts in the marking $\mu(\xi)$, which we denote $\Gamma(\xi)$ and $\Delta(\xi)$. The set $\Gamma(\xi)=_{\text{def}}\mu(\varrho(\xi))\cap\mu(\xi)$ represents the part of the marking $\mu(\varrho(\xi))$ which has not been affected by the execution reaching $\xi$, if any, while $\Delta(\xi)=_{\text{def}}\mu(\xi)-\mu(\varrho(\xi))$ is the part of the marking determined by this execution. This idea is somewhat similar to that of "concurrent residual" and "local residual" of [13]. Note that by definition we have $\Gamma(\xi)\cap\Delta(\xi)=\varnothing$ and $\mu(\xi)=\Gamma(\xi)\cup\Delta(\xi)$. Now the marking $\mu(\xi)$ determined by a marked term $\xi$ in the net $\mathscr{N}(\varrho(\xi))$ is defined as follows:

$$
\begin{aligned}
\mu(p)&=\mu_p \qquad \text{(the initial marking of } \mathscr{N}(p))\\
\mu(\hat{a}.\xi)&=\hat{a}.\mu(\xi)\\
\mu(\xi \parallel \xi')&=\|_0\,\mu(\xi)\cup\|_1\,\mu(\xi')\\
\mu(\xi\,\hat{+}_0\,q)&=(\Gamma(\xi)+\mu(q))\cup+_0\,\Delta(\xi)\\
\mu(p\,\hat{+}_1\,\xi)&=(\mu(p)+\Gamma(\xi))\cup+_1\,\Delta(\xi)\\
\mu(\xi\backslash\alpha)&=\backslash_\alpha\,\mu(\xi).
\end{aligned}
$$

We proceed to show that the transitions of a marked term $\xi$ with ancestor $p$ correspond exactly to the transitions from the marking $\mu(\xi)$ in the net $\mathscr{N}(p)$. To prove this result, that is, the forthcoming Propositions 6.17 and 6.20, we will need to know the precise expressions for $\mathbf{pre}_p(e)$ and $\mathbf{post}_p(e)$, depending on the structure of $p$. These are derived from the definition of $\mathscr{N}(p)$ given in the Section 5, as follows:

$$
\begin{aligned}
\mathbf{pre}_{a.p}(a)&=\{a.p\}\\
\mathbf{post}_{a.p}(a)&=\hat{a}.(\mu(p))\\
\mathbf{pre}_{a.p}(\hat{a}.e)&=\hat{a}.(\mathbf{pre}_p(e))\\
\mathbf{post}_{a.p}(\hat{a}.e)&=\hat{a}.(\mathbf{post}_p(e))
\end{aligned}
$$

† For the rest of this section we assume we are dealing with marked terms $\xi$ whose ancestor $\varrho(\xi)$ is an FCCS term.

$$\mathbf{pre}_{p_0 \| p_1}(\|_i e) = \|_i (\mathbf{pre}_{p_i}(e))$$

$$\mathbf{post}_{p_0 \| p_1}(\|_i e) = \|_i (\mathbf{post}_{p_i}(e))$$

$$\mathbf{pre}_{p \| q}((e, e')) = \|_0 (\mathbf{pre}_p(e)) \cup \|_1 (\mathbf{pre}_q(e'))$$

$$\mathbf{post}_{p \| q}((e, e')) = \|_0 (\mathbf{post}_p(e)) \cup \|_1 (\mathbf{post}_q(e'))$$

$$\mathbf{pre}_{p_0 + p_1}(+_i e) = \begin{cases} +_0 (\mathbf{pre}_{p_0}(e) - \mu(p_0)) \cup ((\mathbf{pre}_{p_0}(e) \cap \mu(p_0)) + \mu(p_1)) \\ \quad \text{for} \quad i = 0 \\ +_1 (\mathbf{pre}_{p_1}(e) - \mu(p_1)) \cup (\mu(p_0) + (\mathbf{pre}_{p_1}(e) \cap \mu(p_1))) \\ \quad \text{for} \quad i = 1 \end{cases}$$

$$\mathbf{post}_{p_0 + p_1}(+_i e) = +_i (\mathbf{post}_{p_i}(e))$$

$$\mathbf{pre}_{p \setminus \alpha}(\setminus_\alpha e) = \setminus_\alpha (\mathbf{pre}_p(e))$$

$$\mathbf{post}_{p \setminus \alpha}(\setminus_\alpha e) = \setminus_\alpha (\mathbf{post}_p(e)).$$

PROPOSITION 6.17. *For any marked term* $\xi$, *if* $\xi \xrightarrow{e}\!\!\!\!\rightarrow \xi'$ *then* $\mu(\xi) \xmapsto{e} \mu(\xi')$ *in the net* $\mathcal{N}(\varrho(\xi))$.

*Proof.* We show this statement by induction on the proof of $\xi \xrightarrow{e}\!\!\!\!\rightarrow \xi'$. We have to show that

$$\xi \xrightarrow{e}\!\!\!\!\rightarrow \xi' \text{ with } p = \varrho(\xi) \Rightarrow \mathbf{pre}_p(e) \subseteq \mu(\xi)$$

and

$$\mu(\xi') = \mu(\xi) - \mathbf{pre}_p(e) \cup \mathbf{post}_p(e).$$

The last expression should be intended as $(\mu(\xi) - \mathbf{pre}_p(e)) \cup \mathbf{post}_p(e)$. However, for simplicity, we shall omit the parentheses around $\mu(\xi) - \mathbf{pre}_p(e)$ in this proof. We consider the various cases for $\xi \xrightarrow{e}\!\!\!\!\rightarrow \xi'$:

(i) If the transition is $a.p \xrightarrow{a}\!\!\!\!\rightarrow \hat{a}.p$, then $\mathbf{pre}_{a.p} = \{a.p\} = \mu(a.p)$ and

$$\mu(a.p) - \mathbf{pre}_{a.p}(a) \cup \mathbf{post}_{a.p}(a) = \{a.p\} - \{a.p\} \cup \hat{a}.(\mu(p)) = \mu(\hat{a}.p).$$

(ii) If the transition is $\hat{a}.\xi \xrightarrow{\hat{a}.e}\!\!\!\!\rightarrow \hat{a}.\xi'$, with $\xi \xrightarrow{e}\!\!\!\!\rightarrow \xi'$ and $\varrho(\xi) = p$, then $\varrho(\hat{a}.\xi) = a.p$, and by induction we have:

$$\mathbf{pre}_{a.p}(\hat{a}.e) = \hat{a}.(\mathbf{pre}_p(e)) \subseteq \hat{a}.(\mu(\xi)) = \mu(\hat{a}.\xi)$$

and

$$\begin{aligned} &\mu(\hat{a}.\xi) - \mathbf{pre}_{a.p}(\hat{a}.e) \cup \mathbf{post}_{a.p}(\hat{a}.e) \\ &\quad = \hat{a}.(\mu(\xi)) - \hat{a}.(\mathbf{pre}_p(e)) \cup \hat{a}.(\mathbf{post}_p(e)) \\ &\quad = \hat{a}.(\mu(\xi) - \mathbf{pre}_p(e) \cup \mathbf{post}_p(e)) \\ &\quad = \hat{a}.(\mu(\xi')) = \mu(\hat{a}.\xi'). \end{aligned}$$

(iii) If the transition is $(\xi_0 \parallel \xi_1) \xrightarrow{\|_0 e} \xi_0' \parallel \xi_1$, with $\xi_0 \xrightarrow{e} \xi_0'$, $\varrho(\xi_0) = p$, and $\varrho(\xi_1) = q$, then $\varrho(\xi_0 \parallel \xi_1) = (p \parallel q)$ and by induction

$$\mathbf{pre}_{p \parallel q}(\|_0 e) = \|_0 (\mathbf{pre}_p(e)) \subseteq \|_0 (\mu(\xi_0)) \subseteq \mu(\xi_0 \parallel \xi_1)$$

and

$$\begin{aligned}
&\mu(\xi_0 \parallel \xi_1) - \mathbf{pre}_{p \parallel q}(\|_0 e) \cup \mathbf{post}_{p \parallel q}(\|_0 e) \\
&\quad = \|_0 (\mu(\xi_0)) \cup \|_1 (\mu(\xi_1)) - \|_0 (\mathbf{pre}_p(e)) \cup \|_0 (\mathbf{post}_p(e)) \\
&\quad = \|_0 (\mu(\xi_0) - \mathbf{pre}_p(e) \cup \mathbf{post}_p(e)) \cup \|_1 (\mu(\xi_1)) \\
&\quad = \|_0 (\mu(\xi_0')) \cup \|_1 (\mu(\xi_1)) \\
&\quad = \mu(\xi_0' \parallel \xi_1).
\end{aligned}$$

(iv) If the transition is $(\xi_0 \parallel \xi_1) \xrightarrow{(e, e')} (\xi_0' \parallel \xi_1')$ because $\xi_0 \xrightarrow{e} \xi_0'$ and $\xi_1 \xrightarrow{e'} \xi_1'$, then if $\varrho(\xi_0) = p$ and $\varrho(\xi_1) = q$, we have $\varrho(\xi_0 \parallel \xi_1) = (p \parallel q)$, and by induction

$$\begin{aligned}
\mathbf{pre}_{p \parallel q}((e, e')) &= \|_0 (\mathbf{pre}_p(e)) \cup \|_1 (\mathbf{pre}_q(e')) \subseteq \|_0 (\mu(\xi_0)) \cup \|_1 (\mu(\xi_1)) \\
&= \mu(\xi_0 \parallel \xi_1)
\end{aligned}$$

and

$$\begin{aligned}
&\mu(\xi_0 \parallel \xi_1) - \mathbf{pre}_{p \parallel q}((e, e')) \cup \mathbf{post}_{p \parallel q}((e, e')) \\
&\quad = \|_0 (\mu(\xi_0)) \cup \|_1 (\mu(\xi_1)) - \|_0 (\mathbf{pre}_p(e)) - \|_1 (\mathbf{pre}_q(e')) \\
&\qquad \cup \|_0 (\mathbf{post}_p(e)) \cup \|_1 (\mathbf{post}_q(e')) \\
&\quad = \|_0 (\mu(\xi_0) - \mathbf{pre}_p(e) \cup \mathbf{post}_p(e)) \\
&\qquad \cup \|_1 (\mu(\xi_1) - \mathbf{pre}_q(e') \cup \mathbf{post}_q(e')) \\
&\quad = \|_0 (\mu(\xi_0')) \cup \|_1 (\mu(\xi_1')) = \mu(\xi_0' \parallel \xi_1').
\end{aligned}$$

(v) If the transition is $\xi \backslash \alpha \xrightarrow{\backslash_\alpha e} \xi' \backslash \alpha$ with $\xi \xrightarrow{e} \xi'$, $\mathbf{nm}(\ell(e)) \neq \alpha$, and $\varrho(\xi) = p$, then $\varrho(\xi \backslash \alpha) = p \backslash \alpha$ and by induction

$$\mathbf{pre}_{p \backslash \alpha}(\backslash_\alpha e) = \backslash_\alpha (\mathbf{pre}_p(e)) \subseteq \backslash_\alpha \mu(\xi) = \mu(\xi \backslash \alpha)$$

and

$$\begin{aligned}
\mu(\xi \backslash \alpha) - \mathbf{pre}_{p \backslash \alpha}(\backslash_\alpha e) \cup \mathbf{post}_{p \backslash \alpha}(\backslash_\alpha e) &= \backslash_\alpha (\mu(\xi)) - \backslash_\alpha (\mathbf{pre}_p(e)) \\
&\quad \cup \backslash_\alpha (\mathbf{post}_p(e)) \\
&= \backslash_\alpha (\mu(\xi')) = \mu(\xi' \backslash \alpha).
\end{aligned}$$

(vi) If the transition is $p+q \xrightarrow{+_0 e} (\xi \hat{+}_0 q)$ because $p \xrightarrow{e} \xi$, then by induction $\mathbf{pre}_p(e) \subseteq \mu(p)$, therefore $\mathbf{pre}_p(e) - \mu(p) = \varnothing$ and $\mathbf{pre}_p(e) \cap \mu(p)$, hence

$$\begin{aligned}\mathbf{pre}_{p+q}(+_0 e) &= \mathbf{pre}_p(e) + \mu(q) \\ &\subseteq \mu(p) + \mu(q) = \mu(p+q).\end{aligned}$$

By induction, we have $\mu(\xi) = \mu(p) - \mathbf{pre}_p(e) \cup \mathbf{post}_p(e)$, therefore

$$\Delta(\xi) = (\mu(p) - \mathbf{pre}_p(e) \cup \mathbf{post}_p(e)) - \mu(p)$$

and

$$\Gamma(\xi) = (\mu(p) - \mathbf{pre}_p(e) \cup \mathbf{post}_p(e)) \cap \mu(p).$$

Since $\mathbf{post}_p(e) \cap \mu(p) = \varnothing$ for any $p$ and $e$, we have $\Delta(\xi) = \mathbf{post}_p(e)$ and $\Gamma(\xi) = \mu(p) - \mathbf{pre}_p(e)$; therefore

$$\begin{aligned}\mu(\xi \hat{+}_0 q) &= (\Gamma(\xi) + \mu(q)) \cup +_0 \Delta(\xi) \\ &= (\mu(p) - \mathbf{pre}_p(e)) + \mu(q) \cup +_0 \mathbf{post}_p(e) \\ &= \mu(p+q) - \mathbf{pre}_{p+q}(+_0 e) \cup \mathbf{post}_{p+q}(+_0 e).\end{aligned}$$

(vii) If the transition is $(\xi \hat{+}_0 q) \xrightarrow{+_0 e} (\xi' \hat{+}_0 q)$ with $\xi \xrightarrow{e} \xi'$, let $p = \varrho(\xi) = \varrho(\xi')$. Then by induction $\mathbf{pre}_p(e) \subseteq \mu(\xi)$, therefore $\mathbf{pre}_p(e) - \mu(p) \subseteq \Delta(\xi)$ and $\mathbf{pre}_p(e) \cap \mu(p) \subseteq \Gamma(\xi)$; hence

$$\begin{aligned}\mathbf{pre}_{p+q}(+_0 e) &= ((\mathbf{pre}_p(e) \cap \mu(p)) + \mu(p)) \cup +_0 (\mathbf{pre}_p(e) - \mu(p)) \\ &\subseteq (\Gamma(\xi) + \mu(q)) \cup +_0 \Delta(\xi) = \mu(\xi \hat{+}_0 q).\end{aligned}$$

By induction $\mu(\xi') = \mu(\xi) - \mathbf{pre}_p(e) \cup \mathbf{post}_p(e)$. Since $\mathbf{post}_p(e) \cap \mu(p) = \varnothing$ for any $p$ and $e$, we have

$$\begin{aligned}\Gamma(\xi') &= (\mu(\xi) - \mathbf{pre}_p(e)) \cap \mu(p) \\ &= \Gamma(\xi) - (\mathbf{pre}_p(e) \cap \mu(p))\end{aligned}$$

and

$$\begin{aligned}\Delta(\xi') &= (\mu(\xi) - \mathbf{pre}_p(e)) - \mu(p) \cup \mathbf{post}_p(e) \\ &= \Delta(\xi) - (\mathbf{pre}_p(e) - \mu(p)) \cup \mathbf{post}_p(e).\end{aligned}$$

Then

$$\begin{aligned}\mu(\xi' \hat{+}_0 q) &= (\Gamma(\xi') + \mu(q)) \cup +_0 \Delta(\xi') \\ &= ((\Gamma(\xi) - (\mathbf{pre}_p(e) \cap \mu(p))) + \mu(q)) \\ &\quad \cup +_0 (\Delta(\xi) - (\mathbf{pre}_p(e) - \mu(p)) \cup \mathbf{post}_p(e)) \\ &= ((\Gamma(\xi) + \mu(q)) - (\mathbf{pre}_p(e) \cap \mu(p)) + \mu(q)) \\ &\quad \cup +_0 \Delta(\xi) - +_0 (\mathbf{pre}_p(e) - \mu(p)) \cup +_0 \mathbf{post}_p(e) \\ &= \mu(\xi \hat{+}_0 q) - \mathbf{pre}_{p+q}(+_0 e) \cup \mathbf{post}_{p+q}(+_0 e).\end{aligned}$$

The remaining cases are similar. ∎

We want now to show the reverse correspondence, namely that any reachable marking $\mu$ of $\mathcal{N}(p)$ is the marking of some marked term $\xi$ such that $\varrho(\xi) = p$ and that the transitions from $\mu$ induce corresponding transitions from $\xi$. We first prove that the part $\Gamma(\xi)$ of the marking associated with $\xi$ decreases along event transition sequences:

LEMMA 6.18. *If $\xi \xrightarrow{e}\!\!\!\!\rightarrow \xi'$ then $\Gamma(\xi') \subseteq \Gamma(\xi)$. Moreover $\Gamma(\xi') \subset \mu(\varrho(\xi))$.*

*Proof.* By induction on the inference of the transition, straightforward. Note that in the case $\xi = (\xi_0 \parallel \xi_1)$ we may have $\Gamma(\xi') = \Gamma(\xi)$. ∎

An obvious consequence of this lemma is the following:

COROLLARY 6.19. *If $\xi$ is a marked term reachable from its ancestor, that is if there exists $e_1, \ldots, e_n$ such that*

$$\varrho(\xi) \xrightarrow{e_1}\!\!\!\!\rightarrow \cdots \xrightarrow{e_n}\!\!\!\!\rightarrow \xi$$

*then $\Gamma(\xi) \subseteq \Gamma(\varrho(\xi))$. Moreover if $n > 0$ then $\Gamma(\xi) \subset \mu(\varrho(\xi))$.*

Now we may proceed to show the correspondence between net transitions and event transitions:

PROPOSITION 6.20. *For any marked term $\xi$ which is reachable from its ancestor, if $\mu(\xi) \xrightarrow{e} \mu'$ in the net $\mathcal{N}(\varrho(\xi))$, then there exists $\xi'$ such that $\xi \xrightarrow{e}\!\!\!\!\rightarrow \xi'$.*

*Proof.* By induction on the structure of the term $\xi$. We consider the various cases.

(i) If $\xi = a.p$ then the unique event fireable at $\mu(\xi)$ is $e = a$. Then $a.p \xrightarrow{a}\!\!\!\!\rightarrow \hat{a}.p$ is the required event transition.

(ii) If $\xi = q + r$ then the events fireable at $\mu(\xi) = (\mu(q) + \mu(r))$ are of the form $+_i e_0$. Take the case $e = +_0 e_0$. Now $+_0 e_0$ is fireable at $\mu(\xi)$ because $e_0$ is fireable at $\mu(q)$. By induction $\exists \xi_0$ s.t. $q \xrightarrow{e_0} \xi_0$. Then $(q + r) \xrightarrow{+_0 e_0} (\xi_0 \hat{+}_0 r)$ is the required event transition.

(iii) If $\xi = \hat{a}.\xi'$ then any event fireable at $\mu(\xi) = \hat{a}.\mu(\xi')$ is of the form $\hat{a}.e$. Here, $\hat{a}.e$ is fireable at $\mu(\xi)$ because $e$ is fireable at $\mu(\xi')$. By induction $\exists \xi''$ s.t. $\xi' \xrightarrow{e} \xi''$. Hence $\hat{a}.\xi' \xrightarrow{\hat{a}.e} \hat{a}.\xi''$ is the required transition.

(iv) If $\xi = (\xi_0 \parallel \xi_1)$ then the events fireable from the marking $\mu(\xi_0 \parallel \xi_1) = \|_0(\mu(\xi_0)) \cup \|_1(\mu(\xi_1))$ may be either of the form $\|_i(e_i)$ or of the form $(e_0, e_1)$:

(a) $e = \|_0 e_0$ because $e_0$ is fireable at $\mu(\xi_0)$. By induction $\exists \xi_0'$ s.t. $\xi_0 \xrightarrow{e'} \xi_0'$. Then the required transition is $(\xi_0 \parallel \xi_1) \xrightarrow{\|_0 e'} (\xi_0 \parallel \xi_1')$.

(b) $e = (e_0, e_1)$ because $e_i$ is fireable at $\mu(\xi_i)$, for $i = 0, 1$. By induction $\exists \xi_0', \xi_1'$ s.t. $\xi_i \xrightarrow{e'} \xi_i'$. Then $(\xi_0 \parallel \xi_1) \xrightarrow{(e_0, e_1)} (\xi_0' \parallel \xi_1')$ is the required transition.

(v) If $\xi = \xi_0 \hat{+}_0 q$ then $\mu(\xi) = (\Gamma(\xi_0) + \mu(q)) \cup +_0(\Delta(\xi_0))$. Clearly if $e$ is fireable from $\mu(\xi)$ then $e$ is of the form $+_i e_i$ for $i = 0$ or $i = 1$. We show first that $i = 1$ is impossible. If $\varrho(\xi_0) = p$ we have

$$\mathbf{pre}_{p+q}(+_1 e_1) = +_1(\mathbf{pre}_q(e_1) - \mu(q)) \cup (\mu(p) + (\mathbf{pre}_q(e_1) \cap \mu(q))).$$

If $\mathbf{pre}_q(e_1) - \mu(q) \neq \varnothing$, the event $+_1 e_1$ is not fireable at $\mu(\xi)$ since no place in $\mu(\xi)$ is of the form $+_1 b$. If $\mathbf{pre}_q(e_1) \subseteq \mu(q)$ let $b \in \mathbf{pre}_q(e_1)$. Since $\xi$ is not a CCS term, we have $\Gamma(\xi_0) \subset \mu(p)$ by the previous corollary, so let $b' \in \mu(p) - \Gamma(\xi_0)$. Then $(b' + b) \in \mathbf{pre}_{p+q}(+_1 e_1)$, therefore $\mathbf{pre}_{p+q}(+_1 e_1) \not\subseteq \mu(\xi)$. Thus indeed $e = +_0 e_0$ with $e_0$ fireable at $\mu(\xi_0)$. By induction $\xi_0 \xrightarrow{e_0} \xi_0'$ and thus $\xi_0 \hat{+}_0 q \xrightarrow{+_0 e_0} \xi_0' \hat{+}_0 q$ is the required transition.

(vi) If $\xi = \xi' \backslash \alpha$ then $e$ is fireable at $\mu(\xi)$ because $e = \backslash_\alpha e'$ with $\mathbf{nm}(\ell(e)) \neq \alpha$ and $e'$ fireable at $\mu(\xi')$. By induction $\xi' \xrightarrow{e'} \xi''$, hence $\xi' \backslash \alpha \xrightarrow{\backslash_\alpha e'} \xi'' \backslash \alpha$ is the required transition. ∎

Putting together the two propositions we obtain the announced correspondence result:

COROLLARY 6.21. *For any marked term $\xi$ which is reachable from its ancestor $p = \varrho(\xi)$, we have $\xi \xrightarrow{e} \xi'$ in the event system if and only if $\mu(\xi) \xrightarrow{e} \mu(\xi')$ in the net $\mathcal{N}(p)$.*

*Remark.* From the fact that both the event system and the transition system on markings in the net $\mathcal{N}(p)$ are deterministic and acyclic, it follows moreover that the event system starting from $p$ and the transition

system of the markings reachable from $\mu_p$ in the net $\mathcal{N}(p)$ are *isomorphic* transition systems. Then we have:

PROPOSITION 6.22. *For any FCCS term $p$, a set of events $X$ is a net computation of $p$ if and only if there exists a sequence of event transitions $\sigma \in \mathcal{T}_{ev}(p)$ such that $X = \mathrm{Ev}(\sigma)$.*

We have shown in the previous section, Lemma 6.15, that for sequences $\sigma_0$ and $\sigma_1$ of event transitions of a CCS term $p$, the following holds:

$$[\![\sigma_0]\!] \sqsubseteq [\![\sigma_1]\!] \Leftrightarrow \mathrm{Ev}(\sigma_0) \subseteq \mathrm{Ev}(\sigma_1).$$

Therefore an obvious consequence of the previous results is the announced correspondence between net computations and event computations:

THEOREM 6.23. *For any FCCS term $p$ the domains of computations $\mathcal{D}_{ev}(p) = (\mathcal{T}_{ev}(p)/\simeq, \sqsubseteq)$ and $\mathcal{D}_{net}(p) = (\mathcal{C}_{net}(p), \subseteq)$ are isomorphic. Moreover the configurations of the event structure $\mathcal{S}(p)$ are the same as the computations of the net $\mathcal{N}(p)$, therefore $\mathcal{D}_{es}(p) = \mathcal{D}_{net}(p)$.*

## APPENDIX: TRANSITIONS OF FLOW EVENT STRUCTURES

In [8] we defined a transition relation on flow event structures, where transitions are labelled by configurations. However, the definition of *remainder* was slightly incorrect there. We give here the amended definition. For any *finite* configuration $X$ of $S = (E, \prec, \#)$, let us define a flow event structure $S[X]$, the *remainder* of $S$ after $X$, by:

$$S[X] =_{\mathrm{def}} (E, \prec_X, \#_X) \text{ where } \begin{cases} \prec_X = \prec - \{(e, e') \mid \exists e'' \in X. e \sharp e'' \prec e'\} \\ \#_X = \# \cup \{(e, e) \mid \exists e' \in X. e \sharp e'\} \end{cases}$$

Note that $S[\varnothing] = S$. Note also that if $\exists e'. e \prec_X e'$ then $e \notin X$. Now we prove that this definition is correct, in the sense that configurations of $S[X]$ are exactly the computations that can be performed "after $X$," that is:

LEMMA 6.24. $Y \in \mathcal{F}^\infty(S[X]) \Leftrightarrow Y \cap X = \varnothing \;\&\; X \cup Y \in \mathcal{F}^\infty(S)$.

*Proof.* Let us assume that $X = \{e_1, ..., e_n\}$, where $e_1 \cdots e_n$ is a proving sequence.

(i) we first prove "$\Rightarrow$": assume that $Y = \{d_1, ..., d_m, ...\}$ is a configuration of $S[X]$, such that $d_1 \cdots d_m \cdots$ is a proving sequence of

$S[X]$. Since $Y$ does not contain two conflicting events, w.r.t. $\#_X$, we have $Y \cap X = \varnothing$, for $e \in X \Rightarrow e \#_X e$. Now let us show that $e_1 \cdots e_n d_1 \cdots d_m \cdots$ is a proving sequence of $S$:

(a) This sequence is made of distinct events; since this is the case for $e_1 \cdots e_n$ and $d_1 \cdots d_m \cdots$, and $Y \cap X = \varnothing$.

(b) Assume that two events of this sequence are in conflict, w.r.t. $\#$. Obviously these cannot be two events of $e_1 \cdots e_n$, nor two events of $d_1 \cdots d_m \cdots$, since $d \# d' \Rightarrow d \#_X d'$. Assume $e_i \# d_j$; then by definition of $\#_X$ we should have $d_j \#_X d_j$, but this is impossible since $d_1 \cdots d_m \cdots$ is conflict-free, w.r.t. $\#_X$. Therefore $e_1 \cdots e_n d_1 \cdots d_m \cdots$ is conflict-free.

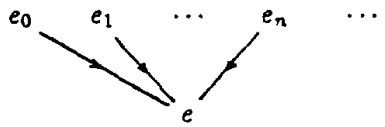
(c) If $e \prec d_j$ then, by definition of $\prec_X$, we have either $\exists i. e \sharp e_i \prec d_j$ or $e \prec_X d_j$. In the first case, we have finished. In the second case, since $d_1 \cdots d_m \cdots$ is a proving sequence of $S[X]$, we have $\exists h < j. e \sharp X\, d_h \prec_X d_j$. Then $d_h \prec d_j$, and, by definition of $\#_X$, either $e \sharp d_h$ and we have finished, or $e = d_h$ and $\exists i. e \sharp e_i$, and this implies $e = e_i$ since $X \cup Y$ is conflict-free w.r.t. $\#$. But $e = d_h = e_i$ is not possible, since $Y \cap X = \varnothing$. Thus $e \sharp d_h$.

(ii) now we prove "$\Leftarrow$": let us assume that $X \cup Y = \{f_1, ..., f_k, ...\}$, where $f_1 \cdots f_k \cdots$ is a proving sequence of $S$, and $Y \cap X = \varnothing$. We show that the subsequence $d_1 \cdots d_m \cdots$ consisting of the elements of $Y$ is a proving sequence of $S[X]$. Obviously the $d_j$'s are distinct.

(a) Assume $d_h \#_X d_j$. Then we would have either $d_h \# d_j$, which is clearly impossible since $X \cup Y$ is conflict-free w.r.t. $\#$, or $d_h = d_j$ and $\exists i. e_i \sharp d_j$. But this is impossible since $e_i = d_j$ would contradict $Y \cap X = \varnothing$, and $e_i \# d_j$ would contradict the fact that $X \cup Y$ is conflict-free w.r.t. $\#$.

(b) Let $d \prec_X d_j$. Then we have $d \prec d_j$; therefore if $d_j = f_l$ there exists $h < l$ such that $d \sharp f_h \prec d_j$. We cannot have $f_h \in X$, since this would contradict $d \prec_X d_j$; therefore $\exists i < j. f_h = d_i$, hence $d \sharp d_i \prec d_j$. Therefore, since $\# \subseteq \#_X$, we have $d \sharp_X d_j$. Finally $d_i \prec_X d_j$ since otherwise we would have either $d_i \in X$, contradicting $Y \cap X = \varnothing$, or $\exists e \in X. d_i \# e$, contradicting the fact that $X \cup Y$ is a configuration of $S$. ∎

*Remark.* The previous definition of $S[X]$ would not work in general for an infinite configuration $X$, as shown by the following example: let $S$ be the flow event structure, without conflict, given by




and $X = \{e_0, ..., e_n, ...\}$. Then $\{e\}$ would be a configuration of $S[X]$, whereas $X \cup \{e\}$ is not a configuration of $S$.

We may now define the *transitions* on flow event structures as follows:

$$S \xrightarrow{X} S[X] \quad \text{for} \quad X \in \mathscr{F}(S).$$

Then to prove Theorem 6.16, we could have defined the flow event structure interpretation of a marked term $\xi$, reachable from its ancestor $\varrho(\xi)$, as follows: $\mathscr{S}(\xi) = (\mathscr{S}(\varrho(\xi)))[\mathrm{Ev}(\sigma)]$, where $\sigma$ is a sequence of event transitions from $\varrho(\xi)$ to $\xi$. Then one could show a correspondence between these flow event structures transitions in the event system.

RECEIVED SEPTEMBER 30, 1991; FINAL MANUSCRIPT RECEIVED JUNE 9, 1992

## REFERENCES

1. BEDNARCZYK, M. A. (1987), "Categories of Asynchronous Systems," Ph.D. Thesis, University of Sussex.
2. BERGSTRA J. A., AND KLOP J. W. (1985), Algebra of communicating processes with abstraction, *Theoret. Comput. Sci.* **37**, 77–21.
3. BERRY, G., AND LÉVY, J.-J. (1979), Minimal and optimal computations of recursive programs, *J. Assoc. Comput. Mach.* **26**, 148–175.
4. BEST, E., AND DEVILLERS, R. (1987), Interleaving and partial orders in concurrency: A formal comparison, *in* "Formal Description of Programming Concepts III," pp. 299–321, North-Holland, Amsterdam.
5. BOUDOL, G. (1985), Computational semantics of term rewriting systems, *in* "Algebraic Methods in Semantics" (M. Nivat and J. C. Reynolds, Eds.), pp. 169–325, Cambridge Univ. Press, London/New York.
6. BOUDOL, G., AND CASTELLANI, I. (1987), On the semantics of concurrency: Partial orders and transition systems, *in* "TAPSOFT 87," pp. 123–137, Lecture Notes in Computer Science, Vol. 249, Springer-Verlag, Berlin/New York.
7. BOUDOL, G., AND CASTELLANI, I. (1988), Concurrency and atomicity, *Theoret. Comput. Sci.* **59**, 25–84.
8. BOUDOL, G., AND CASTELLANI, I. (1988), Permutation of transitions: An event structure semantics for CCS and SCCS, *in* "Linear Time, Branching Time and Partial Orders in Logics and Models for Concurrency," pp. 411–427, Lecture Notes in Computer Science, Vol. 354, Springer-Verlag, Berlin/New-York.
9. BOUDOL G., AND CASTELLANI, I. (1988), A non-interleaving semantics for CCS based on proved transitions, *Fund. Informat.* **11**, 433–452.
10. BOUDOL, G., AND CASTELLANI, I. (1991) "Flow Models of Distributed Computations: Flow Event Structures and Flow Nets," INRIA Research Report 1482; preliminary version in "Actes de l'Ecole de Printemps Sémantique du Parallélisme," Lecture Notes in Computer Science, Vol. 469, Springer-Verlag, Berlin/New York.
11. BROOKES, S., HOARE, C. A. R., AND ROSCOE, A. (1984), A theory of communicating sequential processes, *J. Assoc. Comput. Mach.* **31**, 560–599.
12. BROOKES, S., AND ROUNDS, W. C. (1983), Behavioural equivalence relations induced by programming logics, *in* "ICALP 83," pp. 97–108, Lecture Notes in Computer Science, Vol. 154, Springer-Verlag, Berlin/New York.
13. CASTELLANI I., AND HENNESSY, M. (1989), Distributed bisimulations, *J. Assoc. Comput. Mach.* **36**, 887–911.

14. CASTELLANI, I. (1988), "Bisimulations for Concurrency," Ph.D. Thesis, CST-51-88, University of Edinburgh.
15. CASTELLANI, I., AND ZHANG, G. Q. (1989), "Parallel Product of Event Structures," INRIA Research Report 1078 and DIAMI Report PB-285, Aarhus University.
16. DARONDEAU, PH., AND DEGANO, P. (1989), Causal trees, *in* "ICALP 89," pp. 234–248, Lecture Notes in Computer Science, Vol. 372, Springer-Verlag, Berlin/New York.
17. DEGANO, P., DE NICOLA, R., AND MONTANARI, U. (1985), Partial ordering derivations for CCS, *in* "FCT 85," pp. 520–533, Lecture Notes in Computer Science, Vol. 199, Springer-Verlag Berlin/New York.
18. DEGANO, P., DE NICOLA, R., AND MONTANARI, U. (1987), Concurrent histories: A basis for observing distributed systems, *J. Comput. System Sci.* **34**, 422–461.
19. DEGANO, P., DE NICOLA, R., AND MONTANARI, U. (1988), On the consistency of "truly concurrent" operational and denotational semantics, *in* "LICS 88," pp. 133–141.
20. DEGANO, P., DE NICOLA, R., AND MONTANARI, U. (1988), A distributed operational semantics for CCS based on condition/event systems, *Acta Informat.* **26**, 59–91.
21. DEGANO, P., MESSEGUER, J., AND MONTANARI, U. (1989), Axiomatizing net computations and processes, *in* "LICS 89," pp. 175–185.
22. DE NICOLA, R. (1987), Extensional equivalences for transition systems, *Acta Inform.* **24**, 211–237.
23. FERRARI, G. L., AND MONTANARI, U. (1990), Towards the unification of models for concurrency, *in* "CAAP 90," pp. 162–176, Lecture Notes in Computer Science, Vol. 431, Springer-Verlag, Berlin/New York.
24. FERRARI, G. L. (1990), "Unifying Models of Concurrency," Ph.D. Thesis, University of Pisa.
25. GENRICH, H. J., LAUTENBACH, K., AND THIAGARAJAN, P. S. (1980) Elements of general net theory, *in* "Net Theory and Applications" (W. Brauer, Ed.), pp. 21–163, Lecture Notes in Computer Science, Vol. 84, Springer-Verlag, Berlin/New York.
26. VAN GLABBEEK, R., AND VAANDRAGER, F. (1987), Petri net models for algebraic theories of concurrency, *in* "Proceedings PARLE Conference, Eindhoven," pp. 224–242, Lecture Notes in Computer Science, Vol. 259, Springer-Verlag, Berlin/New York.
27. VAN GLABBEEK R., AND GOLTZ, U. Equivalence notions for concurrent systems and refinement of actions, *in* "MFCS 89," pp. 237–248, Lecture Notes in Computer Science, Vol. 379, Springer-Verlag, Berlin/New York.
28. VAN GLABBEEK R., AND GOLTZ, U. (1990), Equivalence notions and refinement of actions for flow event structures, draft.
29. GOLTZ, U., AND MYCROFT, A., (1984), On the relationship of CCS and Petri nets, *in* "ICALP 84," pp. 196–208, Lecture Notes in Computer Science, Vol. 172, Springer-Verlag, Berlin/New York.
30. GOLTZ, U. (1986), "Building structured Petri nets," Arbeitspapiere der GMD, No. 223.
31. GOLTZ, U. (1988), On representing CCS programs by finite Petri nets, *in* "MFCS 88," pp. 339–350, Lecture Notes in Computer Science, Vol. 324, Berlin/New York.
32. GOLTZ, U., AND LOOGEN, R. (1991), Modelling nondeterministic concurrent processes with event structures, *Fund. Informat.* **14** 39–73.
33. HENNESSY, M. (1988), "An Algebraic Theory of Processes," MIT Press, Cambridge, MA.
34. HENNESSY, M. (1988), Observing processes, *in* "Linear Time, Branching Time and Partial Orders in Logics and Models for Concurrency," 173–200, Lecture Notes in Computer Science, Vol. 354, Springer-Verlag, Berlin/New York.
35. HUET, G., AND LÉVY, J.-J. (1979), "Call-by-Need Computations in Non-ambiguous Linear Term Rewriting Systems," IRIA-LABORIA Report 359.
36. KWIATKOWSKA, M. Z. (1989), "Fairness for Non-interleaving Concurrency, Ph.D. Thesis, Department of Computing Studies, University of Leicester, Technical Report 22.

37. LÉVY, J.-J. (1980), Optimal reductions in the lambda calculus, *in* "To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism" (J. P. Seldin and J. R. Hindley, Eds), pp. 159–191, Academic Press, San Diego.
38. MAZURKIEWICZ, A. (1979) Concurrent program schemes and their interpretations, *in* "Aarhus Workshop on Vertification of Parallel Programs, Diami PB-78," Aarhus University.
39. MAZURKIEWICZ, A. (1988), Basic notions of trace theory, *in* "Linear Time, Branching Time and Partial Orders in Logics and Models for Concurrency," pp. 285–363, Lecture Notes in Computer Science, Vol. 354, Springer-Verlag, Berlin/New York.
40. MILNER, R. (1980), "A Calculus of Communicating Systems," Lecture Notes in Computer Science, Vol. 92, reprinted in Report ECS-LFCS-86-7, Edinburgh University.
41. MILNER R. (1988), "Operational and Algebraic Semantics of Concurrent Processes," LFCS Report ECS-LFCS-88-46, Edinburgh University.
42. MILNER R. (1989), "Communication and Concurrency," International Series in Computer Science, Prentice-Hall, Englewood Cliffs, NJ.
43. NIELSEN, M., PLOTKIN, G., AND WINSKEL, G., (1981), Petri Nets, event structures and domains, *Theoret. Comput. Sci.* **13**, 85–108.
44. NIELSEN, M., ROZENBERG, G., AND THIAGARAJAN, P. S. (1990), "Elementary Transition Systems," Diami PB-310, AarhusUniversity; to appear in *Theoret. Comput. Sci.*
45. OLDEROG, E.-R. (1987), Operational Petri net semantics for CCSP, *in* "Advances in Petri Nets 87," pp. 196–223, Lecture Notes in Computer Science, Vol. 266, Springer-Verlag, Berlin/New York.
46. PETRI, C. A. (1977), "Non-Sequential Processes," GMD-ISF Report 77-05.
47. PLOTKIN, G. (1981), "A Structural Approach to Operational Semantics," Daimi FN-19, Aarhus University.
48. PRATT, V. R. (1986), Modelling concurrency with partial order, *Int. J. Parallel Programming* **15**, 33–71.
49. REISIG, W. (1985), "Petri Nets: An Introduction," EATCS Monographs, Springer-Verlag, Berlin/New York.
50. B. ROZOY, (1990), On distributed languages and models for distributed computation, *in* "Semantics of Systems of Concurrent Processes," pp. 434–456, Lecture Notes in Computer Science, Vol. 469, Springer-Verlag, Berlin/New York.
51. SHIELDS, M. W. (1985), Deterministic asynchronous automata, *in* "Formal Methods in Programming," North-Holland, Amsterdam.
52. STARK, E. W. (1989) Concurrent transition systems, *Theoret. Comput. Sci.* **64**, 221–269.
53. TAUBNER, D. (1990), Representing CCS programs by finite predicate/transition nets, *Acta Informat.* **27**, 533–565.
54. THIAGARAJAN, P. S. Some behavioural aspects of net theory, *in* "ICALP 88," pp. 630–653, Lecture Notes in Computer Science, Vol. 317, Springer-Verlag, Berlin/New York.
55. VAANDRAGER, F. (1989), "A Simple Definition for Parallel Composition of Prime Event Structures," Technical Report CS-R8903, CWI, Amsterdam.
56. WINSKEL, G. (1982), "Event Structure Semantics for CCS and Related Languages," Daimi PB-159, Aarhus University; *in* "9th ICALP, pp. 561–576, Lecture Notes in Computer Science," Vol. 140, Springer-Verlag, Berlin/New York.
57. WINSKEL, G. (1985), Categories of models for concurrency, *in* "Seminar on Concurrency," pp. 246–267, Lecture Notes in Computer Science, Vol. 197, Springer-Verlag, Berlin/New York.
58. WINSKEL, G. (1987), Event structures, *in* "Advances in Petri Nets 86," pp. 325–392, Lecture Notes in Computer Science, Vol. 255, Springer-Verlag, Berlin/New York.
59. WINSKEL G., (1988), An introduction to event structures, *in* "Linear Time, Branching Time and Partial Orders in Logics and Models for Concurrency," pp. 364-397, Lecture Notes in Computer Science, Vol. 354, Springer-Verlag, Berlin/New York.