Bundle Event Structures and CCSP*

Rob van Glabbeek[1] and Frits Vaandrager[2]

[1] National ICT Australia
School of Computer Science & Engineering, Univ. of New South Wales, Sydney 2052, Australia
rvg@cs.stanford.edu
[2] Nijmegen Institute for Computing and Information Sciences, University of Nijmegen
Postbus 9010, 6500 GL Nijmegen, The Netherlands
fvaan@cs.kun.nl

Abstract. We investigate which event structures can be denoted by means of closed CCS $\cup$ CSP expressions. Working up to isomorphism we find that

- all denotable event structures are bundle event structures,
- upon adding an infinitary parallel composition all bundle event structures are denotable,
- without it every finite bundle event structure can be denoted,
- as well as every countable prime event structure with binary conflict.

Up to hereditary history preserving bisimulation equivalence finitary conflict can be expressed in terms of binary conflict. In this setting all countable stable event structures are denotable.

Introduction

In concurrency theory many languages for the representation of concurrent systems have been proposed, including CCS, SCCS, CSP, MEIJE, ACP, COSY and LOTOS, all in several variations. Although most of these languages were originally equipped with an interleaving semantics, concurrency respecting interpretations have been proposed by various authors, using semantical models like Petri nets, event structures, transition systems—optionally with additional structure to represent causal independence—, causal trees, families of posets, etc. In recent years it has been established that there are canonical translations between most of these models, thereby making them into different representations of one and the same semantic concept [13, 17, 2, 19, 6]. In addition, the languages mentioned above are to a large extent intertranslatable, and can be regarded as dialects of one and the same system specification language.

This paper deals with the question which of these unified semantic objects can be denoted by closed expressions in this unified language. As a representative semantic model we take the *event structures* from WINSKEL [17]. Our findings can then be transmitted to other models by means of the canonical translations found in the literature. As a representative language we combine some operators from CCS [12] and CSP [3,

* To appear in: Proceedings 14th International Conference on Concurrency Theory (CONCUR 2003), Marseille, France, LNCS, Springer, September 2003.

[1] This work was performed in part while the first author was employed at Stanford University. It was supported by ONR under grant number N00014-92-J-1974.

9]. Following a suggestion of Mogens Nielsen, such a combination is called CCSP. Our version of CCSP is sufficiently expressive to emulate most constructions from other languages found in the literature, including the ones provided with an event structure semantics in [17]. The chosen combination of operators appears to be optimal for carrying out the constructions in this paper. However, many other combinations would lead to the same results.

In [17] the subclass of *stable event structures* is defined, as well as the further subclass of *prime event structures*. In [18] a subclass of event structures with a *binary conflict relation* is proposed (see Figure 1 below). The prime event structures with binary conflict are exactly the (finitary) event structures originally introduced in [13]. It is well known that unstable event structures cannot be represented in CCSP-like languages in a causality respecting way. It is an interesting quest to extend such languages with novel operators that make this possible. This quest is not pursued here; we will be happy to just find out which of the stable event structures are denotable.

It is unreasonable to expect to find a CCSP expression denoting a given event structure exactly. Hence we will try to find for any given stable event structure a CCSP expression whose denotation as event structure is semantically equivalent. This makes our quest parametrised by the choice of a suitable semantic equivalence. We consider three choices for this parameter: *isomorphism*, *history preserving bisimulation equivalence* and (in this introduction only) *ST-bisimulation equivalence*.

Denotability up to isomorphism Up to isomorphism we characterise the denotable event structures as the *bundle event structures* proposed in LANGERAK [11]. As we will recall in Section 1, these include all prime event structures with binary conflict, and are included in the stable event structures with binary conflict (cf. Figure 1). In [11] examples can be found showing that these inclusions are strict.

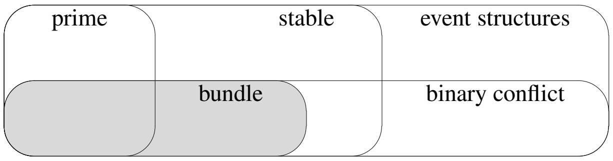

Fig. 1. *Several classes of event structures*

Our characterisation of the bundle event structures as the event structures that can be expressed by CCSP expressions is exact when dealing with the original finite bundle event structures and recursion-free CCSP. Our characterisation is also exact when dealing with arbitrary infinite bundle event structures and a version of CCSP with an infinite parallel composition operator. However, when dealing with countable event structures and CCSP expressions with arbitrary systems of recursion equations, or with recursive enumerable event structures and a recursive enumerable version of CCSP, all we can show is that the denotable event structures are a subclass of the bundle event structures that include all prime event structures with binary conflict.

In Section 2 a denotational semantics of CCSP is given in terms of event structures with binary conflict. This semantics follows the standard lines of [17, 18]. In the same section we show that the class of bundle event structures is closed under the CCSP operators, thereby establishing that CCSP expressions can denote bundle event structures only. Along the same lines one can show that the bundle event structures are closed under *action refinement* [5], the choice operators $\Box$ and $\sqcap$ of CSP, and many other operators found in the literature. We are not aware of any operator interpreted on event structures for which the class of bundle event structures is not closed.

In Section 3 we show that up to isomorphism

- every *finite* bundle event structure can be denoted by a recursion-free CCSP expression,
- every *countable* prime event structure with binary conflict can be denoted by a CCSP expression,
- and *every* bundle event structure can be denoted by a CCSP expression with an infinitary parallel composition.

We also provide a recursive enumerable version of the second result. The same results can be obtained for the language of [17, 18].

Denotability up to hereditary history preserving bisimulation equivalence The concept of *history preserving bisimulation equivalence* stems from RABINOVICH & TRAKHTENBROT [15] and was adapted to event structures in VAN GLABBEEK & GOLTZ [5]. There it was suggested that the notion could be regarded as the coarsest equivalence that takes the interplay of causality and branching time completely into account. This makes the equivalence a semantically interesting choice of parameter to instantiate our quest with. We arrive at the positive conclusion that up to history preserving bisimulation *every* countable stable event structure can be denoted by a CCSP expression. This result is obtained in three steps, the first of which is the aforementioned denotability by CCSP expressions of countable prime event structures with binary conflict. In Section 4 we extend this to countable stable event structures, by observing that that every countable stable event structure with binary conflict is history preserving bisimulation equivalent with a countable prime event structure with binary conflict.

In Section 5 we complete the proof by showing that every countable stable event structure is history preserving bisimulation equivalent with a countable stable event structure with binary conflict. This result was first claimed by us in [8] for finite prime event structures. The claim was strengthened in [6] to include infinite ones. The first published proof (for prime event structures) appears in NIELSEN & WINSKEL [14], who discovered the result independently. Their proof is somewhat nonconstructive however, in the sense that there is no construction giving a specific countable event structure with binary conflict for any given countable stable event structure with arbitrary conflict. Our proof offers such a construction and is somewhat shorter as well.

The results above hold even when merely working up to *hereditary history preserving bisimulation equivalence*, which is a finer variant of history preserving bisimulation equivalence, proposed by BEDNARCZYK [1].

Denotability up to ST-bisimulation equivalence The coarser *ST-bisimulation equivalence*, proposed in VAN GLABBEEK & VAANDRAGER [7], respects branching time and the possibility of actions to overlap in time, but abstracts from the faithful modelling of causality. By Theorem 1 in VAN GLABBEEK & PLOTKIN [6], every event structure, stable or not, is ST-bisimulation equivalent to a prime event structure. This result keeps being valid when assuming and requiring countability. It follows that up to ST-bisimulation equivalence every event structure can be denoted by a CCSP expression.

1 Bundle event structures

Bundle event structures are introduced in LANGERAK [11]. Here we add the alphabets for *typed* bundle event structures and generalise the notion to structures with infinite sets of events.

Definition 1. A *(typed) bundle event structure* is a 5-tuple $\mathrm{E} = (E, \#, \mapsto, A, l)$ where

- E is a set of *events*,
- $\# \subseteq E \times E$ is an irreflexive and symmetric relation, the *conflict relation*,
- $\mapsto \subseteq 2^E \times E$ is the *bundle set*, satisfying $X \mapsto e \Rightarrow \forall e_1, e_2 \in X.\, e_1 \neq e_2 \Rightarrow e_1 \# e_2$,
- A is a set of *actions*, the *alphabet* of E,
- and $l : E \to A$ is the *labelling function*.

A bundle event structure represents a concurrent system in the following way: action names $a \in A$ represent actions the system might perform and an event $e \in E$ labelled with a represents an occurrence of a during a possible run of the system. In order for e to happen it is necessary that for every bundle $X \mapsto e$, one of the elements of X occurred previously. The conflict $d\#e$ means that the events d and e cannot happen both in the same run.

The components of a bundle event structure E will be denoted by $E_{\mathrm{E}}, \#_{\mathrm{E}}, \mapsto_{\mathrm{E}}, A_{\mathrm{E}}$ and l_{E}; a convention that will also apply to other structures given as tuples.

The behaviour of a bundle event structure is described by explaining which subsets of events constitute possible (partial) runs of the represented system (thus formalising the interpretation of the bundle sets and the conflict relation). These subsets are called *configurations*. The causal relationships between events in a configuration x can be represented by a partial order $<_x$.

Definition 2. The set $C(\mathrm{E})$ of (finite) *configurations* of a bundle event structure $\mathrm{E} = (E, \#, \mapsto, A, l)$ consists of those finite $x \subseteq E$ which are

- *conflict-free*: $\# \cap (x \times x) = \emptyset$,
- and *secured*:
  $\exists e_1, ..., e_n \, (n \geq 0) :\, x = \{e_1, ..., e_n\} \wedge \forall i < n \, (Y \mapsto e_{i+1} \Rightarrow \{e_1, ..., e_i\} \cap Y \neq \emptyset)$.

The *causality relation* $<_x$ on $x \in C(\mathrm{E})$ is $\{(d, e) \in x \times x \mid \exists Y : d \in Y \mapsto e\}^+$. Here R^+ denotes the transitive closure of a relation R.

Following [5], we only consider finite configurations here; since the infinite configurations which are usually considered are completely determined by the finite ones, this causes no loss of generality. Note that if $e \in x \in C(\mathrm{E})$ and $Y \mapsto_{\mathrm{E}} e$ then $x \cap Y$ has exactly one element. Hence $<_x$ is always a partial order.

We now define the *prime* and the *stable* event structures with binary conflict, stemming from WINSKEL [18], and show that the bundle event structures can be regarded as a generalisation of the former and a special case of the latter.

Definition 3. A *(typed) prime event structure with binary conflict* is a 5-tuple $\mathrm{E} = (E, <, \#, A, l)$ where

- E, A, and l are as above,
- $< \subseteq E \times E$ is a partial order such that $\forall e \in E : \{d \in E \mid d < e\}$ is finite,
- and $\# \subseteq E \times E$ is an irreflexive, symmetric relation satisfying

$$\forall d, e, f \in E : d < e \wedge d \# f \Rightarrow e \# f.$$

Here $d < e$ means that d is a prerequisite for e. Prime event structures with binary conflict can be regarded as special bundle event structures, by defining

$$X \mapsto e \quad \Leftrightarrow \quad X = \{d\} \wedge d < e.$$

The definition of configurations given above is then consistent with the one in [18].

Definition 4. A *(typed) event structure with binary conflict* is a 5-tuple $\mathrm{E} = (E, \#, \vdash, A, l)$ where

- E, $\#$, A, and l are as for bundle event structures,
  $Con = \{X \subseteq E \mid X \text{ finite and } d\#e \text{ for no } d, e \in X\}$ is the *consistency predicate*,
- and $\vdash \subseteq Con \times E$ is the *enabling relation*, satisfying

$$X \vdash e \wedge X \subseteq Y \in Con \Rightarrow Y \vdash e.$$

E is *stable* if $Y \vdash e$ implies that there is a least subset X of Y with $X \vdash e$.

$X \vdash e$ means that X is a *possible cause* of e in the sense that e can occur only if for certain Y with $Y \vdash e$ all events in Y have occurred before.

Definition 5. The set $C(\mathrm{E})$ of (finite) *configurations* of an event structure with binary conflict $\mathrm{E} = (E, \#, \vdash, A, l)$ consists of those finite $x \subseteq E$ which are

- *conflict-free*: $\# \cap (x \times x) = \emptyset$, i.e. $x \in Con$,
- and *secured*: $\exists e_1, ..., e_n \ (n \geq 0) : x = \{e_1, ..., e_n\} \wedge \forall i < n \ \{e_1, ..., e_i\} \vdash e_{i+1}$.

The *causality relation* $<_x$ on x is $\{(d, e) \in x \times x \mid \forall Y : Y \vdash e \Rightarrow d \in Y\}^+$.

The causality relation gives a faithful description of the causal relations in a configuration only if E is stable. As shown in [17, 18], unstable event structures can model causal relationships that cannot be captured in terms of partial orders. The following shows how bundle event structures can be regarded as special stable event structures with binary conflict.

Definition 6. Given a bundle event structure $\mathrm{E} = (E, \#, \mapsto, A, l)$, the *associated* event structure with binary conflict $\mathcal{E}(\mathrm{E}) = (E, \#, \vdash, A, l)$ is given by

$$X \vdash e \quad \Leftrightarrow \quad X \in Con \wedge \forall Y : (Y \mapsto e \Rightarrow X \cap Y \neq \emptyset).$$

Proposition 1. *$\mathcal{E}(\mathrm{E})$ is always stable. Moreover, the translation $\mathcal{E}$ preserves configurations and the causality relations $<_x$ on them.*

Proof. Straightforward.

2 A denotational event structure semantics of CCSP

CCSP is parametrised by the choice of an infinite set Act of actions, that we will assume to be fixed for this paper. We also assume an infinite set V of *variable names*. A *variable* is a pair X_A with $X \in V$ and $A \subseteq Act$. The syntax of CCSP is given by

$$P ::= 0_A \;\Big|\; aP \;\Big|\; P+P \;\Big|\; P\|P \;\Big|\; R(P) \;\Big|\; X_A \;\Big|\; \langle X_A|S\rangle \text{ (with } X_A \in V_S)$$

with $A \subseteq Act$, $a \in Act$, $R \subseteq Act \times Act$, $X \in V$ and S a *recursive specification*: a set of equations $\{X_A = P_{X_A} \mid X_A \in V_S\}$ with $V_S \subseteq V \times Act$ (the *bound variables* of S) and $\alpha(P_{X_A}) = A$ for all $X_A \in V_S$ (were $\alpha(P_{X_A})$ is defined below). The constant 0_A represents a process that is unable to perform any action. The process aP first performs the action a and then proceeds as P. The process $P + Q$ will behave as either P or Q, $\|$ is a partially synchronous parallel composition operator, R a renaming, and $\langle X_A|S\rangle$ represents the X_A-component of a solution of the system of recursive equations S. A CCSP expression P is *closed* if every occurrence of a variable X_A occurs in a subexpression $\langle Y_B|S\rangle$ of P with $X_A \in V_S$. An expression $a0_\emptyset$ is abbreviated a.

Just like the version of CSP from HOARE [9], the version of CCSP used here is a typed language, in the sense that with every process P an explicit alphabet $\alpha(P) \subseteq Act$ is associated, which is a superset of the set of all actions the process could possibly perform. This alphabet is exploited in the definition of $P\|Q$: actions in the intersection of the alphabets of P and Q are required to synchronise, whereas all other actions of P and Q happen independently. Because of this, processes with different alphabets may never be identified, even if they can perform the same set of actions and are alike in all other aspects. It is for this reason that we interpret CCSP in terms of *typed* event structures. The constant 0 and the variables are indexed with an alphabet. The alphabet of an arbitrary CCSP expression is given by:

- $\alpha(0_A) = \alpha(X_A) = \alpha(\langle X_A|S\rangle) = A$
- $\alpha(aP) = \{a\} \cup \alpha(P)$
- $\alpha(P+Q) = \alpha(P\|Q) = \alpha(P) \cup \alpha(Q)$
- $\alpha(R(P)) = \{b \mid \exists a \in \alpha(P) : (a,b) \in R\}$.

Substitutions of expressions for variables are allowed only if the alphabets match. For this reason a recursive specification S is declared syntactically incorrect if $\alpha(P_{X_A}) \neq A$ for some $X_A \in V_S$.

Below we define the CCSP operators formally on the domains of (typed) bundle and stable event structures with binary conflict. As our bundle and stable interpretations agree on the components E, $\#$, A and l, they will be given as 6-tuples $(E, \#, \mapsto, \vdash, A, l)$, so that the bundle interpretation is found by dropping $\vdash$, and the stable interpretation by dropping $\mapsto$. When E is an event structure representing a CCSP expression P then $A_{\mathrm{E}} = \alpha(P)$. Hence we can abstain from explicitly mentioning the A-component in the forthcoming constructions.

Definition 7. The operators of CCSP are defined on event structures as follows:
Inaction: $\mathrm{E}_{0_A} = (\emptyset, \emptyset, \emptyset, \emptyset, A, \emptyset)$.
Action prefix (for $a \in Act$):

- $E_{a\mathrm{E}} = \{a\} \cup \{\hat{a}e \mid e \in E_{\mathrm{E}}\}$
- $l_{a\mathrm{E}}(a) = a$ and $l_{a\mathrm{E}}(\hat{a}e) = l_{\mathrm{E}}(e)$
- $\#_{a\mathrm{E}} = \{(\hat{a}e, \hat{a}e') \mid e \#_{\mathrm{E}} e'\}$
- $\mapsto_{a\mathrm{E}} = \{(\hat{a}X, \hat{a}e) \mid X \mapsto_{\mathrm{E}} e\} \cup \{(\{a\}, \hat{a}e) \mid e \in E_{\mathrm{E}}\}$ in which $\hat{a}X = \{\hat{a}e \mid e \in X\}$
- $\vdash_{a\mathrm{E}} = \{(X, a) \mid X \in Con_{a\mathrm{E}}\} \cup \{(\hat{a}X \cup \{a\}, \hat{a}e) \mid X \vdash_{\mathrm{E}} e\}$.

Alternative composition:

- $E_{\mathrm{E}_1+\mathrm{E}_2} = \{+_1 e \mid e \in E_{\mathrm{E}_1}\} \cup \{+_2 e \mid e \in E_{\mathrm{E}_2}\}$
- $l_{\mathrm{E}_1+\mathrm{E}_2}(+_i e) = l_{\mathrm{E}_i}(e)$ $(i = 1, 2)$
- $\#_{\mathrm{E}_1+\mathrm{E}_2} = \{(+_i e, +_i e') \mid e \#_{\mathrm{E}_i} e',\ i = 1, 2\} \cup \{(+_i e, +_j f) \mid e \in E_{\mathrm{E}_i},\ f \in E_{\mathrm{E}_j},\ i \neq j\}$
- $\mapsto_{\mathrm{E}_1+\mathrm{E}_2} = \{(+_i X, +_i e) \mid X \mapsto_{\mathrm{E}_i} e,\ i = 1, 2\}$ in which $+_i X = \{+_i e \mid e \in X\}$
- $\vdash_{\mathrm{E}_1+\mathrm{E}_2} = \{(+_i X, +_i e) \mid X \vdash_{\mathrm{E}_i} e,\ i = 1, 2\}$.

Parallel composition:

- $E_{\mathrm{E}\|\mathrm{F}} = \{(e\|*) \mid l_{\mathrm{E}}(e) \notin A_{\mathrm{F}}\} \cup \{(*\|f) \mid l_{\mathrm{F}}(f) \notin A_{\mathrm{E}}\} \cup \{(e\|f) \mid l_{\mathrm{E}}(e) = l_{\mathrm{F}}(f) \in A_{\mathrm{E}} \cap A_{\mathrm{F}}\}$
- $l_{\mathrm{E}\|\mathrm{F}}(e\|*) = l_{\mathrm{E}}(e)$, $l_{\mathrm{E}\|\mathrm{F}}(*\|f) = l_{\mathrm{F}}(f)$ and $l_{\mathrm{E}\|\mathrm{F}}(e\|f) = l_{\mathrm{E}}(e) = l_{\mathrm{F}}(f)$
- $\#_{\mathrm{E}\|\mathrm{F}} = \{(e\|f, e'\|f') \mid (e\|f \neq e'\|f') \wedge (e \#_{\mathrm{E}} e' \vee e = e' \neq * \vee f \#_{\mathrm{F}} f' \vee f = f' \neq *)\}$
- $\mapsto_{\mathrm{E}\|\mathrm{F}} = \{(X\|\mathrm{F}, e\|f) \mid X \mapsto_{\mathrm{E}} e\} \cup \{(\mathrm{E}\|Y, e\|f) \mid Y \mapsto_{\mathrm{F}} f\}$
  in which $X\|\mathrm{F} = \{(e\|f) \in E_{\mathrm{E}\|\mathrm{F}} \mid e \in X\}$ and $\mathrm{E}\|Y = \{(e\|f) \in E_{\mathrm{E}\|\mathrm{F}} \mid f \in Y\}$
- $\vdash_{\mathrm{E}\|\mathrm{F}} = \{(X, e\|f) \mid (e = * \vee \pi_1(X) \vdash_{\mathrm{E}} e) \wedge (f = * \vee \pi_2(X) \vdash_{\mathrm{F}} f)\}$
  in which $\pi_1(X) = \{e \in E_{\mathrm{E}} \mid \exists f \in E_{\mathrm{F}} \cup \{*\} : e\|f \in X\}$
  and $\pi_2(X) = \{f \in E_{\mathrm{F}} \mid \exists e \in E_{\mathrm{E}} \cup \{*\} : e\|f \in X\}$.

Relational renaming (for $R \subseteq Act \times Act$):

- $E_{R(\mathrm{E})} = \{R_b e \mid e \in E_{\mathrm{E}},\ (l_{\mathrm{E}}(e), b) \in R\}$
- $l_{R(\mathrm{E})}(R_b e) = b$
- $\#_{R(\mathrm{E})} = \{(R_b e, R_c e') \mid e \#_{\mathrm{E}} e' \vee (e = e' \wedge b \neq c)\}$
- $\mapsto_{R(\mathrm{E})} = \{(R(X), R_b e) \mid X \mapsto_{\mathrm{E}} e\}$
  in which $R(X) = \{R_b e \mid e \in X \wedge (l_{\mathrm{E}}(e), b) \in R\}$
- $\vdash_{R(\mathrm{E})} = \{(X, R_b e) \mid X \in Con \wedge R^{-1}(X) \vdash e\}$
  in which $R^{-1}(X) = \{e \in E_{\mathrm{E}} \mid \exists b \in A_{R(\mathrm{E})} : R_b e \in X\}$.

The semantics for 0, $a\mathrm{E}$, $\mathrm{E} + \mathrm{F}$ and $\mathrm{E}\|\mathrm{F}$ follows the lines of [17, 18, 2, 11]. *Relational renaming* appears in [16] and [4]. For every relation $R \subseteq Act \times Act$ there is an operator

R that replaces each occurrence of an action a by fresh occurrences of the actions $\{b \mid aRb\}$. These occurrences are pairwise in conflict, and inherit their causal relationships from their source. The *relabelling* operators of CCS [12], CSP [3] and WINSKEL [18] are special cases where R is a function; the *inverse image operator* of CSP [3] is the special case where R is the inverse of a function. In case $R(a) = \emptyset$, the definition implies that the events labelled a are removed; thus also the *restriction* operators of CCS and [18] constitute special cases of relational renaming. Relational renaming in turn is a special case of *action refinement*, as studied for instance in [5]. Also note that every relational renaming operator can be written as the composition of an inverse image operator and a functional renaming operator.

The meaning of the recursion constructs $\langle X_A|S\rangle$ can be given by means of least fixed point techniques, see e.g. [17, 18]. The fact that we allow recursive specifications of arbitrary size (in [17, 18] they are of size 1) does not create complications; we will not repeat the definitions here. Following the standard denotational approach this yields a bundle event structure $[\![P]\!]$ and a general event structure $[\![P]\!]_{\mathcal{E}}$ for every closed CCSP expression P. For open CCSP expressions $[\![P]\!]$ and $[\![P]\!]_{\mathcal{E}}$ are functions from valuations of the variables to event structures.

Proposition 2. *For every CCSP expression P we have $\mathcal{E}([\![P]\!]) = [\![P]\!]_{\mathcal{E}}$. Hence the bundle event structures, seen as a subclass of the stable event structures, are closed under the operators of CCSP.*

Proof. Straightforward with Definition 7.

3 Denoting bundle event structures in CCSP

In this section we address the question which event structures can be denoted by closed CCSP expressions of various kinds. As the events in structures $[\![P]\!]$ with P a closed CCSP expression have very particular names, whose choice seems to carry little semantic relevance, it is for this purpose most appropriate to study event structures up to isomorphism. Here two event structures E and F are *isomorphic* ($\mathrm{E} \cong \mathrm{F}$) iff $A_{\mathrm{E}} = A_{\mathrm{F}}$ and there exists a bijection between their sets of events preserving $\mapsto$ (resp. $\vdash$), $\#$ and labelling. Later we will see if the class of denotable event structures increases when considering a coarser equivalence.

The following proposition allows us to exchange, within a recursion-free CCSP expression, a closed subexpression by another expression denoting an isomorphic event structure.

Proposition 3. **(Congruence)** *Let E, E' and F be event structures with $\mathrm{E} \cong \mathrm{E}'$. Then $a\mathrm{E} \cong a\mathrm{E}'$ for $a \in Act$, $\mathrm{E}+\mathrm{F} \cong \mathrm{E}'+\mathrm{F}$, $\mathrm{F}+\mathrm{E} \cong \mathrm{F}+\mathrm{E}'$, $\mathrm{E}\|\mathrm{F} \cong \mathrm{E}'\|\mathrm{F}$, $\mathrm{F}\|\mathrm{E} \cong \mathrm{F}\|\mathrm{E}'$ and $R(\mathrm{E}) \cong R(\mathrm{E}')$ for $R : Act \to Act$.*

Proof. Immediate from the definitions.

The next one, essentially due to HOARE [9], allows us to drop brackets and abstract from the order of components in nested parallel compositions.

Proposition 4. *Let* E, F *and* G *be event structures. Then* $\mathrm{E}\|(\mathrm{F}\|\mathrm{G}) \cong (\mathrm{E}\|\mathrm{F})\|\mathrm{G}$ *and* $\mathrm{E}\|\mathrm{F} \cong \mathrm{F}\|\mathrm{E}$.

Proof. Straightforward.

Now we are ready to state the main theorems. We start with the simplest case of finite prime event structures with binary conflict.

Theorem 1. *For every finite prime event structure with binary conflict* E *there is a closed recursion-free CCSP expression* P *such that* $[\![P]\!] \cong \mathrm{E}$.

Proof. As we are interested in E only up to isomorphism, w.l.o.g. we may assume that $E_{\mathrm{E}} \subset Act$, i.e. the names of the events can also be used as names of CCSP actions. Let E' be the variant of E in which every event is labelled by itself. We first build a CCSP expression denoting E' (up to isomorphism), by encoding all events and all elements of the conflict and causality relation of E' in terms of CCSP constructions. Subsequently, an expression for E is obtained by applying a renaming operator. Let $E_{\mathrm{E}} = \{a_1, ..., a_n\}$, $\#_{\mathrm{E}} = \{(b_1, c_1), ..., (b_m, c_m)\}$ and $<_{\mathrm{E}} = \{(d_1, e_1), ..., (d_k, e_k)\}$. Then

$$P = l_{\mathrm{E}}\left[a_1\| \cdots \|a_n\|(b_1 + c_1)\| \cdots \|(b_m + c_m)\|(d_1 e_1)\| \cdots \|(d_k e_k)\right] + 0_{A_{\mathrm{E}}}.$$

Here l_{E} is not only the labelling function of E, but also one of the renaming operators of CCSP. Note that the actions b_i and c_i ($i = 1, ..., m$), as well as d_j and e_j ($j = 1, ..., k$) are among the actions $a_1, ..., a_n$. We have that $[\![a\|(a+b)]\!] \cong [\![a+b]\!]$ and $[\![a\|(ab)]\!] \cong [\![b\|(ab)]\!] \cong [\![ab]\!]$. Hence it would suffice to list as $a_1, ..., a_n$ only those events not in conflict or in any causal relationship with another event. It is routine to check that the constructed expression denotes E (up to isomorphism). The term $0_{A_{\mathrm{E}}}$ is added in case the alphabet of E contains actions that do not arise as the label of any event.

It is interesting to observe that the relational renaming operator is not needed in this proof; functional renaming would suffice. The proof above holds for the syntax of CSP—as in [3]—as well. The same cannot be done in CCS [12], because there only handshaking communication is available.

Now we pass to the case of finite bundle event structures.

Theorem 2. *For every finite bundle event structure* E *there is a closed recursion-free CCSP expression* P *such that* $[\![P]\!] \cong \mathrm{E}$.

Proof. The proof goes along the same lines as the previous one, except that instead of causal links $d <_{\mathrm{E}} e$ we now have to encode bundles $X \mapsto_{\mathrm{E}} e$. Let $X = \{d_1, ..., d_h\}$. Then the bundle $X \mapsto_{\mathrm{E}} e$ is represented by $R(de)$ where d is a fresh action and R is the relational renaming $\{(d, d_1), (d, d_2), ..., (d, d_h)(e, e)\}$.

One may wonder to what extent relational renaming is really needed here. For the language CCSP as given here it is, because with a straightforward structural induction one can check that all bundle event structures that can be denoted by recursion-free CCSP expressions with merely functional renaming only have bundles $X \mapsto e$ in which all events in X have the same label. However, there are other process algebraic operators that can take over the rôle of relational renaming.

Theorem 3. *For every finite bundle event structure* E *there is a closed recursion-free expression* P *in the language from* WINSKEL [18] *such that* $[\![P]\!] \cong \mathrm{E}$.

Proof. Winskel's language does not have relational renaming, but only functional renaming and a restriction operator $\lceil$. The restriction $\mathrm{E}\lceil A$ behaves like E but with its events restricted to those with labels which lie in the set A. The parallel composition $\times$ in Winskel's language allows every pair of events to synchronise; if e_0 is labelled a_0 and e_1 is labelled a_1 the synchronisation event is then labelled (a_0, a_1). Events need not synchronise however; an event e_0 in the first component that does not synchronise with any event of the second will be labelled by $(a_0, *)$, where a_0 is the label of e_0. For the rest Winskel's language is the same as CCSP, but untyped. The parallel operator of CCSP can be defined in terms of the operators $\times$, functional renaming and restriction. Although in a setting without recursion it is not possible to define CCSP's relational renaming operation in terms of the operations of Winskel's language, we can, for any finite bundle event structure E in which all actions have a different label and any image finite relational renaming R, define a context $C_{R,\mathrm{E}}[.]$ in Winskel's language that behaves like R. In the definition of this context, we use as a derived construct the *interleaving* operator $|||$ that is given by

$$P|||Q = f((P \times Q)\lceil\{(a, *) \mid a \in \alpha(P)\} \cup \{(*, b) \mid b \in \alpha(Q)\}),$$

where f is a functional renaming that renames each action $(a, *)$ into a, and each action $(*, b)$ into b. Let E and R be as stipulated. Now the context $C_{R,\mathrm{E}}[.]$ can be defined as

$$C_{R,\mathrm{E}}[.] = g(([.] \times (a_1||| \cdots |||a_n))\lceil R'),$$

where $\{a_b \mid \exists e \in E_\mathrm{E} \wedge l_\mathrm{E}(e) = a \wedge (a, b) \in R\} = \{a_1, \ldots a_n\}$, $R' = \{(a, a_b) \mid (a, b) \in R\}$ and g is the functional renaming that renames each action (a, a_b) into b. We claim that $[\![R(\mathrm{E})]\!] = [\![C_{R,\mathrm{E}}[\mathrm{E}]]\!]$ for each bundle event structure E in which all actions have a different label and any image finite relational renaming R. Using $C_{R,de}(de)$ instead of $R(de)$ in the proof of Theorem 2 now yields the required result.

In the presence of sequential composition, such as the operator ; in CSP, a bundle $\{d_1, ..., d_h\} \mapsto_\mathrm{E} e$ can also be represented as $(d_1 + \cdots + d_h); e$. However, a semantics of ; requires the introduction of a special event-label $\surd$, or some other additional structure, that helps to distinguish deadlock from successful termination. Arbitrary bundle event structures with this additional structure can in general not be represented by CSP expressions, at least not with the method employed here. Nevertheless, the construction above would work when taking ; to be a *sequencing* operator [5] that starts its second argument as soon as its first argument can perform no further actions.

Next we turn to infinite bundle event structures. Obviously *any* bundle event structure can be denoted, up to isomorphism, in a variant of CCSP with a suitable infinitary parallel composition $\|_{i\in I} P_i$. If we stick to the binary versions of $\|$ and $+$ it is straightforward to check that only countable event structures can be denoted (event structures with countably many events and only countably many bundles), even in the presence of arbitrary large recursive specifications. Thus, the best we can hope for is that every countable bundle event structure can be denoted by a CCSP expression. We are not sure if this is true; however, it can be established for prime event structures with binary conflict.

Theorem 4. *For every countable prime event structure with binary conflict* E *there is a closed CCSP expression* P *such that* $[\![P]\!] \cong \mathrm{E}$.

Proof. Although only the binary parallel composition exists in the syntax, a countable parallel composition $P_0 \| P_1 \| \cdots$ can be created with infinite unguarded recursion, namely as $\langle X^0_{A_0} | S \rangle$ where S contains the equations $X^i_{A_i} = P_i \| X^{i+1}_{A_{i+1}}$ where $A_i = \bigcup_{j \geq i} \alpha(P_j)$, for $i \in \mathbb{N}$. However, the denotational interpretation of such a system of equations contains only events whose existence can be proved by unwinding the recursion a finite number of times. There are for instance no events in $[\![\langle X_A | (X_A = a \| X_A) \rangle]\!]$. Thus, for the generated parallel composition to be useful, we need to require that for each $a \in A_0$ there is an i with $a \notin A_i$, i.e. $\bigcap_{i \in \mathbb{N}} A_i = \emptyset$. (*)

Now let $E_{\mathrm{E}} = \{a_i \mid i \in \mathbb{N}\}$, where the numbering is chosen in such a way that $a_i <_{\mathrm{E}} a_j \Rightarrow i < j$. As $<_{\mathrm{E}}$ is a partial order in which $\{d \in E \mid d <_{\mathrm{E}} e\}$ is finite for all $e \in E_{\mathrm{E}}$, this is always possible. Then the (possibly infinite) parallel composition $\|\{a_j \mid a_i <_{\mathrm{E}} a_j\}$ contains all events that have a_i as a causal predecessor, executed in parallel. Hence $[\![a_i(\|\{a_j \mid a_i <_{\mathrm{E}} a_j\})]\!]$ is the fragment of the desired event structure that contains all causal links starting in a_i. Its alphabet is contained in $\{a_j \mid j \geq i\}$. The parallel composition of all such event structures for $i \in \mathbb{N}$ therefore contains all causal links of E, and satisfies (*). This structure is to be put in parallel with one containing all conflicts, constructed in a similar way. As $\#_{\mathrm{E}}$ is irreflexive and symmetric, we only need to implement the conflicts $a_i \#_{\mathrm{E}} a_j$ with $i < j$. We find that E is denoted by

$$P = l_{\mathrm{E}} \left[\|_{i=0}^{\infty} (a_i(\|\{a_j \mid a_i <_{\mathrm{E}} a_j\}) \| \ \|_{i=0}^{\infty} (a_i + \|\{a_j \mid j > i \wedge a_i \#_{\mathrm{E}} a_j\}) \right] + 0_{A_E}$$

We leave it as an open problem whether the same can be achieved using only finite recursive specifications.

Due to the presence of uncountably many renaming operators, the signature of CCSP is undecidable. This can be changed by only allowing recursive enumerable renaming operators, i.e. operators $R \subseteq Act \times Act$ for which there exists a Turing machine enumerating all pairs $(a, b) \in R$. Such renaming operators can be represented by the source code describing the generating Turing machine. Codes are finite objects, and it is decidable whether a piece of text is the source code describing such a Turing machine. Now define a recursive enumerable version of CCSP, call it $\mathrm{CCSP}^{\mathrm{r.e.}}$, by requiring

- that Act is a r.e. set and all renaming operators are r.e.,
- that only r.e. subsets of Act are allowed as indices of 0 and the variables.
- and that recursive specifications S, seen as functions from V_S to the CCSP expressions, should be *primitive recursive*, with V_S a primitive decidable set.

This makes the signature of the language decidable. The primitive recursive requirement on S even makes it decidable whether a variable in a $\mathrm{CCSP}^{\mathrm{r.e.}}$ expression is free [4]. Now we have the following recursive enumerable version of Theorem 4:

Theorem 5. *Let* E *be a prime event structure with binary conflict such that* E_{E}*,* $\#_{\mathrm{E}}$*,* $<_{\mathrm{E}}$*,* A_{E} *are recursive enumerable sets,* l_{E} *is a recursive function, and there is an algorithm that for every event returns the finite set of its causal predecessors. Then there is a closed* $\mathrm{CCSP}^{\mathrm{r.e.}}$ *expression* P *such that* $[\![P]\!] \cong \mathrm{E}$.

4 Denoting stable event structures with binary conflict in CCSP

In this section we infer from Theorem 4 that up to hereditary history preserving bisimulation equivalence any countable stable event structure with binary conflict can be denoted by a closed CCSP expression.

Definition 8. Two stable event structures E and F are *history preserving bisimulation equivalent* ($\mathrm{E} \leftrightarroweq_h \mathrm{F}$) iff $A_\mathrm{E} = A_\mathrm{F}$ and there exists a relation $R \subseteq C(\mathrm{E}) \times C(\mathrm{F}) \times \mathcal{P}(E_\mathrm{E} \times E_\mathrm{F})$—called a *history preserving bisimulation*—such that $(\emptyset, \emptyset, \emptyset) \in R$ and whenever $(x, y, f) \in R$ then

- f is an isomorphism between $(x, <_x, l_\mathrm{E} \restriction x)$ and $(y, <_y, l_\mathrm{F} \restriction y)$,
- $x \subseteq x' \in C(\mathrm{E}) \Rightarrow \exists y', f'$ with $y \subseteq y' \in C(\mathrm{F})$, $(x', y', f') \in R$ and $f' \restriction x = f$,
- $y \subseteq y' \in C(\mathrm{F}) \Rightarrow \exists x', f'$ with $x \subseteq x' \in C(\mathrm{E})$, $(x', y', f') \in R$ and $f' \restriction x = f$.

The bisimulation and the equivalence are *hereditary* ($\mathrm{E} \leftrightarroweq_{hh} \mathrm{F}$) if moreover

- $x \supseteq x' \in C(\mathrm{E}) \Rightarrow \exists y', f'$ with $y \supseteq y' \in C(\mathrm{F})$, $(x', y', f') \in R$ and $f' = f \restriction x'$,
- $y \supseteq y' \in C(\mathrm{F}) \Rightarrow \exists x', f'$ with $x \supseteq x' \in C(\mathrm{E})$, $(x', y', f') \in R$ and $f' = f \restriction x'$.

R is *functional* if $R = \{(x, f(x), f \restriction x) \mid x \in C(\mathrm{E})\}$ for a function $f\colon E_\mathrm{E} \rightarrow E_\mathrm{F}$.

Note that a functional bisimulation is always hereditary. Moreover, when checking that a function $f : E_\mathrm{E} \rightarrow E_\mathrm{F}$ induces an history preserving bisimulation, the second requirement is trivially fulfilled. JOYAL, NIELSEN & WINSKEL [10] characterised a functional history preserving bisimulation as a categorical construction called *open map*.

Definition 8 also applies when E and F are prime or bundle event structures, or when one of them is prime and the other is stable. We now show that every (countable) stable event structure with binary conflict is hereditary history preserving bisimulation equivalent with a (countable) prime event structure with binary conflict.

Definition 9. *Given a stable event structure* $\mathrm{E} = (E, \#, \vdash, A, l)$ *with binary conflict, the associated prime event structure* $\mathrm{E}' = (E', <, \#', A, l')$ *is given by*

- $E' = \{e_x \mid e \in x \in C(\mathrm{E})$ *and x is a minimal configuration containing e*$\}$,
- $d_x < e_y$ *iff* $x \subset y$,
- $d_x \#' e_y$ *iff* E *has no configuration containing both x and y,*
- $l'(e_X) = l(e)$.

E' is obviously a prime event structure with binary conflict, and if E is countable, then so is E'. Moreover, it is not too hard to check that the function $f : E' \rightarrow E$ given by $f(e_X) = e$ for $e \in E'$ induces a history preserving bisimulation. Therefore, any (countable) stable event structure with binary conflict is hereditary history preserving bisimulation equivalent with a (countable) prime event structure with binary conflict. This result also follows from the category theoretic results in [10]. In view of this, Theorem 4 implies

Theorem 6. *For every countable stable event structure with binary conflict* E *there is a closed CCSP expression* P *such that* $[\![P]\!] \leftrightarroweq_{hh} \mathrm{E}$.

5 Arbitrary conflict reduces to binary conflict

In [17] event structures of the form $(E, Con, \vdash, A, l)$ appear, in which the predicate Con of Definition 4 is explicitly given rather then generated by a binary conflict relation. It is postulated that Con is a downwards closed nonempty set of finite sets of events. The configurations of such event structures and the causality relations on them are determined exactly as in Definition 5. Note that Con can equivalently be represented by its complement: an upwards closed set CONFL of finite nonempty sets of events. Another equivalent representation is in terms of the minimal members of CONFL: a collection $\#$ of finite nonempty sets of events, such that there are no two different $\gamma, \gamma' \in \#$ with $\gamma \subset \gamma'$. Now a finite set x is *consistent* or *conflict-free* if $\gamma \subseteq x$ for no $\gamma \in \#$. In this representation event structures with a binary conflict relation are literally a special case of the ones with arbitrary conflict relations. Statement $\gamma \in \#$ means that the events in γ cannot *all* happen in the same run. It does not place a restriction on proper subsets of γ.

In this section we show that every (countable) stable event structure is history preserving bisimulation equivalent to a (countable) prime event structure with binary conflict. For finite prime event structures this theorem was claimed by us in [8]. The generalisation to infinite event structures was reported in [6]. The same theorem has been discovered independently by NIELSEN & WINSKEL [14], where the first published proof can be found. Although our proof is based on the same idea as the one of [14], it is somewhat shorter and more constructive.

Definition 10. Let E be a countable event structure with arbitrary conflict. For $e \in E_{\mathrm{E}}$ let $\#_e$ be the set of conflicts involving e: $\#_e = \{\gamma \in \#_{\mathrm{E}} \mid e \in \gamma\}$. Define the event structure $2(\mathrm{E})$ by

- $E_{2(\mathrm{E})} = \{(e,t) \mid e \in E_{\mathrm{E}},\ t\colon \#_e \to \mathbb{N}$ with t recursive and $\forall \gamma \in \#_e : t(\gamma) < |\gamma| - 1\}$
- $A_{2(\mathrm{E})} = A_{\mathrm{E}}$ and $l_{2(\mathrm{E})}(e,t) = l_{\mathrm{E}}(e)$
- $\#_{2(\mathrm{E})} = \{((e,t),(e',t')) \mid (e = e' \wedge t \neq t') \vee (e \neq e' \wedge \exists \gamma \in \#_e : t(\gamma) = t'(\gamma))\}$
- $\mapsto_{2(\mathrm{E})} = \{(2(X),(e,t)) \mid X \mapsto_{\mathrm{E}} e\}$ in which $2(X) = \{(e,t) \mid e \in X\}$
- $\vdash_{2(\mathrm{E})} = \{(X,(e,t)) \mid \pi_1(X) \vdash e\}$ in which $\pi_1(X) = \{e \in E_{\mathrm{E}} \mid \exists t : (e,t) \in X\}$.

The idea behind this definition is the following: every member γ of the conflict relation on E has $|\gamma|$ elements, of which only $|\gamma| - 1$ can be executed. This can be modelled as an allocation of $|\gamma| - 1$ seats to $|\gamma|$ events. Let us number these seats from 0 to $|\gamma| - 2$. The event that is last in grabbing a seat can not happen. In general an event can occur in many elements γ of $\#_{\mathrm{E}}$, namely the ones in $\#_e$. In order to happen it has to grab a seat for each of these γ's. Now $2(\mathrm{E})$ is an event structure where this abstract notion of conflict has been implemented on a more down-to-earth level. The new events are allocations of old events to seats. To be precise, they are pairs (e,t), where e is the name of the source event and t a function that for every competition $\gamma \in \#_e$ in which e participates selects a seat $t(\gamma) < |\gamma| - 1$. In [14] a pair (e,t) is called an event with a *twist*; hence the choice of the letter t. Now the new events, which are old events allocated to seats, inherit their labelling and their causal dependencies from their source events. The causal dependencies are implemented by $\mapsto_{2\mathrm{E}}$ or $\vdash_{2\mathrm{E}}$, depending on whether the original event structure was a bundle, or a general one. Compare these definitions with the relational renaming operator in Section 2. The conflict relation on $2(\mathrm{E})$ is binary. The first set of conflicts ensures that an event can occur with only one allocation to seats in the various conflicts. The second set, that no two events are assigned the same seat in any particular conflict. This implements the abstract notion of conflict in E.

When an event e' occurs is does not really matter which seats it chooses in the various conflicts it participates in, as long as these seats are not yet taken by other events. For each event e that happened already, the chosen seats are given by the function t it happened with. Now e' has to choose an allocation function t' that is different from t in each conflict that involves both e' and e. In order to make such a choice in a computationally respectful way, we assume that all allocation functions of events that happened previously are recursive. When e is about to happen, it can then calculate which seats are still free and choose a function that is recursive as well. (A function $t : \#_e \rightarrow \mathbb{N}$ is *recursive* if there is a partial recursive function $t' : \mathcal{P}_{\mathit{fin}}(E_{\mathrm{E}}) \rightarrow \mathbb{N}$ with $t = t' \upharpoonright \#_e$ a total function. There is no need to assume that $\#_e$ is a decidable set.) The resulting requirement in Definition 10 that all functions t should be recursive, ensures that $2(\mathrm{E})$ is still countable. Without the recursiveness requirement this would not be the case.

Theorem 7. *Let* E *be a countable stable event structure. Then* $2(\mathrm{E})$ *is a countable stable event structure with binary conflict and the function* $f : 2(\mathrm{E}) \rightarrow \mathrm{E}$ *given by* $f(e,t) = e$ *induces a history preserving bisimulation. Hence* $2(\mathrm{E}) \leftrightarroweq_{hh} \mathrm{E}$.

Proof. As $(e,t)\#_{2(\mathrm{E})}(e,t')$ for $t \neq t'$, it follows immediately that $f \upharpoonright x$ is injective for every configuration x. Now suppose $f(x)$ contains a conflict $\gamma \in \#_{\mathrm{E}}$. Then in x there must be events $\{(e_i,t_i) \mid e_i \in \gamma\}$ with $t(e_i) < |\gamma| - 1$. Hence two of these events must be in conflict, contradicting that x is a configuration. If follows that $f(x)$ is conflict-free. It is immediate from the definition of $\mapsto_{2(\mathrm{E})}$ resp. $\vdash_{2\mathrm{E}}$ that if $(e_1,t_1),...,(e_n,t_n)$ secures x in $2(\mathrm{E})$ then $e_1,...,e_n$ secures $f(x)$. Hence x is a configuration of E. It is also immediate from the definition of $\mapsto_{2(\mathrm{E})}$ resp. $\vdash_{2\mathrm{E}}$ that f preserves $<_x$ and labelling.

Now suppose $x \in C(2(\mathrm{E}))$ and $f(x) \subseteq y' \in C(\mathrm{E})$. We need to show that there is an $x' \in C(2(\mathrm{E}))$ with $x \subseteq x'$ and $f(x') = y'$. By induction on $|y'|$ it suffices to restrict attention to the case that there is exactly one event in $y' - f(x)$, call it e. As y' is conflict-free, for every $\gamma \in \#_e$ we have that $|\gamma \cap f(x)| \leq |\gamma| - 2$. Hence there exists a recursive $t : \#_e \rightarrow \mathbb{N}$ satisfying, for all $\gamma \in \#_e$, $t(\gamma) < |\gamma| - 1$ and for no $(e',t') \in x$: $t'(\gamma) = t(\gamma)$. It follows that $x' \stackrel{\mathrm{def}}{=} x \cup \{(e,t)\}$ is conflict-free. Moreover, any securing $(e_1,t_1),...,(e_n,t_n)$ of x can be extended with (e,t) into a securing of x'. This follows because $e_1,...,e_n,e$ is a securing of y', using the definition of $\mapsto_{2(\mathrm{E})}$ resp. $\vdash_{2\mathrm{E}}$. Thus $x' \in C(2(\mathrm{E}))$, which had to be proved. The other requirement for f inducing a history-preserving bisimulation is trivial.

By combining this insight with Theorem 6 it follows immediately that up to hereditary history preserving bisimulation equivalence all countable stable event structures with arbitrary conflicts are expressible in CCSP.

Acknowledgement Thanks to Rom Langerak for valuable feedback.

References

1. M. BEDNARCZYK (1991): *Hereditary history preserving bisimulation, or what is the power of the future perfect in program logics.* Technical report, Polish Academy of Sciences, Gdańsk. Available at `ftp://ftp.ipipan.gda.pl/marek/historie.ps.gz`.
2. G. BOUDOL & I. CASTELLANI (1994): *Flow models of distributed computations: Three equivalent semantics for CCS.* Information and Computation 114(2), pp. 247–314.
3. S.D. BROOKES, C.A.R. HOARE & A.W. ROSCOE (1984): *A theory of communicating sequential processes.* Journal of the ACM 31(3), pp. 560–599.
4. R.J. VAN GLABBEEK (1994): *On the expressiveness of ACP (extended abstract).* In A. Ponse, C. Verhoef & S.F.M. van Vlijmen, editors: Proceedings First Workshop on the Algebra of Communicating Processes, ACP94, Utrecht, Workshops in Computing, Springer, pp. 188–217. Available at `http://boole.stanford.edu/pub/acp.ps.gz`.
5. R.J. VAN GLABBEEK & U. GOLTZ (2001): *Refinement of actions and equivalence notions for concurrent systems.* Acta Informatica 37, pp. 229–327.
6. R.J. VAN GLABBEEK & G.D. PLOTKIN (1995): *Configuration structures (extended abstract).* In D. Kozen, editor: Proceedings 10^{th} Annual IEEE Symposium on Logic in Computer Science (LICS95), San Diego, USA, IEEE Computer Society Press, pp. 199–209.
7. R.J. VAN GLABBEEK & F.W. VAANDRAGER (1987): *Petri net models for algebraic theories of concurrency (extended abstract).* In J.W. de Bakker, A.J. Nijman & P.C. Treleaven, editors: Proceedings PARLE, Parallel Architectures and Languages Europe, Eindhoven, The Netherlands, June 1987, Vol. II: Parallel Languages, LNCS 259, Springer, pp. 224–242.
8. R.J. VAN GLABBEEK & F.W. VAANDRAGER (1991): *The difference between splitting in n and $n+1$ (abstract).* In E. Best & G. Rozenberg, eds.: Proc. 3^{rd} Workshop on Concurrency and Compositionality, Goslar, GMD-Studien Nr. 191, Universität Hildesheim, pp. 117–121. Full version in Information and Computation 136(2), 1997, pp. 109–142.
9. C.A.R. HOARE (1985): *Communicating Sequential Processes.* Prentice Hall.
10. A. JOYAL, M. NIELSEN & G. WINSKEL (1996): *Bisimulation from open maps.* Information and Computation 127(2), pp. 164–185.
11. R. LANGERAK (1992): *Transformations and Semantics for LOTOS.* PhD thesis, Department of Computer Science, University of Twente.
12. R. MILNER (1989): *Communication and Concurrency.* Prentice Hall, Englewood Cliffs.
13. M. NIELSEN, G.D. PLOTKIN & G. WINSKEL (1981): *Petri nets, event structures and domains, part I.* Theoretical Computer Science 13(1), pp. 85–108.
14. M. NIELSEN & G. WINSKEL (1996): *Petri nets and bisimulation.* Theoretical Computer Science 153, pp. 211–244.
15. A. RABINOVICH & B.A. TRAKHTENBROT (1988): *Behavior structures and nets.* Fundamenta Informaticae 11(4), pp. 357–404.
16. F.W. VAANDRAGER (1993): *Expressiveness results for process algebras.* In J.W. de Bakker, W.P. de Roever & G. Rozenberg, eds.: Proc. REX Workshop on Semantics: Foundations and Applications, Beekbergen, The Netherlands, June 1992, LNCS 666, Springer, pp. 609–638.
17. G. WINSKEL (1987): *Event structures.* In W. Brauer, W. Reisig & G. Rozenberg, editors: Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II; Proceedings of an Advanced Course, Bad Honnef, September 1986, LNCS 255, Springer, pp. 325–392.
18. G. WINSKEL (1989): *An introduction to event structures.* In J.W. de Bakker, W.P. de Roever & G. Rozenberg, editors: REX School and Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Noordwijkerhout, The Netherlands, May/June 1988, LNCS 354, Springer, pp. 364–397.
19. G. WINSKEL & M. NIELSEN (1995): *Models for concurrency.* In S. Abramsky, D. Gabbay & T. Maibaum, editors: Handbook of Logic in Computer Science, Oxford University Press, pp. 1–148.