HILDESHEIMER INFORMATIK-BERICHTE

ISSN 0941-3014

On Syntactic and Semantic Action Refinement

Ursula Goltz
Roberto Gorrieri
Arend Rensink

17/92 (Dezember 1992)

UNIVERSITÄT HILDESHEIM

Dieser Bericht ist herausgegeben vom

Institut für Informatik

Postfach 10 13 63
31113 Hildesheim

Shortened version (without proofs) published as: U. Goltz, R. Gorrieri, and A. Rensink. On syntactic and semantic action refinement. In Hagiya and Mitchell, editors, *Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 385–404. Springer-Verlag, April 1994.

On Syntactic and Semantic Action Refinement[a]

Ursula Goltz
University of Hildesheim
goltz@informatik.uni-hildesheim.de

Roberto Gorrieri
University of Bologna
gorrieri@cs.unibo.it

Arend Rensink
University of Twente[b]
rensink@cs.utwente.nl

Abstract

The semantic definition of action refinement on labelled event structures is compared with the notion of *syntactic substitution*, which can be used as another notion of action refinement in a process algebraic setting. This is done by studying a process algebra equipped with the ACP sequential composition, parallel composition with an explicit synchronization set, and an operator for action refinement. On the one hand, the language (including the refinement operator) is given a flow event structure semantics. On the other hand, a reduction procedure transforms a process term P into a *flat* term (i.e., with the refinement operator not occurring in it) $red(P)$ by means of syntactic substitution, defined in a structural inductive way.

The main aim of the paper is to find general conditions under which the terms P and $red(P)$ have the same semantics. The results we present are essentially dependent on the question whether the refined action can be synchronized or not. In the latter case, P and $red(P)$ give rise to isomorphic flow event structures under mild assumptions. The former case is considerably more difficult. We give necessary and sufficient *semantic* conditions under which refinement can be distributed over synchronization up to isomorphic domains of configurations. Subsequently we also give sufficient (but not necessary) *syntactic* conditions for reducible terms.

1 Introduction

The refinement of actions in concurrency theories has been proposed as a means for relating descriptions of concurrent systems at different levels of abstraction and for helping in their top-down design. The basic principle is to implement a given abstract action in terms of larger and more complex concrete behaviour. In this paper it is expressed by terms of the form $P[a \rightsquigarrow Q]$ where, intuitively, every time action a should be executed in P, the term Q is executed instead. This conceptually attractive principle has received widespread interest; however, to formalize it effectively is proving to be a complex issue, and consequently research on this subject has taken various different approaches.

Two lines of research can be recognized. On the one hand there is *atomic refinement* [4, 11, 12], where one takes the point of view that actions are atomic and their refinements should in some sense preserve this atomicity. On the other hand there is a more liberal notion of refinement according to which atomicity is always relative to the current level of abstraction, and may in a sense be destroyed by refinement. This paper is concerned with the second approach.

[a] This work has been partially supported by Esprit Basic Research Working Group 6067 CALIBAN (Causal Calculi Based on Nets).

[b] On leave at the University of Hildesheim while doing this research.

Within this approach there are again essentially two notions of action refinement, which we call *semantic* and *syntactic*. In the *semantic* interpretation, a refinement operation is defined in the semantic domain that is used to interpret terms. Then the semantics of $P[a \rightsquigarrow Q]$ can be defined using this operator. For example, when using event structures as semantic domains, an event structure $\mathcal{F} = [\![Q]\!]$, representing the semantics of Q, would be substituted for every a-labelled event e in the event structure $\mathcal{E} = [\![P]\!]$. The refinement operation preserves the semantic embedding of events: e.g., if e is in conflict with an event d, then all the events of $\mathcal{F}$ will be in conflict with d. Investigations of such refinement operators can be found in [3, 6, 8, 10, 14, 17, 19, 21, 20] over the semantic domains of Prime, Free and Flow Event Structures, Causal Trees and Petri Nets.

The *syntactic* approach takes a different starting point, namely a process algebra equipped with an ACP-like operation of sequential composition. Action refinement is understood as an operation of syntactic substitution of a process term for an action. Hence $P[a \rightsquigarrow Q]$ is interpreted as the term obtained from P if action a is replaced by Q; i.e., Q is to be substituted for a in the *term* P rather than in the semantics of P. Therefore, the semantics of $P[a \rightsquigarrow Q]$ is, *by definition*, the semantics of the term $P\left\{{}^{Q}/_{a}\right\}$. This line of research has been pursued in [1, 2, 15].

Simple examples (see below) show that the two approaches do not coincide in general. This is essentially due to the interplay between refinement and synchronization. In this paper, we compare the two approaches with the aim to identify under which restrictions they yield the same result. This is interesting for two reasons. Firstly, it helps to understand how the two approaches deal with the aforementioned interplay between refinement and synchronization. Secondly, it is interesting for applications of action refinement to know when semantic refinement can be implemented by the simpler syntactic substitution.

We consider a process algebra with sequential composition and synchronization. We provide the language with a flow event structure semantics, as defined in [19], and address the problem of finding necessary and sufficient conditions under which syntactic substitution agrees with the semantic operation of action refinement. That is, we investigate conditions under which the following diagram commutes:

$$\begin{array}{ccc} P[a \rightsquigarrow Q] & \xrightarrow[\text{syntactic ref.}]{} & P\left\{{}^{Q}/_{a}\right\} \quad \textit{syntax} \\ \Big\downarrow & & \Big\downarrow \\ [\![P]\!][a \rightsquigarrow [\![Q]\!]] & \xrightarrow[\text{semantic ref.}]{} & [\![P\left\{{}^{Q}/_{a}\right\}]\!] \quad \textit{semantics} \end{array} \tag{1}$$

It turns out that the problem can be reduced to the following question: under which conditions does refinement distribute over parallel composition with synchronization? In this paper we use a TCSP-like synchronization operator,[1] which takes the form $P_1 \parallel_A P_2$, where A denotes the set of *communication actions*, i.e. those actions on which both P_1 and P_2 are forced to synchronize. Distribution of refinement over parallel composition then means that the semantic equation

$$(P_1 \parallel_A P_2)[a \rightsquigarrow Q] \cong P_1[a \rightsquigarrow Q] \parallel_{A'} P_2[a \rightsquigarrow Q] \tag{2}$$

holds, where A' may be some modification of the synchronization set A, in case synchronizing actions are refined. This equation however does not hold in general. The terms $(a \parallel_{\{b\}} b; c)[a \rightsquigarrow b]$ and $(a[a \rightsquigarrow b] \parallel_{\{b\}} b; c[a \rightsquigarrow b])$ for instance are not equivalent: intuitively, in the first term, c on

[1] This choice does not affect the central problem essentially. In Section 7 we briefly discuss the CCS setting of [1, 2].

the right hand side is prevented from occurring since the preceding b cannot synchronize with anything on the left hand side; hence this behaviour may only execute a, which is however refined to b. In the second term, b occurs as a result of synchronization, after which c is executed.

In this example one could argue that the mismatch is due to the fact that on the right hand side, "new" actions (the b resulting from the refinement of a) are permitted to synchronize with "old" ones (the b already occurring before refinement). This is in contrast with the intuition that, in $P[a \rightsquigarrow Q]$, the actions of P and Q should be considered at different levels of abstraction (see also [4] on this point). We will adopt this view and restrict our attention to those terms satisfying the following alphabet-disjointness condition: $P[a \rightsquigarrow Q]$ is *well-formed* if $L(P) \cap L(Q) = \emptyset$, where $L(P)$ denotes the alphabet of P. We first consider the case that synchronizing actions are not refined, that is, $a \notin A$ for $(P_1 \parallel_A P_2)[a \rightsquigarrow Q]$. In this case we show that under well-formedness, (2) holds and we are therefore able to establish commutativity of (1). Non-well-formed terms may be dealt with at the price of adding an auxiliary operator of relabelling, as illustrated in Section 6.

The situation becomes much more difficult if we allow to refine synchronizing actions, that is, $a \in A$ for $(P_1 \parallel_A P_2)[a \rightsquigarrow Q]$. Then there are well-formed terms P_1, P_2, Q such that (2) fails to hold even for ordinary (interleaving) bisimulation. Additional, more restrictive conditions on terms have to be imposed. The second result of this paper is the formulation of necessary and sufficient *semantic* conditions and sufficient *syntactic* conditions for (2) to hold. We subsequently extend the latter conditions for terms of the form $(P_1 \parallel_A P_2)[a \rightsquigarrow Q]$ to a characterization of the sublanguage in which refinement may be replaced by substitution.

2 Syntax and semantics of the language

We assume a global (infinite) set of actions Act. The following grammar defines the terms of the language (a finite process algebra with action refinement) that we will study in this paper.

$$P ::= a \mid P + P \mid P; P \mid P \parallel_A P \mid P[a \rightsquigarrow P]$$

Most of the operators are standard. We use a *family* of synchronization operators $\{\parallel_A\}_{A \subseteq Act}$ corresponding to the TCSP approach. The *refinement operator* $P[a \rightsquigarrow Q]$ acts on single actions at a time. The behaviour of $P[a \rightsquigarrow Q]$ is derived from the behaviour of P by replacing every execution of the action a by the behaviour of Q. Σ will denote the set of all the terms generated by the syntax above; $\Sigma_{flat} \subseteq \Sigma$ denotes the set of terms that do not contain refinement operators. Brackets will be used as usual to show the structure of terms in Σ; to improve the readability, we will let sequential composition bind stronger than choice and synchronization, and refinement stronger than any of the binary operators.

In the conclusions we briefly discuss how the results of this report can be extended to infinite behaviour, specified using systems of equations.

2.1 Well-formed terms

A useful notion in this investigation is the *alphabet* of a term P, denoted $L(P)$. Another, less standard notion is the *set of synchronizing actions* of a term P, denoted $S(P)$. These are defined inductively in Table 1. It follows that $S(P) \subseteq L(P)$ for all terms $P \in \Sigma$.

We now argue that it makes sense to restrict the refinements under consideration to a certain format. Consider a term of the form $P[a \rightsquigarrow Q]$. The intuition behind refinement tells us that Q represents an implementation of a and hence a is in some sense *more abstract* than the actions in $L(Q)$. It is only a small step from there to the assumption that *all* the actions of P are more abstract than those of Q; in other words, $L(Q)$ contains "new" actions that did not yet occur

$$
\begin{array}{rcl}
L(a) & := & \{a\} \\
L(P+Q) & := & L(P) \cup L(Q) \\
L(P;Q) & := & L(P) \cup L(Q) \\
L(P \parallel_A Q) & := & L(P) \cup L(Q) \cup A \\
L(P[a \rightsquigarrow Q]) & := & \begin{cases} (L(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in L(P) \\ L(P) & \text{otherwise} \end{cases} \\
\hline
S(a) & := & \emptyset \\
S(P+Q) & := & S(P) \cup S(Q) \\
S(P;Q) & := & S(P) \cup S(Q) \\
S(P \parallel_A Q) & := & S(P) \cup S(Q) \cup ((L(P) \cup L(Q)) \cap A) \\
S(P[a \rightsquigarrow Q]) & := & \begin{cases} (S(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in S(P) \\ S(P) \cup S(Q) & \text{if } a \in L(P) \setminus S(P) \\ S(P) & \text{otherwise.} \end{cases}
\end{array}
$$

Table 1: Label set and synchronizing set

$$
\frac{}{\vdash a} \qquad \frac{\vdash P, Q \quad * \in \{;, +, \parallel_A\}}{\vdash P * Q} \qquad \frac{\vdash P, Q \quad L(P) \cap L(Q) = \emptyset}{\vdash P[a \rightsquigarrow Q]}
$$

Table 2: The well-formedness predicate

in the specification P. This makes it impossible for actions in P to synchronize with those in Q (after refinement) and hence rules out a kind of confusion of abstraction levels. In other words we assume

$$L(P) \cap L(Q) = \emptyset \tag{3}$$

To put this assumption into effect we will restrict ourselves to a subset of the terms satisfying the *well-formedness predicate* $\vdash$ defined in Table 2, which effectively ensures (3). If this is felt to be an undue restriction, then —at the price of adding an (auxiliary) operator of *renaming* to the syntax— this assumption can be dropped and our results can be generalized to the entire Σ, as we will show in Section 6.

2.2 Flow event structure semantics

We interpret the terms of Σ in the model of *flow event structures* proposed by Boudol and Castellani [5]. The interpretation is standard and can be found for instance in [19].

2.1 Definition. A *flow event structure* is a tuple $\mathcal{E} = \langle E, \prec, \#, \ell \rangle$ where

- E is a set of *events*;
- $\prec \subseteq E \times E$ is an irreflexive *flow relation*;
- $\# \subseteq E \times E$ is a symmetric *conflict relation*;
- $\ell \colon E \to Act$ is a *labelling function.*

The components of a flow event structure $\mathcal{E}$ are denoted $E_\mathcal{E}$, $\prec_\mathcal{E}$ etc. The class of flow event structures will be denoted $\mathbf{E}$ and ranged over by $\mathcal{E}, \mathcal{F}$. The operational intuition behind flow event structures is given by the *configurations* that it may execute, as follows:

2.2 Definition. Let $\mathcal{E}$ be a flow event structure and $F \subseteq E_{\mathcal{E}}$ a subset of the events of $\mathcal{E}$.
F is a *configuration of* $\mathcal{E}$ if it satisfies the following conditions:

- F does not contain flow-cycles, i.e. $(\prec_{\mathcal{E}} \cap (F \times F))^+$ (the transitive closure of $\prec_{\mathcal{E}}$) is irreflexive;
- F is conflict-free, i.e. $\neg\exists d, e \in F.\, d \#_{\mathcal{E}} e$;
- F is closed under non-conflicting causes: $\forall d \in (E_{\mathcal{E}} \setminus F).\, \forall e \in F.\, d \prec_{\mathcal{E}} e \implies \exists d' \in F.\, d \#_{\mathcal{E}} d' \prec_{\mathcal{E}} e$.

Two event structures $\mathcal{E}, \mathcal{F}$ are *isomorphic*, denoted $\mathcal{E} \cong \mathcal{F}$, if there exists a bijection $f\colon E_{\mathcal{E}} \to E_{\mathcal{F}}$ such that for all $d, e \in E_{\mathcal{E}}$ the following hold:

$$\begin{array}{rcl} d \prec_{\mathcal{E}} e & \iff & f(d) \prec_{\mathcal{F}} f(e) \\ d \#_{\mathcal{E}} e & \iff & f(d) \#_{\mathcal{F}} f(e) \\ \ell_{\mathcal{E}}(e) & = & \ell_{\mathcal{F}}(f(e)) \ . \end{array}$$

A number of operations over $\mathbf{E}$, corresponding to the syntactic ones, are defined. We will use the fact (the proof of which is straightforward) that isomorphism is a congruence with respect to all these operators. In this paper, we overload the notation for the syntactic operators (in Σ) and their semantic counterparts (to be defined below over $\mathbf{E}$); the context will clarify which of the two we mean.

2.3 Definition. Let $\mathcal{E}, \mathcal{F} \in \mathbf{E}$ be such that $E_{\mathcal{E}} \cap E_{\mathcal{F}} = \emptyset$.
The *choice between* $\mathcal{E}$ *and* $\mathcal{F}$ is defined by

$$\mathcal{E} + \mathcal{F} := \langle E_{\mathcal{E}} \cup E_{\mathcal{F}}, \prec_{\mathcal{E}} \cup \prec_{\mathcal{F}}, \#_{\mathcal{E}} \cup \#_{\mathcal{F}} \cup (E_{\mathcal{E}} \times E_{\mathcal{F}}) \cup (E_{\mathcal{F}} \times E_{\mathcal{E}}), \ell_{\mathcal{E}} \cup \ell_{\mathcal{F}} \rangle \ .$$

2.4 Definition. Let $\mathcal{E}, \mathcal{F} \in \mathbf{E}$ be such that $E_{\mathcal{E}} \cap E_{\mathcal{F}} = \emptyset$.
The *sequential composition of* $\mathcal{E}$ *and* $\mathcal{F}$ is defined by

$$\mathcal{E}; \mathcal{F} := \langle E_{\mathcal{E}} \cup E_{\mathcal{F}}, \prec_{\mathcal{E}} \cup \prec_{\mathcal{F}} \cup (E_{\mathcal{E}} \times E_{\mathcal{F}}), \#_{\mathcal{E}} \cup \#_{\mathcal{F}}, \ell_{\mathcal{E}} \cup \ell_{\mathcal{F}} \rangle \ .$$

2.5 Definition. Let $\mathcal{E}, \mathcal{F} \in \mathbf{E}$ and $A \subseteq Act$; let $* \notin E_{\mathcal{E}} \cup E_{\mathcal{F}}$.
The *synchronization of* $\mathcal{E}$ *and* $\mathcal{F}$ *over* A is defined by $\mathcal{E} \|_A \mathcal{F} := \langle E, \prec, \#, \ell \rangle$, where:

- $E = (E_{\mathcal{E}} \times \{*\}) \cup (\{*\} \times E_{\mathcal{F}}) \cup \{\, (d, e) \in E_{\mathcal{E}} \times E_{\mathcal{F}} \mid \ell_{\mathcal{E}}(d) = \ell_{\mathcal{F}}(e) \in A \,\}$;
- $(d, e) \prec (d', e')$ iff $d \prec_{\mathcal{E}} d' \vee e \prec_{\mathcal{F}} e'$;
- $(d, e) \# (d', e')$ iff $\begin{array}[t]{l} d \#_{\mathcal{E}} d' \vee e \#_{\mathcal{F}} e' \\ \vee\, (d = d' \neq * \wedge e \neq e') \vee (e = e' \neq * \wedge d \neq d') \\ \vee\, (d = d' = * \wedge e = e' \wedge \ell_{\mathcal{F}}(e) \in A) \\ \vee\, (e = e' = * \wedge d = d' \wedge \ell_{\mathcal{E}}(d) \in A); \end{array}$
- $\ell(d, e) = \begin{cases} \ell_{\mathcal{E}}(d) & \text{if } e = *; \\ \ell_{\mathcal{F}}(e) & \text{otherwise.} \end{cases}$

2.6 Definition. Let $\mathcal{E}, \mathcal{F} \in \mathbf{E}$, $E_{\mathcal{F}} \neq \emptyset$ and $a \in Act$; let $* \notin E_{\mathcal{E}} \cup E_{\mathcal{F}}$.
The *refinement of* a *by* $\mathcal{F}$ *in* $\mathcal{E}$ is defined by $\mathcal{E}[a \rightsquigarrow \mathcal{F}] := \langle E, \prec, \#, \ell \rangle$, where

- $E = \{\, (d, e) \in E_{\mathcal{E}} \times (E_{\mathcal{F}} \cup \{*\}) \mid \ell_{\mathcal{E}}(d) = a \iff e \neq * \,\}$;
- $(d, e) \prec (d', e')$ iff $d \prec_{\mathcal{E}} d' \vee (d = d' \wedge e \prec_{\mathcal{F}} e')$;
- $(d, e) \# (d', e')$ iff $d \#_{\mathcal{E}} d' \vee (d = d' \wedge e \#_{\mathcal{F}} e')$;
- $\ell(d, e) = \begin{cases} \ell_{\mathcal{E}}(d) & \text{if } e = *; \\ \ell_{\mathcal{F}}(e) & \text{otherwise.} \end{cases}$

The semantics of our terms is given by a function $[\![\cdot]\!]\colon \Sigma \to \mathbf{E}$, defined inductively as follows:

$$\begin{aligned} [\![a]\!] &:= \langle \{e\}, \emptyset, \emptyset, (e,a)\rangle \\ [\![P+Q]\!] &:= [\![P]\!] + [\![Q]\!] \\ [\![P;Q]\!] &:= [\![P]\!]; [\![Q]\!] \\ [\![P \parallel_A Q]\!] &:= [\![P]\!] \parallel_A [\![Q]\!] \\ [\![P[a \rightsquigarrow Q]]\!] &:= [\![P]\!][a \rightsquigarrow [\![Q]\!]] \end{aligned}$$

This semantics then induces an equivalence relation $\cong_e$ over Σ:

$$P \cong_e Q :\Leftrightarrow [\![P]\!] \cong [\![Q]\!]$$

Because $\cong$ is a congruence over all the operators we have defined, $\cong_e$ is a congruence over Σ.

It should be noted that although this equivalence relation is defined in a straightforward manner and is fairly easy to prove, it is by no means the only reasonable equivalence one may consider over flow event structures. In fact for some purposes $\cong_e$ is too strong; it is for instance easy to see that $a \parallel_{\{a\}} a \not\cong_e a$ whereas in many cases these terms are considered equivalent. In Section 5 this sort of problem will cause us to consider isomorphism of the underlying *configuration structures* instead, which is the strictest relation that can be defined naturally on flow event structures and is less discriminating than event structure isomorphism.

2.3 Configuration structures

If one takes the set of all configurations of a given flow event structures, together with the labelling functions, this again forms a model of behaviour. We will use CS to denote the mapping from flow event structures to their sets of configurations:

$$CS\colon \mathcal{E} \mapsto \langle \{\, F \subseteq E_{\mathcal{E}} \mid F \text{ is a configuration of } \mathcal{E} \,\}, \ell_{\mathcal{E}} \rangle$$

These sets of configurations are called *families of configurations* or *configuration structures.*[2] Configuration structures form the standard underlying semantic model for event structures: cf. [22, 23]. They can be compared using a notion of isomorphism which allows label-preserving event renaming, just as event structure isomorphism. Configuration structure isomorphism is weaker than event structure isomorphism but stronger than the various bisimulation relations proposed for partial order models in e.g. [18, 20]. Hence proving equality modulo configuration structure isomorphism immediately implies that it holds under those weaker equivalences, too.

2.7 Definition. Let E be a set of events.

A *stable configuration structure over* E is a tuple $\langle C, \surd, \ell \rangle$ where

- $C \subseteq \mathit{Fin}(E)$ is a set of *finite configurations* such that
  - $\emptyset \in C$;
  - $\forall F, G, H \in C.\ (F \cup G) \subseteq H \Longrightarrow (F \cup G) \in C \wedge (F \cap G) \in C$;
  - $\forall F \in C.\ \forall d, e \in F.\ d \neq e \Longrightarrow \exists G \in C.\ d \in G \Leftrightarrow e \notin G$.
- $\surd \subseteq C$ is a set of *terminated configurations*, sometimes treated as a predicate in postfix notation, such that $F \subset G$ implies $\neg F\surd$ (i.e. terminated configurations must be maximal w.r.t. $\subseteq$);
- $\ell\colon E \to \mathit{Act}$ is a *labelling function.*

[2] There is a subtle difference between these two concerning the presence of certain infinite objects; this is however irrelevant to the discussion at hand.

The class of configuration structures is denoted **C**. We will sometimes use C to denote the entire structure, and $\surd_C$ and ℓ_C to denote the termination predicate and labelling function. $E_C := \bigcup C$ will denote the set of events of C; if C is a configuration structure over E then $E_C \subseteq E$. The following is a standard result; cf. [5].

2.8 Proposition. For every flow event structure $\mathcal{E}$, $CS(\mathcal{E})$ is a configuration structure.

Two configuration structures C, D are *isomorphic*, denoted $C \cong D$, if there exists a bijection $f\colon E_C \to E_D$ which preserves event labelling, such that the pointwise extension of f to sets of events maps C to D and $\surd_C$ to $\surd_D$:

$$\begin{aligned} f(C) &= D \\ \surd_D &= f(\surd_C) \\ \ell_C &= \ell_D \circ f \ . \end{aligned}$$

Note that the $\subseteq$-structure is preserved automatically, because f is a bijection. Together with the mapping from flow event structures to configuration structures, this induces the following equivalence over Σ:

$$P \cong_c Q \quad :\Leftrightarrow \quad CS[\![P]\!] \cong CS[\![Q]\!] \ .$$

As mentioned above, this is *weaker* than flow event structure isomorphism; that is,

$$P \cong_e Q \Longrightarrow P \cong_c Q \ .$$

In this paper we do not consider a compositional semantics on the level of configuration structures, except in the proof of the main theorem where we define synchronization and refinement over **C**. A compositional semantics does however exist; see e.g. [7]. Moreover, the following is known to hold.

2.9 Proposition. $\cong_c$ is a congruence for the operators of Σ.

3 Syntactic versus semantic refinement

In the remainder of this paper we will implicitly assume all terms to be well-formed, except where stated otherwise.

As mentioned in the introduction, the main goal of the paper is to investigate under which conditions syntactic action refinement coincides with its semantic version, presented in the previous section. Here we formally define what syntactic action refinement is. To this aim, we introduce the notation $P\left\{{}^{Q}/_{a}\right\}$ to denote the process term where all the occurrences of action a in P are replaced by Q. This intuitive concept can be rigorously defined by structural induction.

3.1 Definition. Let $P, Q \in \Sigma_{flat}$ be two flat terms.

The operation of *syntactic substitution*, denoted $P\left\{{}^{Q}/_{a}\right\}$, is defined by induction on the syntactical structure of P as follows:

$$\begin{aligned} b\left\{{}^{Q}/_{a}\right\} &:= \begin{cases} Q & \text{if } b = a \\ b & \text{otherwise} \end{cases} \\ (P_1 * P_2)\left\{{}^{Q}/_{a}\right\} &:= (P_1\left\{{}^{Q}/_{a}\right\}) * (P_2\left\{{}^{Q}/_{a}\right\}) \quad \text{where } * \in \{+, ;\} \\ (P_1 \parallel_A P_2)\left\{{}^{Q}/_{a}\right\} &:= (P_1\left\{{}^{Q}/_{a}\right\}) \parallel_{A\{{}^{Q}/_{a}\}} (P_2\left\{{}^{Q}/_{a}\right\}) \end{aligned}$$

where $A\left\{{}^{Q}/_{a}\right\}$ in the last rule is defined as follows:

$$A\left\{{}^{Q}/_{a}\right\} := \begin{cases} (A \setminus \{a\}) \cup L(Q) & \text{if } a \in A \\ A & \text{otherwise.} \end{cases}$$

Note that we also substitute the actions in synchronization sets. The following is immediate.

3.2 Proposition. If $P, Q \in \Sigma_{flat}$ and $a \in Act$ then $P\left\{{}^{Q}/_{a}\right\} \in \Sigma_{flat}$.

We now define a *reduction* function over arbitrary terms which removes all occurrences of refinement operators from a given process expression, from the inside out so that syntactic substitution is only applied to terms which already have been reduced, i.e. to flat terms.

3.3 Definition. The *reduction* of a term $P \in \Sigma$, denoted $red(P)$, is defined inductively on the structure of P as follows:

$$\begin{array}{rcll} red(a) & := & a & \\ red(P * Q) & := & red(P) * red(Q) & \text{where } * \in \{ +, ;, \|_A \} \\ red(P[a \rightsquigarrow Q]) & := & red(P)\left\{{}^{red(Q)}/_{a}\right\} & . \end{array}$$

Note that in the rule for refinement, we have $L(P) \cap L(Q) = \emptyset$ because we only consider well-formed terms. Due to Proposition 3.2, $red(P)\left\{{}^{red(Q)}/_{a}\right\}$ is always defined. The following proposition states that *red* is well-behaved in the sense that the alphabet and set of synchronizing actions of a given term are insensitive to reduction of that term.

3.4 Proposition. If $P \in \Sigma$ then $L(red(P)) = L(P)$ and $S(red(P)) = S(P)$.

Proof. Straightforward, by induction on the structure of P. The only interesting case is refinement, for which the property to be proved is the following: if P and Q are flat terms such that $L(P) \cap L(Q) = \emptyset$ then

$$L\left(P\left\{{}^{Q}/_{a}\right\}\right) = \begin{cases} (L(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in L(P) \\ L(P) & \text{otherwise} \end{cases}$$

$$S\left(P\left\{{}^{Q}/_{a}\right\}\right) = \begin{cases} (S(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in S(P) \\ S(P) \cup S(Q) & \text{if } a \in L(P) \setminus S(P) \\ S(P) & \text{otherwise.} \end{cases}$$

The proof is contained in the appendix (Lemma A.1). □

The aim of this paper can now be phrased as follows: we are looking for general conditions under which

$$P \cong red(P) \tag{4}$$

where $\cong$ is some semantic equivalence, such as flow event structure isomorphism $\cong_e$.

4 Refinement of non-synchronizing actions

In this section we focus our attention on a particular aspect of the problem which may be solved in a simple, neat way: the case when actions to be refined cannot be involved in a synchronization. Recalling that $S(P)$ denotes the set of synchronizing actions, this condition can be formally stated by requiring that $a \notin S(P)$ for any term of the form $P[a \rightsquigarrow Q]$. The general case where a may also occur in $S(P)$ will be the subject of the next section. In order to prove the equivalence result we firstly need the following Lemma.

4.1 Lemma. Let $P, P_1, P_2, Q \in \Sigma_{flat}$ be flat terms, let $a, b \in Act$ and $A \subseteq Act$.

1. $a[a \rightsquigarrow Q] \cong_e Q$
2. $b[a \rightsquigarrow Q] \cong_e b$ provided that $b \neq a$
3. $(P_1; P_2)[a \rightsquigarrow Q] \cong_e P_1[a \rightsquigarrow Q]; P_2[a \rightsquigarrow Q]$
4. $(P_1 + P_2)[a \rightsquigarrow Q] \cong_e P_1[a \rightsquigarrow Q] + P_2[a \rightsquigarrow Q]$
5. $(P_1 \parallel_A P_2)[a \rightsquigarrow Q] \cong_e P_1[a \rightsquigarrow Q] \parallel_A P_2[a \rightsquigarrow Q]$ provided that $a \notin A$.

Proof. Let us assume that $[\![P_i]\!] = \langle E_i, \prec_i, \#_i, \ell_i \rangle$ and $[\![P_i[a \rightsquigarrow Q]]\!] = \langle E'_i, \prec'_i, \#'_i, \ell'_i \rangle$ for $i = 1, 2$. Finally let $\langle E_Q, \prec_Q, \#_Q, \ell_Q \rangle$ denote $[\![Q]\!]$.

1. Let $[\![a]\!] = \langle \{e\}, \emptyset, \emptyset, (e, a) \rangle$ and $[\![a[a \rightsquigarrow Q]]\!] = \langle E, \prec, \#, \ell \rangle$, hence $E = \{e\} \times E_Q$. Therefore the function $f\colon E \to E_Q$ defined by $f\colon (e, d) \mapsto d$ is a bijection. Preservation of flow and conflict relations, as well as of the labelling, are immediate; hence f is an isomorphism.

2. Trivial: in both structures there is only one event, which is labelled b.

3. Let $[\![(P_1; P_2)[a \rightsquigarrow Q]]\!] = \langle E, \prec, \#, \ell \rangle$ and $[\![P_1[a \rightsquigarrow Q]; P_2[a \rightsquigarrow Q]]\!] = \langle E', \prec', \#', \ell' \rangle$. According to Definitions 2.4 and 2.6 we have

$$\begin{aligned} E &= \{\, (e, *) \mid e \in (E_1 \setminus \ell_1^{-1}(a)) \cup (E_2 \setminus \ell_2^{-1}(a)) \,\} \\ &\quad \cup \{\, (e, d) \mid e \in E_1 \cup E_2 \wedge \ell(e) = a \wedge d \in E_Q \,\} \\ E'_i &= \{\, (e, *) \mid e \in E_i \setminus \ell_i^{-1}(a) \,\} \\ &\quad \cup \{\, (e, d) \mid e \in E_i \wedge \ell_i(e) = a \wedge d \in E_Q \,\} \ . \end{aligned}$$

   Since $E' = E'_1 \cup E'_2 = E$, we can take as isomorphism simply the identity function. The flow relation is preserved as $\prec$ can be partitioned in the following subsets, making explicit that it coincides with $\prec'$:

$$\prec = \prec_{P_1 \times Q} \cup \prec_{P_2 \times Q} \cup \prec_{(P_1 \times P_2) \times Q}$$

   where

$$\begin{aligned} (e, e') \prec_{P_1 \times Q} (d, d') &\iff e \prec_1 d \vee (e = d \wedge e' \prec_Q d') \\ (e, e') \prec_{P_2 \times Q} (d, d') &\iff e \prec_2 d \vee (e = d \wedge e' \prec_Q d') \\ (e, e') \prec_{(P_1 \times P_2) \times Q} (d, d') &\iff e \in E_1 \wedge d \in E_2 \end{aligned}$$

   Obviously, $\prec_{P_i \times Q} = \prec'_i$, for $i = 1, 2$, and $\prec_{(P_1 \times P_2) \times Q} = E'_1 \times E'_2$. Hence, $\prec = \prec'$. A similar argument can be produced for the conflict relation and the labelling function.

4. Similar to the previous one and thus omitted.

5. Let $[\![(P_1 \parallel_A P_2)[a \rightsquigarrow Q]]\!] = \langle E, \prec, \#, \ell \rangle$ and $[\![P_1[a \rightsquigarrow Q] \parallel_A P_2[a \rightsquigarrow Q]]\!] = \langle E', \prec', \#', \ell' \rangle$. According to Definitions 2.5 and 2.6 we have

$$\begin{aligned} E = {} & \{\, ((e, *), *) \mid e \in E_1 \setminus \ell_1^{-1}(a) \,\} && (= X_1) \\ & \cup \{\, ((e, *), d) \mid e \in \ell_1^{-1}(a) \wedge d \in E_Q \,\} && (= X_2) \\ & \cup \{\, ((*, e), *) \mid e \in E_2 \setminus \ell_2^{-1}(a) \,\} && (= X_3) \\ & \cup \{\, ((*, e), d) \mid e \in \ell_2^{-1}(a) \wedge d \in E_Q \,\} && (= X_4) \\ & \cup \{\, ((e_1, e_2), *) \mid \ell_1(e) = \ell_2(e) \in A \,\} && (= X_5) \ . \end{aligned}$$

A similar partition can be imposed over the events in E'.

$$\begin{array}{rcll} E' & = & \{\, ((e,*),*) \mid e \in E_1 \setminus \ell_1^{-1}(a) \,\} & (= Y_1) \\ & & \cup \{\, ((e,d),*) \mid e \in \ell_1^{-1}(a) \wedge d \in E_Q \,\} & (= Y_2) \\ & & \cup \{\, (*,(e,*)) \mid e \in E_2 \setminus \ell_2^{-1}(a) \,\} & (= Y_3) \\ & & \cup \{\, (*,(e,d)) \mid e \in \ell_2^{-1}(a) \wedge d \in E_Q \,\} & (= Y_4) \\ & & \cup \{\, ((e_1,*),(e_2,*)) \mid \ell_1(e) = \ell_2(e) \in A \,\} & (= Y_5)\ . \end{array}$$

The bijection $f: E \to E'$, which is our candidate isomorphism, is the union $\bigcup_{1 \le i \le 5} f_i$ where $f_i: X_i \to Y_i$ are defined as follows:

$$\begin{array}{rrcl} f_1\colon & ((e,*),*) & \mapsto & ((e,*),*) \\ f_2\colon & ((e,*),d) & \mapsto & ((e,d),*) \\ f_3\colon & ((*,e),*) & \mapsto & (*,(e,*)) \\ f_4\colon & ((*,e),d) & \mapsto & (*,(e,d)) \\ f_5\colon & ((e_1,e_2),*) & \mapsto & ((e_1,*),(e_2,*)) \end{array}$$

Functions f_{1-5} are obviously bijective, as well as label-preserving. For the flow relation, let us consider two events $((e_1,e_2),d), ((e_1',e_2'),d') \in E$ — any of the components can be $*$ — such that $((e_1,e_2),d) \prec ((e_1',e_2'),d')$. This may be due only to one of the following:

$e_1 \prec_1 e_1'$**:** Whichever is the actual shape of the two events (the reader can try the several possibilities), the two events $f((e_1,e_2),d)$ and $f((e_1',e_2'),d')$ are in the flow relation $\prec'$ because $(e_1,-) \prec_1' (e_1',-)$.

$e_2 \prec_2 e_2'$**:** Symmetrically, the two events $f((e_1,e_2),d)$ and $f((e_1',e_2'),d')$ are in the flow relation $\prec'$ because $(e_2,-) \prec_2' (e_2',-)$.

$e_1 = e_1', e_2 = e_2', d \prec_Q d'$**:** Without loss of generality, we can assume that $e_2 = *$. It follows trivially from the definitions that $((e_1,d),*) \prec' ((e_1',d'),*)$.

Also the reverse holds, i.e., whenever two events in the image of f are in the flow relation $\prec'$, then their pre-images are in the relation $\prec$. Similar arguments may be used for proving the preservation of the conflict relation. □

The following example shows that rule 5 of the lemma does *not* hold in general for *non-well-formed terms*:

4.2 Example. Let $P_1 = Q = a$, $A = \{a\}$ and $P_2 = a;b$.

$$\begin{array}{rcl} (P_1 \parallel_A P_2)[b \rightsquigarrow Q] & \cong_e & (a \parallel_{\{a\}} a);a \\ P_1[b \rightsquigarrow Q] \parallel_A P_2[b \rightsquigarrow Q] & \cong_e & a \parallel_{\{a\}} (a;a)\ . \end{array}$$

These terms describe different behaviours. The upper one will execute action a twice and terminate successfully, whereas the lower one can execute only one a, whereafter it deadlocks: the right hand synchronization component wants to execute one more a in synchrony with the other, but the other component is already finished.

4.3 Theorem. Let $P, Q \in \Sigma_{flat}$ and $a \in Act$. If $a \notin S(P)$ then $P[a \rightsquigarrow Q] \cong_e P\left\{{}^{Q}/_{a}\right\}$.

Proof. By induction on the syntactic structure of P. The base cases are when P is an action. If $P = a$, then $a[a \rightsquigarrow Q] \cong_e Q$ by 1 of Lemma 4.1, and $Q \cong_e a\left\{{}^{Q}/_{a}\right\}$ because of Definition 3.1. Analogously if $P = b$. For the inductive case, let $* \in \{ +, ;, \parallel_A \}$; then

$$\begin{array}{rcll} (P_1 * P_2)[a \rightsquigarrow Q] & \cong_e & P_1[a \rightsquigarrow Q] * P_2[a \rightsquigarrow Q] & \text{(lemma 4.1)} \\ & \cong_e & (P_1\left\{{}^{Q}/_{a}\right\}) * (P_2\left\{{}^{Q}/_{a}\right\}) & \text{(induction hypothesis and congruence of } \cong_e) \\ & = & (P_1 * P_2)\left\{{}^{Q}/_{a}\right\} & \text{(definition 3.1).} \end{array}$$

$$\frac{}{\vdash_i a} \qquad \frac{\vdash_i P, Q \quad * \in \{+, ;, \|_A\}}{\vdash_i P * Q} \qquad \frac{\vdash_i P, Q \quad a \notin S(P)}{\vdash_i P[a \rightsquigarrow Q]}$$

Table 3: Interference freedom

If $* = \|_A$ then Lemma 4.1 is applicable because $A \subseteq S(P)$, hence $a \notin A$. □

The final corollary, which extends the above result to the full language, relies on a further predicate, called *interference freedom* and denoted $\vdash_i$. This is defined in Table 3. Interference freedom essentially ensures that none of the refined actions in a term appear in a synchronization, and hence Theorem 4.3 is always applicable. Now a straightforward consequence of the theorem above is the following:

4.4 Corollary. Let $P \in \Sigma$. If $\vdash_i P$ then $P \cong_e red(P)$.

Proof. Straightforward by induction. We show the case for refinement:

$$\begin{array}{lcll} red(P[a \rightsquigarrow Q]) & = & red(P)\left\{{}^{red(Q)}/_{a}\right\} & \\ & \cong_e & red(P)[a \rightsquigarrow red(Q)] & \text{(theorem 4.3)} \\ & \cong_e & P[a \rightsquigarrow Q] & \text{(induction and congruence of } \cong_e\text{).} \end{array}$$

As mentioned above, Theorem 4.3 is applicable because by definition, $\vdash_i P[a \rightsquigarrow Q]$ guarantees $a \notin S(P)$ $(= S(red(P)))$. □

5 Refinement of synchronizing actions

In this section we compare semantic and syntactic refinement for non-interference-free terms, i.e. terms in which it is allowed to refine synchronization actions. The following example shows that once the condition $a \notin A$ in rule 5 of Lemma 4.1 is violated, the corresponding equation does not hold any more.

5.1 Example. Let $P_1 = P_2 = a$, $A = \{a\}$ and $Q = b$.

$$\begin{array}{rcl} (P_1 \|_A P_2)[a \rightsquigarrow Q] & \cong_e & b \|_{\{b\}} b \\ P_1[a \rightsquigarrow Q] \|_A P_2[a \rightsquigarrow Q] & \cong_e & b \|_{\{a\}} b \ . \end{array}$$

These terms are not equivalent: in the upper one, b is executed only once, whereas in the lower it is executed twice independently.

We can try to repair this situation by formulating a more accurate rule for distributing refinement over parallel composition.

5.1 Distributing refinement over synchronization

Since we are studying the correspondence of semantic and syntactic refinement, it is a natural choice to reuse the definition of syntactic substitution as a distribution rule for refinement, yielding

$$(P_1 \|_A P_2)[a \rightsquigarrow Q] \cong_e P_1[a \rightsquigarrow Q] \|_{(A \setminus \{a\}) \cup L(Q)} P_2[a \rightsquigarrow Q] \ . \tag{5}$$

(There are alternative ways of distributing refinement over synchronization. In Section 7 we briefly discuss one particular other choice based on a CCS-like synchronization operator.)

Example 5.1 above is indeed repaired by this change, because now the second term (in which refinement is distributed over the subterms) becomes

$$P_1[a \rightsquigarrow Q] \parallel_{(A \setminus \{a\}) \cup L(Q)} P_2[a \rightsquigarrow Q] \cong_e b \parallel_{\{b\}} b$$

which is equivalent to the first (undistributed) term. It is however important to note that there are instances of P_1, P_2 and Q for which we do not expect (5) to hold under *any* deadlock-sensitive equivalence relation. For instance, the following terms are not even completed trace equivalent (b is a completed trace on the right hand side but not on the left hand side):

$$(a \parallel_{\{a\}} a)[a \rightsquigarrow b; c_1 + b; c_2] \quad \text{and} \quad (a[a \rightsquigarrow b; c_1 + b; c_2]) \parallel_{\{b,c_1,c_2\}} (a[a \rightsquigarrow b; c_1 + b; c_2]) \ .$$

Hence at this point, instead of looking for a semantic relation under which (5) holds always, we fix a relation that we consider reasonable and investigate conditions under which it holds. Unfortunately, the very strong semantic notion of flow event structure isomorphism is *not* reasonable in this sense, as the following example shows.

5.2 Example. Let $P_1 = P_2 = a$ and $Q = b$. One would expect the following to hold for a reasonable equivalence relation $\cong$:

$$a[a \rightsquigarrow Q] \cong b \cong b \parallel_{\{b\}} b \cong a[a \rightsquigarrow Q] \parallel_{\{b\}} a[a \rightsquigarrow Q] \ .$$

However, this is not satisfied if we replace $\cong$ by $\cong_e$, due to the flow event structure construction for synchronization, which introduces *inconsistent events* (i.e. events e such that $e \# e$). In fact we have $b \ncong_e b \parallel_{\{b\}} b$ and hence

$$(P_1 \parallel_{\{a\}} P_2)[a \rightsquigarrow Q] \ncong_e P_1[a \rightsquigarrow Q] \parallel_{\{b\}} P_2[a \rightsquigarrow Q] \ .$$

Hence the validity of the distribution rule (5) is hindered by the fact that flow event structure isomorphism is more discriminating than intuitively justified. We repair the situation by using the weaker *configuration structure isomorphism* introduced in Section 2.3.

It turns out that we can give necessary and sufficient conditions for the validity of (5) under $\cong_c$. This extends rule 5 of Lemma 4.1 to non-interference-free terms. Note that this result necessarily depend on the choice of semantics: in a stronger semantics our conditions will in general no longer be sufficient (Example 5.2 already shows that) whereas in a weaker semantics they will no longer be necessary.

To formulate our conditions we define a number of properties over configuration structures. For this purpose the following notation is useful: if C is a configuration structure and $F, G \in C$ then

$$F \xrightarrow{a}_C G \quad :\Leftrightarrow \quad \exists e \in G.\, F = G \setminus \{e\} \wedge \ell_C(e) = a \ .$$

We will drop the subscript C when it is clear from the context. Furthermore we define

$$\pi_i(F) \quad := \quad \{\, e_i \in \mathbf{E} \mid (e_1, e_2) \in F \,\}$$

5.3 Definition. Let C be a configuration structure; let a be an action.

- a is *executed* in C (at F) if $F \xrightarrow{a} G$ for some $F, G \in C$;
- a is *initial* in C if $\emptyset \xrightarrow{a} F$ for some $F \in C$.
- a is *noninitial* in C if $\emptyset \neq F \xrightarrow{a} G$ for some $F, G \in C$; otherwise a is *initial-only* in C (note that initial-only does not imply that a is in fact executed).

- a is *nondeterministic* in C if $F \xrightarrow{a} G$ and $F \xrightarrow{a} H \neq G$ for some $F, G, H \in C$; otherwise a is *deterministic* in C;
- a is *auto-concurrent* in C (at F) if $F \xrightarrow{a} G$ and $F \xrightarrow{a} H \neq G$ and $G \cup H \in C$ for some $F, G, H \in C$; otherwise a is *auto-sequential* in C (at F). (Note that auto-sequentiality of a also does not imply that a is actually executed, whereas auto-concurrency implies nondeterminism.)

The following is a derived property that is defined only over structures of the form $C = C_1 \parallel_A C_2$, where $C_i \in \mathbf{C}$ for $i = 1, 2$:

- $a \in A$ is *two-way sequential* in C if a is auto-sequential in C_i at $\pi_i(F)$ for both $i = 1, 2$ whenever a is executed in C at F.

The following properties concern C as a whole, without reference to any particular action a:

- C is *deterministic* (as a whole) if every action is deterministic in C;
- C is *distinct* if C is deterministic and every initial action in C is initial-only;
- C is *atomic* if C is deterministic and *every* action is initial-only in C (hence all nonempty configurations in C are singleton sets).

We will say that a is executed, deterministic etc. in a *process term* P if it is executed, deterministic etc. in $CS[\![P]\!]$. The property of *two-way sequentiality* is the least familiar: it implies that every *execution of* a in a synchronization is auto-sequential in both synchronizing partners. It is slightly weaker than requiring that a is auto-sequential in both synchronizing partners, since all a-autoconcurrent states in the partners may be unreachable (e.g. because of synchronization deadlocks), in which case a is still two-way sequential.

5.4 Example. If $P_1 = a; (b \parallel_\emptyset b) + b$ and $P_2 = a + b; a$ then b is auto-concurrent in P_1 but two-way sequential in $P_1 \parallel_{\{a,b\}} P_2$.

We now present the main theorem of this paper.

5.5 Theorem. Let $P_1, P_2, Q \in \Sigma$ and $a \in A \subseteq Act$.
Refinement distributes over synchronizaton according to

$$(P_1 \parallel_A P_2)[a \rightsquigarrow Q] \cong_c P_1[a \rightsquigarrow Q] \parallel_{(A \setminus \{a\}) \cup L(Q)} P_2[a \rightsquigarrow Q]$$

if and only if one of the following is satisfied:

C1. a is not executed in $P_1 \parallel_A P_2$;

C2. a is two-way sequential in $P_1 \parallel_A P_2$, and Q is deterministic;

C3. a is auto-sequential in $P_1 \parallel_A P_2$, and Q is distinct;

C4. Q is atomic.

Proof strategy. We only give an outline of the proof here; the various steps are proved in Appendix A. First let us analyze the terms on both sides of the proposed new distribution rule (5). We define

$$\begin{aligned} C &= CS[\![(P_1 \parallel_A P_2)[a \rightsquigarrow Q]]\!] \\ D &= CS[\![P_1[a \rightsquigarrow Q] \parallel_{(A \setminus \{a\}) \cup L(Q)} P_2[a \rightsquigarrow Q]]\!] \ . \end{aligned}$$

We can partially construct the event sets: $E_C \subseteq \bigcup_{1\leq i\leq 4} X_i$ and $E_D \subseteq \bigcup_{1\leq i\leq 4} Y_i$ where

$$\begin{array}{lcl} X_1 & := & \{\,((d_1,d_2),e) \in ((E_1 \times E_2) \times E_Q) \mid \ell_1(d_1) = \ell_2(d_2) = a\,\} \\ X_2 & := & \{\,((d_1,d_2),*) \mid d_i \in E_i \wedge \ell_1(d_1) = \ell_2(d_2) \in (A \setminus \{a\})\,\} \\ X_3 & := & \{\,((d,*),*) \mid d \in E_1 \wedge \ell_1(d) \notin A\,\} \\ X_4 & := & \{\,((*,d),*) \mid d \in E_2 \wedge \ell_2(d) \notin A\,\} \end{array}$$

$$\begin{array}{lcl} Y_1 & := & \{\,((d_1,e_1),(d_2,e_2)) \mid \ell_1(d_1) = \ell_2(d_2) = a \wedge \ell_Q(e_1) = \ell_Q(e_2)\,\} \\ Y_2 & := & \{\,((d_1,*),(d_2,*)) \mid d_i \in E_i \wedge \ell_1(d_1) = \ell_2(d_2) \in (A \setminus \{a\})\,\} \\ Y_3 & := & \{\,((d,*),*) \mid d \in E_1 \wedge \ell_1(d) \notin A\,\} \\ Y_4 & := & \{\,(*,(d,*)) \mid d \in E_2 \wedge \ell_2(d) \notin A\,\} \end{array}$$

Now there is a natural candidate function $f\colon E_C \to E_D$ to prove $C \cong D$, viz. the restriction of the union $\bigcup_{1\leq i\leq 4} f_i$ to E_C where $f_i\colon X_i \to Y_i$ are defined as follows:

$$\begin{array}{lrcl} f_1\colon & ((d_1,d_2),e) & \mapsto & ((d_1,e),(d_2,e)) \\ f_2\colon & ((d_1,d_2),*) & \mapsto & ((d_1,*),(d_2,*)) \\ f_3\colon & ((d,*),*) & \mapsto & ((d,*),*) \\ f_4\colon & ((*,d),*) & \mapsto & (*,(d,*)) \end{array}$$

f_2–f_4 are obviously bijective. For f_1 this is not immediately clear; surjectivity requires that that $e_1 = e_2$ for every $((d_1,e_1),(d_2,e_2)) \in (E_D \cap Y_1)$. However, all f_j, and thereby also f, are clearly injective. Now the proof proceeds as follows.

1. Prove $F_C \in C \Longrightarrow f(F_C) \in D$, independent of conditions C1–4 (Lemma A.8).
2. Prove $F_D \in D \Longrightarrow \exists F_C \in C.\, f(F_C) = F_D$ (Lemma A.9) under each of the conditions C1–4. This proves that f is onto E_D; because we already knew f to be injective, it follows that $f\colon E_C \to E_D$ is bijective and $f(C) = D$.
3. Prove $F_C\surd_C \iff f(F_C)\surd_D$ (Lemma A.10). Because we know that (the pointwise extension of) f is a bijection from C to D this proves $f(\surd_C) = \surd_D$.
4. $\ell_D = \ell_C \circ f^{-1}$ follows immediately from the analysis of the event sets and the definition of f, together with the fact that f is bijective.

This concludes the proof of the "if" part of the theorem.

5. If $C \cong_c D$ then $|C| = |D|$ because we are dealing only with finite terms; moreover f is injective; hence if $C \cong_c D$ then $f(C) = D$. Now prove that each of the following conditions is sufficient to construct a configuration in D which is not in $f(C)$ (Lemma A.11).

   D1. a is executed in $P_1 \parallel_A P_2$ and Q is nondeterministic;
   D2. a is not two-way sequential in $P_1 \parallel_A P_2$ and Q is not distinct;
   D3. a is autoconcurrent in $P_1 \parallel_A P_2$ and Q is not atomic.

This concludes the proof of the "only if" part. □

The following example shows a case where distribtivity fails because the necessary conditions are not satisfied.

5.6 Example. Consider $P_1 := a; c \parallel_\emptyset a; c$, $P_2 := a$, $A := \{a\}$ and $Q := b; b$. It follows that a is not two-way sequential in $P_1 \parallel_A P_2$ and Q is not distinct, and in fact we have

$$\begin{array}{lcl} (P_1 \parallel_A P_2)[a \rightsquigarrow Q] & \cong_c & ((a; c \parallel_\emptyset a; c) \parallel_{\{a\}} a)[a \rightsquigarrow b; b] \\ & \not\cong_c & (b; b; c \parallel_\emptyset b; b; c) \parallel_{\{b\}} b; b \\ & \cong_c & (P_1[a \rightsquigarrow Q]) \parallel_{(A\setminus\{a\})\cup L(Q)} (P_2[a \rightsquigarrow Q])\ . \end{array}$$

5.2 The language of reducible terms

The conditions of Theorem 5.5 are based on the *semantic* properties in Definition 5.3. We are however also interested in a *syntactic* characterization of the (sub)language in which syntactic and semantic refinement coincide, i.e. which are reducible in the sense that $P \cong_c red(P)$. Such a syntactic characterization will allow us to decide, on the basis of a straightforward analysis, whether a given term is reducible.

We will only give *sufficient* syntactic conditions; we argue that it is useless to try giving necessary and sufficient conditions for e.g. the occurrence of an action, since such results could never be extended to a language with recursion: this would imply solving the halting problem. Also, necessary conditions are only necessary with respect to a given semantics: when moving to a weaker equivalence relation they are in general no longer necessary. Sufficient conditions, however, remain sufficient even with respect to weaker equivalences than $\cong_c$ —which is important since as mentioned before, most partial order equivalences found in the literature are indeed weaker than $\cong_c$.

We do not claim that our conditions are optimal in the sense that they identify the maximal number of reducible terms under $\cong_c$. For instance, the syntactic criterion for the occurrence of an action will be its presence in the alphabet of the term; there are many ways to improve on this. We have chosen a fairly direct encoding of the semantic properties, intending to show the principle of syntactic conditions rather than give the most effective solution.

Table 4 defines various functions from Σ to $\mathbf{2}^{Act}$ inductively on the structure of the terms. I returns the *initial actions*, and D the set of *distributed actions* which may occur *auto-concurrently*. It follows that $I(P) \subseteq L(P)$ and $D(P) \subseteq L(P)$ for all $P \in \Sigma$. SH and SD serve a more complicated purpose: they investigate subterms of the form $P_1 \parallel_A P_2$ and record which of the synchronizing actions in A in such a subterm are *distributed in one operand* (for SH) or *distributed in both operands* (for SD). This information is used to approximate the awkward semantical property of *two-way sequentiality*. It follows that $SD(P) \subseteq SH(P) \subseteq D(P) \cap S(P)$ for all $P \in \Sigma$. The following proposition states that these functions (all of which are linear in the size of their arguments) indeed provide characterizations for the corresponding semantic properties.

5.7 Proposition.

1. If a is executed in P then $a \in L(P)$;
2. a is initial in P if and only if $a \in I(P)$;
3. If a is auto-concurrent in P then $a \in D(P)$;

Proof. Straightforward; deferred to Appendix A. □

The following proposition states that all the syntactic functions above are insensitive to the reduction function *red*. This is necessary to make sure that in nested refinements, when a term is syntactically classified as reducible, this decision is not revoked after part of the reduction is done and some of the inner refinements are removed.

5.8 Proposition. If $P \in \Sigma$ then $f(red(P)) = f(P)$ for all $f = I, C, SH, SD$.

Proof. Straightforward, by induction on the structure of P. Because *red* does not affect the outermost operator of P except if it is refinement, this is the only interesting case; the relevant property is stated and proved in the appendix (Lemma A.12). □

Table 5 defines a number of predicates over Σ, intended to capture the rest of the semantic properties of Definition 5.3 in terms of syntax. $\vdash_{det} \subseteq \Sigma$ captures the notion of *determinism*, and

$$
\begin{array}{rcl}
I(a) & := & \{a\} \\
I(P+Q) & := & I(P) \cup I(Q) \\
I(P;Q) & := & I(P) \\
I(P \parallel_A Q) & := & ((I(P) \cup I(Q)) \setminus A) \cup (I(P) \cap I(Q) \cap A) \\
I(P[a \rightsquigarrow Q]) & := & \begin{cases} (I(P) \setminus \{a\}) \cup I(Q) & \text{if } a \in I(P) \\ I(P) & \text{otherwise.} \end{cases} \\
\hline
D(a) & := & \emptyset \\
D(P * Q) & := & D(P) \cup D(Q) \ \text{, where } * \in \{+, ;\} \\
D(P \parallel_A Q) & := & (D(P) \cap D(Q) \cap A) \cup (D(P) \cup D(Q) \cup (L(P) \cap L(Q))) \setminus A \\
D(P[a \rightsquigarrow Q]) & := & \begin{cases} (D(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in D(P) \\ D(P) \cup D(Q) & \text{if } a \in L(P) \setminus D(P) \\ D(P) & \text{otherwise.} \end{cases} \\
\hline
SH(a) & := & \emptyset \\
SH(P * Q) & := & SH(P) \cup SH(Q) \ \text{, where } * \in \{+, ;\} \\
SH(P \parallel_A Q) & := & SH(P) \cup SH(Q) \cup ((D(P) \cup D(Q)) \cap A) \\
SH(P[a \rightsquigarrow Q]) & := & \begin{cases} (SH(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in SH(P) \\ SH(P) \cup D(Q) & \text{if } a \in S(P) \setminus SH(P) \\ SH(P) \cup SH(Q) & \text{if } a \in L(P) \setminus S(P) \\ SH(P) & \text{otherwise.} \end{cases} \\
\hline
SD(a) & := & \emptyset \\
SD(P * Q) & := & SD(P) \cup SD(Q) \ \text{, where } * \in \{+, ;\} \\
SD(P \parallel_A Q) & := & SD(P) \cup SD(Q) \cup (D(P) \cap D(Q) \cap A) \\
SD(P[a \rightsquigarrow Q]) & := & \begin{cases} (SD(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in SD(P) \\ SD(P) \cup D(Q) & \text{if } a \in S(P) \setminus SD(P) \\ SD(P) \cup SD(Q) & \text{if } a \in L(P) \setminus S(P) \\ SD(P) & \text{otherwise.} \end{cases}
\end{array}
$$

Table 4: Initial and distributed actions

$\vdash_{dis} \subseteq \Sigma$ is intended to capture the notion of *distinctness*. It follows that $\vdash_{dis} \subseteq \vdash_{det}$. Finally, $\vdash_{red}$ defines the notion of *reducibility*: in reducible terms, semantic refinement can be interpreted as syntactic substitution. The following proposition expresses that the sublanguages induced by these predicates indeed satisfy the intended properties: in particular, we now have sufficient conditions for determinism, distinctness and atomicity.

5.9 Proposition.

1. If $\vdash_{det} P$ then P is deterministic;
2. If $\vdash_{dis} P$ then P is distinct;
3. If $\vdash_{dis} P$ and $L(P) = I(P)$ then P is atomic.

Proof. Straightforward; proofs of the more interesting cases are contained in Appendix A. □

Similar to the calculation of the action sets (cf. Proposition 5.8), we also need to know that the predicates defined above are insensitive to the process of reduction, with the same motivation: when some inner refinement operator of a number of nested refinements is reduced away doing

$$\frac{}{\vdash_{det} a} \qquad \frac{\vdash_{det} P, Q \quad I(P) \cap I(Q) = \emptyset}{\vdash_{det} P + Q} \qquad \frac{\vdash_{det} P, Q}{\vdash_{det} P; Q} \qquad \frac{\vdash_{det} P, Q \quad L(P) \cap L(Q) \subseteq A}{\vdash_{det} P \parallel_A Q}$$

$$\frac{\vdash_{det} P \quad a \notin L(P)}{\vdash_{det} P[a \rightsquigarrow Q]} \qquad \frac{\vdash_{det} P, Q}{\vdash_{det} P[a \rightsquigarrow Q]}$$

$$\frac{}{\vdash_{dis} a} \qquad \frac{\vdash_{dis} P, Q \quad I(P) \cap L(Q) = L(P) \cap I(Q) = \emptyset}{\vdash_{dis} P + Q}$$

$$\frac{\vdash_{dis} P \quad \vdash_{det} Q \quad I(P) \cap L(Q) = \emptyset}{\vdash_{dis} P; Q} \qquad \frac{\vdash_{dis} P, Q \quad L(P) \cap L(Q) \subseteq A}{\vdash_{dis} P \parallel_A Q}$$

$$\frac{\vdash_{dis} P \quad a \notin L(P)}{\vdash_{dis} P[a \rightsquigarrow Q]} \qquad \frac{\vdash_{dis} P \quad \vdash_{det} Q \quad a \in L(P) \setminus I(P)}{\vdash_{dis} P[a \rightsquigarrow Q]} \qquad \frac{\vdash_{dis} P, Q}{\vdash_{dis} P[a \rightsquigarrow Q]}$$

$$\frac{}{\vdash_{red} a} \qquad \frac{\vdash_{red} P, Q \quad * \in \{ +, ;, \parallel_A \}}{\vdash_{red} P * Q}$$

$$\frac{\vdash_{red} P, Q \quad a \notin S(P)}{\vdash_{red} P[a \rightsquigarrow Q]} \qquad \frac{\vdash_{red} P, Q \quad a \notin SH(P) \quad \vdash_{det} Q}{\vdash_{red} P[a \rightsquigarrow Q]}$$

$$\frac{\vdash_{red} P, Q \quad a \notin SD(P) \quad \vdash_{dis} Q}{\vdash_{red} P[a \rightsquigarrow Q]} \qquad \frac{\vdash_{red} P, Q \quad \vdash_{dis} Q \quad L(Q) = I(Q)}{\vdash_{red} P[a \rightsquigarrow Q]}$$

Table 5: Determinism, distinctness and reducibility

red, properties of the term as a whole should not be affected. This is formulated in the following proposition.

5.10 Proposition. Let $P \in \Sigma$.

1. If $\vdash_{det} P$ then $\vdash_{det} red(P)$;
2. If $\vdash_{dis} P$ then $\vdash_{dis} red(P)$.
3. If $\vdash_{red} P$ then $\vdash_{red} red(P)$.

Proof. By induction on the structure of P. Again, just as for Proposition 5.8, the proof is trivial except for refinement, because *red* does not actually affect the top level operator. For the case of refinement the necessary property is stated and proved in the appendix (Lemma A.13). □

It is our intention that the syntactically decidable criterion of reducibility provides sufficient (but not necessary) conditions for the semantic properties discussed in Theorem 5.5. This leads to the following theorem, which states that for reducible terms, semantic and syntactic refinement coincide, at least for flat terms.

5.11 Theorem. Let $P, Q \in \Sigma_{flat}$ and $a \in Act$. If $\vdash_{red} P[a \rightsquigarrow Q]$ then $P[a \rightsquigarrow Q] \cong_c P\left\{{}^{Q}/_{a}\right\}$.

Proof. By induction on the structure of P. The cases are analogous to the proof of Theorem 4.3, except if $P = P_1 \parallel_A P_2$ such that $a \in A$. By induction $P_i\left\{{}^{Q}/_{a}\right\} \cong_c P_i[a \rightsquigarrow Q]$ for both $i = 1, 2$. There are three subcases.

- $a \notin SH(P)$ and $\vdash_{det} Q$. It follows, by construction of SH, that $a \notin D(P_1) \cup D(P_2)$; hence by Proposition 5.7, a is not auto-concurrent in P_1 or P_2; hence a is two-way sequential in $P_1 \parallel_A P_2$. In addition, Q is deterministic by Proposition 5.9.

- $a \notin SD(P)$ and $\vdash_{dis} Q$. It follows, by construction of SD, that $a \in A \setminus (D(P_1) \cap D(P_2))$; hence $a \notin D(P)$ and by Proposition 5.7, a is auto-sequential in $P_1 \parallel_A P_2$. In addition, Q is distinct by Proposition 5.9.

- $\vdash_{dis} Q$ and $L(Q) = I(Q)$. It follows by Proposition 5.9 that Q is atomic.

In each of these cases, due to Theorem 5.5 we have

$$\begin{aligned} P[a \rightsquigarrow Q] &\cong_c P_1[a \rightsquigarrow Q] \parallel_{(A \setminus \{a\}) \cup L(Q)} P_2[a \rightsquigarrow Q] \\ &\cong_c P_1\left\{{}^{Q}/_{a}\right\} \parallel_{A\{{}^{Q}/_{a}\}} P_2\left\{{}^{Q}/_{a}\right\} \\ &= P\left\{{}^{Q}/_{a}\right\} \ . \end{aligned}$$

This concludes the proof. □

The following corollary extends the above result to Σ, using Propositions 5.8 and 5.10 which state that our syntactic machinery is insensitive to the application of the function red: by removing refinement operators from the inside out using red, it is ensured that syntactic substitution is applied only to flat terms. It follows that every reducible term can be rewritten to a flat term.

5.12 Corollary. Let $P \in \Sigma$. If $\vdash_{red} P$ then $P \cong_c red(P)$.

Proof. By induction on the structure of P. The only interesting case is refinement. Assume $P = P_1[a \rightsquigarrow Q]$; then $\vdash_{red} P_1$ and $\vdash_{red} Q$, implying $\vdash_{red} red(P_1)$ and $\vdash_{red} red(Q)$ by Proposition 5.10, and moreover one of the following holds:

- $a \notin S(P)$, hence $a \notin S(red(P_1))$ by Proposition 3.4;

- $a \notin SH(P)$ and $\vdash_{det} Q$, hence $a \notin SH(red(P))$ by Proposition 5.8 and $\vdash_{det} red(Q)$ by Proposition 5.10.

- $a \notin SD(P)$ and $\vdash_{dis} Q$, hence $a \notin SD(red(P))$ by Proposition 5.8 and $\vdash_{dis} red(Q)$ by Proposition 5.10.

- $L(Q) = I(Q)$ and $\vdash_{dis} Q$, hence $L(red(Q)) = I(red(Q))$ by Propositions 3.4 and 5.8 and $\vdash_{dis} red(Q)$ by Proposition 5.10.

Each of these cases implies $\vdash_{red} red(P_1)[a \rightsquigarrow red(Q)]$.

$$\begin{aligned} red(P) &= red(P_1)\left\{{}^{red(Q)}/_{a}\right\} \\ &\cong_c red(P_1)[a \rightsquigarrow red(Q)] \\ &\cong_c P_1[a \rightsquigarrow Q] \end{aligned}$$

The second step is by Theorem 5.11, and the third one by the induction hypothesis and the fact that $\cong_c$ is a congruence with respect to refinement. □

6 Non-well-formed terms

This section is devoted to show that there is rather simple a way to deal with terms $P[a \rightsquigarrow Q]$ not satisfying the well-formedness condition (3). The possible confusion of abstraction levels, generated by the substitution of Q for a in P when $L(P) \cap L(Q) \neq \emptyset$, can be removed by suitably renaming the actions of Q. Hence one has to consider a slightly more general language, where also a renaming operator is allowed. We will show that any non well-formed term P is equivalent (i.e., gives rise to isomorphic flow event structures or configuration structures) to a term R of the extended language and satisfying (3).

6.1 Definition. Let φ: $Act \to Act$ be a total function over the set Act of actions, *not necessarily injective*. The set of the terms generated by the following syntax

$$R ::= a \mid R + R \mid R; R \mid R \parallel_A R \mid R[a \rightsquigarrow R] \mid R\varphi$$

is denoted Π, whilst $\Pi_{flat} \subseteq \Pi$ denotes the set of terms that do not contain refinement operators. Furthermore we define

$$\begin{aligned} L(P\varphi) &:= \varphi(L(P)) \\ S(P\varphi) &:= \varphi(S(P)) \\ red(P\varphi) &:= red(P)\varphi \ . \end{aligned}$$

The well-formedness relation $\vdash$ can be easily defined also over terms having the renaming operator as top operator in their abstract syntax tree:

$$\frac{\vdash P}{\vdash P\varphi}$$

The denotational semantics for the renaming operator can be given easily, as well :

$$[\![P\varphi]\!] := [\![P]\!]\varphi$$

where the semantic operation is defined as follows.

6.2 Definition. Let $\mathcal{E} \in \mathbf{E}$; let φ: $Act \to Act$ be a total function. The *renaming of $\mathcal{E}$ according to φ* is defined by

$$\mathcal{E}\varphi := \langle E_{\mathcal{E}}, \prec_{\mathcal{E}}, \#_{\mathcal{E}}, \varphi \circ \ell_{\mathcal{E}} \rangle \ .$$

Given a term $P[a \rightsquigarrow Q] \in \Sigma$ which is not well-formed, we can always find an injective renaming function ψ: $Act \to Act$ such that $\psi(L(Q)) \cap L(P) = \emptyset$. Hence, $P[a \rightsquigarrow Q]$ can be replaced by the well-formed term $(P[a \rightsquigarrow Q\psi])\psi^{-1} \in \Pi$. This is stated formally in the following theorem, which is the main result of the present section.

6.3 Theorem. Let $P[a \rightsquigarrow Q] \in \Sigma$ such that $\nvdash P[a \rightsquigarrow Q]$.
There always exists an injective renaming ψ such that

1. $\vdash (P[a \rightsquigarrow Q\psi])\psi^{-1}$, and
2. $P[a \rightsquigarrow Q] \ \cong_e \ (P[a \rightsquigarrow Q\psi])\psi^{-1}$

Proof.

1. The choice of a suitable ψ is always possible because Act is infinite and $L(P)$ is finite.
2. According to Definitions 6.2 and 2.6, the isomorphism is the identity function. □

We will call a function ψ *suitable* if it satisfies the conditions of the theorem above. The theorem then justifies the introduction of an auxiliary function $wf\colon \Sigma \to \Pi$ which transforms any term $P \in \Sigma$ into an equivalent, *well-formed* term $R \in \Pi$. Function wf is defined by structural induction as follows:

$$\begin{array}{rcl} wf(a) & := & a \\ wf(P * Q) & := & wf(P) * wf(Q) \quad \text{where } * \in \{ +, ;, \|_A \} \\ wf(P[a \rightsquigarrow Q]) & := & (wf(P)[a \rightsquigarrow wf(Q\psi)])\psi^{-1} \\ wf(P\varphi) & := & wf(P)\varphi \end{array}$$

where ψ should be a suitable renaming. The remainder of the section is devoted to extend the definitions and results of the paper, especially Corollaries 4.4 and 5.12, to cope also with the renaming operators. So, we firstly define syntactic substitution and then we prove that the main theorems can be trivially lifted to the present case.
Before defining how syntactic substitution applies to renaming terms $(P\varphi)\left\{{}^{Q}/_{a}\right\}$, we need to introduce an obvious property, which may be proved by structural induction, stating that the order of substitution applications is inessential.

6.4 Proposition. Let $P, Q \in \Sigma_{flat}$ such that $L(P) \cap L(Q) = \emptyset$. Then, the following holds:

$$\left(P\left\{{}^{Q}/_{a}\right\}\right)\left\{{}^{Q}/_{b}\right\} = \left(P\left\{{}^{Q}/_{b}\right\}\right)\left\{{}^{Q}/_{a}\right\} \ ,$$

provided that $a, b \notin L(Q)$. □

A consequence of this property is that, under the hypothesis above, we can safely use the following shorthand: given a set $A = \{a_1, \ldots, a_n\}$, $A \cap L(Q) = \emptyset$, let

$$P\left\{{}^{Q}/_{A}\right\} := \left(\ldots\left(P\left\{{}^{Q}/_{a_1}\right\}\right)\ldots\left\{{}^{Q}/_{a_n}\right\}\right) \ .$$

Now we can define syntactic substitution as follows: given $P, Q \in \Pi_{flat}$ and an injective renaming function ψ such that $L(P) \cap L(Q\psi) = \emptyset$, we have

$$(P\varphi)\left\{{}^{Q}/_{a}\right\} = \left(P\left\{{}^{Q\psi}/_{A}\right\}\right)(\psi^{-1} \circ \varphi) \ ,$$

where $A = \varphi^{-1}(a) \cap L(P)$.
A similar shorthand can also be introduced for action refinement. Given a set $A = \{a_1, \ldots, a_n\}$, $A \cap L(Q) = \emptyset$, let us define the following useful abbreviation

$$P[A \rightsquigarrow Q] := (\ldots(P[a_1 \rightsquigarrow Q])\ldots[a_n \rightsquigarrow Q]) \ ,$$

which relies upon the property below.

6.5 Proposition. Let $P, Q \in \Pi_{flat}$ such that $L(P) \cap L(Q) = \emptyset$. Then, the following holds:

$$(P[a \rightsquigarrow Q])[b \rightsquigarrow Q] \cong_e (P[b \rightsquigarrow Q])[a \rightsquigarrow Q] \ .$$

provided that $a, b \notin L(Q)$. □

Hence, we can feel free to write $\vdash P[A \rightsquigarrow Q]$ whenever its defining term is well-formed. Finally we can extend Lemma 4.1 to renamings.

6.6 Lemma. Let $P, Q \in \Pi_{flat}$, $a \in Act$, and let φ be a renaming. Then

$$(P\varphi)[a \rightsquigarrow Q] \cong_e ((P[A \rightsquigarrow Q\psi])(\psi^{-1} \circ \varphi) \ ,$$

where $A = \varphi^{-1}(a) \cap L(P)$ and ψ is a renaming such that $L(P) \cap L(Q\psi) = \emptyset$.

Proof. The isomorphism is the identity function. Note that $P[A \rightsquigarrow Q\psi]$ is well-defined because $A \cap L(Q\psi) = \emptyset$ by construction. □

Note that if P and Q are well-formed, so is $(P[A \rightsquigarrow Q\psi])(\psi^{-1} \circ \varphi)$.
The final result is that any term $P \in \Sigma$, be it well-formed or not, is equivalent to a well-formed term $R \in \Pi_{flat}$. This extends Corollaries 4.4 and 5.12.

6.7 Theorem. Let $P \in \Sigma$. The following hold:

1. $\vdash_i P \implies P \cong_e red(wf(P))$;
2. $\vdash_{red} P \implies P \cong_c red(wf(P))$.

7 Conclusion

We have compared notions of syntactic substitution and semantic refinement, the latter of which is interpreted as a form of substitution as well, albeit on a semantic domain. In particular we have investigated conditions under which the two notions give rise to the same semantics, or in other words, refinement operators can be *removed* from terms by repeated syntactic substitution. It turns out that as long as we do not refine synchronizing actions, the correspondence can be established under only mild assumptions on the alphabets, which can furthermore be done away with at the cost of allowing a *renaming* operator in the language. If we do allow synchronization actions to be refined, the the correspondence is less straightforward. For this case we establish necessary and sufficient semantic properties for the distribution of refinement over synchronization, and sufficient syntactic conditions under which refinement can be removed completely.

One of the parameters in our comparison is the equivalence relation being considered. Initially we work with isomorphism of flow event structures; for the refinement of synchronizing actions this turns out to distinguish more terms than we want, and we move to a slightly weaker but still quite strong equivalence: isomorphism of the underlying configuration structures. The necessity of our semantic conditions for distributing refinement over synchronization is relative to this semantics: it may be expected to disappear in weaker semantics. For instance, in configuration structure isomorphism we have the following inequivalence:

$$a + a + a + a \not\cong_c a + a$$

with the consequence

$$(b \parallel_{\{b\}} b)[a \rightsquigarrow a + a] \not\cong (b[b \rightsquigarrow a + a]) \parallel_{\{a\}} (b[b \rightsquigarrow a + a]) \ .$$

In fact this instance of distribution is ruled out by our conditions (Theorem 5.5) because the refinement term $a + a$ is not deterministic, and hence not atomic. However, there are many partial order bisimulation relations weaker than $\cong_c$, for instance *history preserving bisimulation* [16], which equates $P + P$ and P and hence also $a + a + a + a$ and $a + a$; hence under such a relation our conditions are no longer necessary. For instance, as the above example shows, the side condition of determinism may be removed from the property of atomicity. We conjecture that a general way to relax the conditions would be to define

Q is deterministic if	Q	$\cong$	$Q \parallel_{L(Q)} Q$
Q is distinct if	$Q; Q$	$\cong$	$Q; Q \parallel_{L(Q)} (Q \parallel_\emptyset Q)$
Q is atomic if	$(Q \parallel_\emptyset Q) + (Q \parallel_\emptyset Q)$	$\cong$	$(Q \parallel_\emptyset Q) \parallel_{L(Q)} (Q \parallel_\emptyset Q)$.

On the other hand, the syntactic conditions we develop, which are sufficient to guarantee the correspondence of refinement to syntactic substitution, will obviously remain sufficient when the equivalence relation is relaxed.

Dealing with recursion One of the natural extensions of this paper is to add recursion to the language. We will briefly sketch the principle of removing refinement from terms in such an extended language. We assume that recursion is specified in the form of a system θ of process equations over a set of process variables Var ranged over by X, Y. Let the definition of every $X \in Var$ be given by P_X, which range over the language Σ_{Var} defined by

$$P ::= a \mid X \mid P + P \mid P; P \mid P \parallel_A P \mid P[a \rightsquigarrow P]$$

where $X \in Var$, i.e. Σ extended with process variables. (Note that this means we do *not* follow [3] in considering recursion to be a special kind of refinement.) Hence we have the following vector equation:

$$\vec{X} =_\theta \vec{P_X} \ .$$

For the moment we assume every P_X to be flat. Furthermore assume that a set $L_X \subseteq Act$ is assigned to every $X \in Var$, corresponding to the minimal solution of the derived system of equations over $\mathbf{2}^{Act}$, given by

$$\vec{L_X} = L(\vec{P_X}) \ .$$

This minimal solution may be obtained by standard approximation principles since all our operators are continuous on the alphabets. The same principle can be applied to define S_X, I_X etc. as the minimal solutions of the equations for the corresponding set-valued functions. For the predicates $\vdash_{det}$, $\vdash_{dis}$ and $\vdash_{red}$ on the other hand it might be more reasonable to use the *maximal* fixpoint. Likewise, the semantics of the variables $X \in Var$ is assumed to be defined as the minimal solution (with respect to some consistently complete ordering) of the set of equations

$$[\![\vec{X}]\!] = [\![\vec{P_X}]\!] \ .$$

The semantics of the full language Σ_{Var} is defined compositionally. Now for a given refinement operator $\cdot[a \rightsquigarrow Q]$ we extend Var with derived variables X_a^Q for all $X \in Var$ and we extend θ accordingly:

$$X_a^Q =_\theta P_X[a \rightsquigarrow Q] \ .$$

It automatically follows that $X[a \rightsquigarrow Q] \cong X_a^Q$ in this extended system of equations (where $\cong$ is the equivalence relation being considered). Therefore we can extend syntactic substitution to process variables with the semantics-preserving rule

$$X\left\{{}^{Q}/_{a}\right\} := X_a^Q \ .$$

As a result, if $P[a \rightsquigarrow Q]$ is reducible where P is a flat term then $P\left\{{}^{Q}/_{a}\right\} \cong P[a \rightsquigarrow Q]$. (To prove this, it is necessary to extend the proofs of Sections 4 and 5 concerning reducibility to infinite flow event structures and configuration structures.) We can then also reduce terms with nested refinements, by removing the refinement operators from the inside out as before.

However, this is only half of the solution since the problem has been moved to the equations for the X_a^Q, which are not flat. Fortunately however we can apply the above idea also within systems of equations, without getting into an infinite regression of introducing new variables. Assume that θ as a whole is reducible in the sense that $\vdash_{red} P_X[a \rightsquigarrow Q]$ for all $X \in Var$. According to the

reasoning above it follows that $X_a^Q \cong P_X\left\{{}^{Q}/_{a}\right\}$ for all $X \in Var$. Now we can define a new system of equations θ_a^Q over $Var_a^Q := \{ X_a^Q \mid X \in Var \}$, as follows:

$$\vec{X}_a^Q =_{\theta_a^Q} \vec{P_X}\left\{{}^{Q}/_{a}\right\} \ .$$

It should be clear that θ_a^Q is a flat system of equations such that every solution for Var_a^Q in θ is also a solution of θ_a^Q. The burden of the proof is however to show that every solution of θ_a^Q is also a solution of θ. This requires a semantics for recursion and is outside the scope of this paper.

7.1 Example. Consider $Var = \{X\}$ and assume

$$X =_\theta a \parallel_{\{a\}} b; X + c; (a \parallel_\emptyset a) \ .$$

Intuitively X can do any number of b actions, followed by $c; (a+a)$ after which it deadlocks. It can be deduced that $L_X = \{a, b, c\}$, $S_X = \{a\}$, $I_X = \{b, c\}$, $SH_X = D_X = \{a\}$, $SD_X = \emptyset$. Now let $P := a; b + X$. It follows that $P[a \rightsquigarrow d; d]$ is not reducible whereas $\vdash_{red} P[a \rightsquigarrow d; e]$; furthermore $red(P) = d; e; b + X_a^{d;e}$ where

$$X_a^{d;e} =_{\theta_a^{d;e}} d; e \parallel_{\{d,e\}} b; X_a^{d;e} + c; (d; e \parallel_\emptyset d; e) \ .$$

According to the same intuition as above, $X_a^{d;e}$ can do any number of b's, then $c; (d; e + d; e)$ whereafter it deadlocks. Hence it appears that indeed $red(P[a \rightsquigarrow d; e]) \cong_c P[a \rightsquigarrow d; e]$.

We have started by assuming θ to be flat. The same approach can however be used to remove all refinements from an arbitrary system of equations. Hence essentially without changing the theory of reducibility we can extend the results of this paper to a language with recursion.

Related work The work in [15] can be considered as a forerunner of the present research; there a process algebra with refinement (but without communication) is given a linear-time, causality based semantics, and syntactic substitution is proved to agree with the semantic operator.

The problem of relating the two approaches is taken in the opposite direction in [13]: syntactic substitution, without any limitations, is taken as the starting point and the emphasis is on finding a sensible semantic operation which coincides with it. It turns out that a combination of syntactic refinement and *self-synchronization* is enough to achieve this.

Syntactic refinement has also been investigated in depth in [1, 2]; the latter paper combines it with (CCS) synchronization. There is however no notion of semantic refinement, and consequently the relation between the two approaches is not considered. Indeed, [2] allows refinements which would contradict the commutativity of diagram (1) above under common interpretations of semantic refinement. Consider for instance the following CCS variation of Example 5.6: let $P := (a; c \mid a; c) \mid \overline{a}$ and let ρ: $[a \rightsquigarrow b; b]$ a refinement function (mapping the complement of a to the complement of $\rho(a)$). In the execution of P, the action c is always performed; however this is not the case in the execution of

$$P\rho \simeq P\left\{{}^{\rho(a)}/_{a}\right\} = (b; b; c \mid b; b; c) \mid \overline{b}; \overline{b} \ .$$

On the other hand, in a CCS setting, such as that of [1, 2], our choice for the distribution rule may be questioned. As seen in those papers, one may choose to take advantage the inherent asymmetry of the barred and unbarred versions of every action by refining those versions differently, i.e. such that the refinements of a and $\overline{a}$ are defined independently. The main requirement is then that the

synchronization of those refinements satisfies certain constraints. In our setting this idea could be implemented by a rule of the form

$$(P_1 \parallel_A P_2)[a \rightsquigarrow Q] \cong (P_1[a \rightsquigarrow Q_1]) \parallel_{(A\setminus\{a\})\cup A'} (P_2[a \rightsquigarrow Q_2])$$

where Q_1, Q_2 and A' are such that $Q_1 \parallel_{A'} Q_2 \cong Q$. There is however no obvious notion of syntactic substitution which coincides with this.

In [10], a language similar to ours, with essentially the same denotational flow event structure semantics, is considered. There the emphasis in on finding an SOS operational semantics agreeing with the denotational one, up to history preserving bisimulation. Also our paper can be examined in this perspective. Indeed, syntactic substitution provides a simple sound and complete —with some limitations— implementation technique for semantic action refinement up to isomorphism of (configuration or) flow event structures; the operational semantics of a term $P[a \rightsquigarrow Q]$ is the transition system with initial state $red(P)\left\{{}^{Q}/_{a}\right\}$, which, being flat, can be dealt with in a standard way.

We would also like to mention the approach documented in [3] in which the set of refinable symbols and synchronizable actions are explicitly kept disjoint. This means that if $P[a \rightsquigarrow Q]$ is a term then a can never be synchronized within P, and our well-formedness criteria are always fulfilled. Hence in this approach, (1) always commutes: syntactic and semantic refinement *always* coincide.

A Proofs

A.1 Proofs of Section 3

A.1 Lemma. If $P, Q \in \Sigma_{\mathit{flat}}$ such that $L(P) \cap L(Q) = \emptyset$ and $a \in Act$ then

$$L\left(P\left\{{}^{Q}/_{a}\right\}\right) = \begin{cases} (L(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in L(P) \\ L(P) & \text{otherwise} \end{cases}$$

$$S\left(P\left\{{}^{Q}/_{a}\right\}\right) = \begin{cases} (S(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in S(P) \\ S(P) \cup S(Q) & \text{if } a \in L(P) \setminus S(P) \\ S(P) & \text{otherwise.} \end{cases}$$

Proof. By induction on the structure of P.

Actions. If $P = b$ then

$$\begin{aligned} L\left(P\left\{{}^{Q}/_{a}\right\}\right) &= \begin{cases} L(Q) & \text{if } a = b \\ L(P) & \text{otherwise} \end{cases} \\ &= \begin{cases} (L(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in L(P) = \{b\} \\ L(P) & \text{otherwise.} \end{cases} \end{aligned}$$

$$\begin{aligned} S\left(P\left\{{}^{Q}/_{a}\right\}\right) &= \begin{cases} S(Q) & \text{if } a = b \\ S(P) & \text{otherwise} \end{cases} \\ &= \begin{cases} (S(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in S(P) = \emptyset \\ S(P) \cup S(Q) & \text{if } a \in L(P) \setminus S(P) = \{b\} \\ S(P) & \text{otherwise.} \end{cases} \end{aligned}$$

Choice and sequential composition. Straightforward.

Synchronization. To minimize the number of brackets we will use L_P to denote $L(P)$ etc, and $A \setminus a$ to denote $A \setminus \{a\}$ etc. If $P = P_1 \parallel_A P_2$ then

$$\begin{aligned} L\left(P\{{}^{Q}/_{a}\}\right) &= \begin{cases} L_{P_1\{{}^{Q}/_{a}\}} \cup L_{P_2\{{}^{Q}/_{a}\}} \cup (A \setminus a) \cup L_Q & \text{if } a \in A \\ L_{P_1\{{}^{Q}/_{a}\}} \cup L_{P_2\{{}^{Q}/_{a}\}} \cup A & \text{otherwise} \end{cases} \\ &= \begin{cases} ((L_{P_1} \setminus a) \cup L_Q) \cup ((L_{P_2} \setminus a) \cup L_Q) \cup (A \setminus a) \cup L_Q & \text{if } a \in A \cap L_{P_1} \cap L_{P_2} \\ ((L_{P_1} \setminus a) \cup L_Q) \cup L_{P_2} \cup (A \setminus a) \cup L_Q & \text{if } a \in (A \cap L_{P_2}) \setminus L_{P_1} \\ L_{P_1} \cup ((L_{P_2} \setminus a) \cup L_Q) \cup (A \setminus a) \cup L_Q & \text{if } a \in (A \cap L_{P_2}) \setminus L_{P_2} \\ L_{P_1} \cup L_{P_2} \cup (A \setminus a) \cup L_Q & \text{if } a \in A \setminus (L_{P_1} \cup L_{P_2}) \\ ((L_{P_1} \setminus a) \cup L_Q) \cup ((L_{P_2} \setminus a) \cup L_Q) \cup A & \text{if } a \in (L_{P_1} \cap L_{P_2}) \setminus A \\ ((L_{P_1} \setminus a) \cup L_Q) \cup L_{P_2} \cup A & \text{if } a \in L_{P_2} \setminus (L_{P_1} \cup A) \\ L_{P_1} \cup ((L_{P_2} \setminus a) \cup L_Q) \cup A & \text{if } a \in L_{P_1} \setminus (L_{P_2} \cup A) \\ L_{P_1} \cup L_{P_2} \cup A & \text{otherwise} \end{cases} \\ &= \begin{cases} \big((L_{P_1} \cup L_{P_2} \cup A) \setminus a\big) \cup L_Q & \text{if } a \in A \cup L_{P_1} \cup L_{P_2} \\ L_{P_1} \cup L_{P_2} \cup A & \text{otherwise} \end{cases} \\ &= \begin{cases} (L(P) \setminus a) \cup L(Q) & \text{if } a \in L(P) \\ L(P) & \text{otherwise} \end{cases} \end{aligned}$$

The second equality is by induction. In the proof for S we skip one step.

$$S\left(P\{{}^{Q}/_{a}\}\right) = \begin{cases} S_{P_1\{{}^{Q}/_{a}\}} \cup S_{P_2\{{}^{Q}/_{a}\}} \cup \big(\big(L_{P_1\{{}^{Q}/_{a}\}} \cup L_{P_2\{{}^{Q}/_{a}\}}\big) \cap ((A \setminus a) \cup L_Q)\big) & \text{if } a \in A \\ S_{P_1\{{}^{Q}/_{a}\}} \cup S_{P_2\{{}^{Q}/_{a}\}} \cup \big(\big(L_{P_1\{{}^{Q}/_{a}\}} \cup L_{P_2\{{}^{Q}/_{a}\}}\big) \cap A\big) & \text{otherwise} \end{cases}$$

$$= \begin{cases} \big((S_{P_1} \cup S_{P_2} \cup ((L_{P_1} \cup L_{P_2}) \cap A)) \setminus a\big) \cup L_Q & \text{if } a \in A \cup S_{P_1} \cup S_{P_2} \\ S_{P_1} \cup S_{P_2} \cup ((L_{P_1} \cup L_{P_2}) \cap A) \cup S_Q & \text{if } a \in (L_{P_1} \setminus S_{P_1}) \cup (L_{P_2} \setminus S_{P_2}) \\ S_{P_1} \cup S_{P_2} \cup ((L_{P_1} \cup L_{P_2}) \cap A) & \text{otherwise} \end{cases}$$

$$= \begin{cases} (S(P) \setminus a) \cup L(Q) & \text{if } a \in S(P) \\ S(P) \cup S(Q) & \text{if } a \in L(P) \setminus S(P) \\ S(P) & \text{otherwise.} \end{cases}$$

□

A.2 Proof of the main theorem

First of all, we need characterizations of synchronization and refinement directly as operations on configuration structures. The following definitions are inspired by resp. [7] and [19]. Some more notation first: if $F \subseteq (E \cup \{*\}) \times (E \cup \{*\})$ then

$$\begin{aligned} \pi_i(F) &:= \{\, e \mid \exists (e_1, e_2) \in F.\, e_i = e \neq * \,\} \text{ for } i = 1, 2 \\ F(d) &:= \{\, e \mid (d, e) \in F \,\} \ . \end{aligned}$$

The latter regards F as a binary relation over $E \cup \{*\}$ and extends the notion of *image* of d from functions to this type of relations. It is used in *refinement* (see below), such that if d is a refined event then $F(d)$ is the configuration into which it is refined.

A.2 Definition. Let $\langle C_i, \sqrt{}_i, \ell_i \rangle$ be configuration structures for $i = 1, 2$ and $A \subseteq Act$.
The *synchronization of C_1 and C_2 over A* is given by $C_1 \parallel_A C_2 := \langle C, \sqrt{}, \ell \rangle$ such that

- $E_C \subseteq \{\, (e, *) \mid \ell_1(e) \notin A \,\} \cup \{\, (*, e) \mid \ell_2(e) \notin A \,\} \cup \{\, (d, e) \mid \ell_1(d) = \ell_2(e) \in A \,\}$;
- $\emptyset \in C$; and if $F \in C$ and $(e_1, e_2) \in \mathbf{E} \setminus F$ then $F \cup \{(e_1, e_2)\} \in C$ if and only if for both $i = 1, 2$, either $e_i = *$ or $\pi_i(F) \neq \pi_i(F) \cup \{e_i\} \in C_i$;
- $F\sqrt{}$ if and only if $F \in C$ and $\pi_i(F)\sqrt{}_i$ for both $i = 1, 2$;
- $\ell(e_1, e_2) = \begin{cases} \ell_1(e_1) & \text{if } e_2 = *; \\ \ell_2(e_2) & \text{otherwise.} \end{cases}$

Note that the first condition does not characterize E_C completely: there may be synchronization events that are prevented from ever occurring. The following can be proved by induction.

A.3 Proposition. Let $C_1, C_2 \in \mathbf{C}$. If $C := C_1 \parallel_A C_2$ then for all $F \in C$, π_i is injective on F and $\pi_i(F) \in C_i$ for $i = 1, 2$.

Proof. Straightforward by induction on $|P|$. □

A.4 Proposition. Let $C_1, C_2 \in \mathbf{C}$ and $C := C_1 \parallel_A C_2$. If $F \in C$ and $\pi_i(F) \setminus \{e_i\} \in C_i$ for $i = 1, 2$ then $F \setminus \{(e_1, e_2)\} \in C$.

Proof. By induction on $|F|$.

base case If $|F| = 0$ there are no such e_i; if $|F| = 1$ the property is trivial.

induction step Assume the lemma holds whenever $|F| = n \geq 1$; now let $F \in C$ be such that $|F| = n + 1$, and define $F_i := \pi_i(F)$ for $i = 1, 2$. Let $G \in C$ be such that $G \xrightarrow{(d_1, d_2)} F$; define $G_i := \pi_i(G)$ for $i = 1, 2$. Because both π_i are injective on F (Proposition A.3) it follows that $d_i = e_i$ for either $i = 1$ or $i = 2$ implies $(d_1, d_2) = (e_1, e_2)$, in which case the result is trivial. Now assume $d_i \neq e_i$ for both $i = 1, 2$. Because $F_i \setminus \{e_i\} \cup G_i = F_i \in C_i$ it follows (by definition of stable configuration structures) that $G_i \setminus \{e_i\} = (F_i \setminus \{e_i\}) \cap G_i \in C_i$. Hence by induction $G' := G \setminus \{(e_1, e_2)\} \in C$, and hence $G \setminus \{(e_1, e_2)\} = G' \cup \{(d_1, d_2)\} \in C$ by Definition A.2. □

A.5 Definition. Let C, D be configuration structures and $a \in Act$.

The *refinement of a by D in C* is given by $C[a \rightsquigarrow D] := \langle C, \surd, \ell \rangle$ such that

- $E_C = \{ (d, *) \mid \ell_C(d) \neq a \} \cup \{ (d, e) \in E_C \times E_D \mid \ell_C(d) = a \}$;
- $F \in C$ if and only if $\pi_1(F) \in C$ and for all $(d, e) \in F$, either $e = *$ or $F(d) \in D$ and $\pi_1(F) \setminus \{d\} \notin C \Longrightarrow F(d)\surd_D$;
- $F\surd$ if and only if $\pi_1(F)\surd_C$ and for all $(d, e) \in F$, either $e = *$ or $F(d)\surd_D$.
- $\ell(d, e) = \begin{cases} \ell_C(d) & \text{if } e = *; \\ \ell_D(e) & \text{otherwise.} \end{cases}$

The following states that these operations over configuration structures are exactly the lower-level counterpart of the corresponding operations on flow event structures.

A.6 Proposition. Let $P, Q \in \Sigma$.

$$\begin{array}{rcl} CS[\![P \parallel_A Q]\!] & = & CS[\![P]\!] \parallel_A CS[\![Q]\!] \\ CS[\![P[a \rightsquigarrow Q]]\!] & = & CS[\![P]\!][a \rightsquigarrow CS[\![Q]\!]] \end{array}$$

Proof. See [7, 19]. □

We come to the actual proof. For ease of reference we copy some of the definitions from Section 5. Let

$$\begin{array}{rcl} C & = & CS[\![(P_1 \parallel_A P_2)[a \rightsquigarrow Q]]\!] \\ D & = & CS[\![P_1[a \rightsquigarrow Q] \parallel_{(A \setminus \{a\}) \cup L(Q)} P_2[a \rightsquigarrow Q]]\!] \end{array}$$

then $E_C \subseteq X := \bigcup_{1 \leq i \leq 4} X_i$ and $E_D \subseteq Y := \bigcup_{1 \leq i \leq 4} Y_i$ where

$$\begin{array}{rcl} X_1 & := & \{ ((d_1, d_2), e) \in ((E_1 \times E_2) \times E_Q) \mid \ell_1(d_1) = \ell_2(d_2) = a \} \\ X_2 & := & \{ ((d_1, d_2), *) \mid d_i \in E_i \wedge \ell_1(d_1) = \ell_2(d_2) \in (A \setminus \{a\}) \} \\ X_3 & := & \{ ((d, *), *) \mid d \in E_1 \wedge \ell_1(d) \notin A \} \\ X_4 & := & \{ ((*, d), *) \mid d \in E_2 \wedge \ell_2(d) \notin A \} \end{array}$$

$$\begin{array}{rcl} Y_1 & := & \{ ((d_1, e_1), (d_2, e_2)) \mid \ell_1(d_1) = \ell_2(d_2) = a \wedge \ell_Q(e_1) = \ell_Q(e_2) \} \\ Y_2 & := & \{ ((d_1, *), (d_2, *)) \mid d_i \in E_i \wedge \ell_1(d_1) = \ell_2(d_2) \in (A \setminus \{a\}) \} \\ Y_3 & := & \{ ((d, *), *) \mid d \in E_1 \wedge \ell_1(d) \notin A \} \\ Y_4 & := & \{ (*, (d, *)) \mid d \in E_2 \wedge \ell_2(d) \notin A \} \ . \end{array}$$

$f\colon E_C \to E_D$ is defined as the restriction of the union $\bigcup_{1 \leq i \leq 4} f_i$ to E_C where $f_i\colon X_i \to Y_i$ are defined as follows:

$$\begin{array}{rrcl} f_1\colon & ((d_1, d_2), e) & \mapsto & ((d_1, e), (d_2, e)) \\ f_2\colon & ((d_1, d_2), *) & \mapsto & ((d_1, *), (d_2, *)) \\ f_3\colon & ((d, *), *) & \mapsto & ((d, *), *) \\ f_4\colon & ((*, d), *) & \mapsto & (*, (d, *)) \ . \end{array}$$

We present one auxiliary lemma, which holds by definition of f.

A.7 Lemma. If $F \subseteq X$ then

$$\forall (d_1, d_2) \in \pi_1(F).\ F(d_1, d_2) = \pi_1(f(F))(d_1) = \pi_2(f(F))(d_2) \ .$$

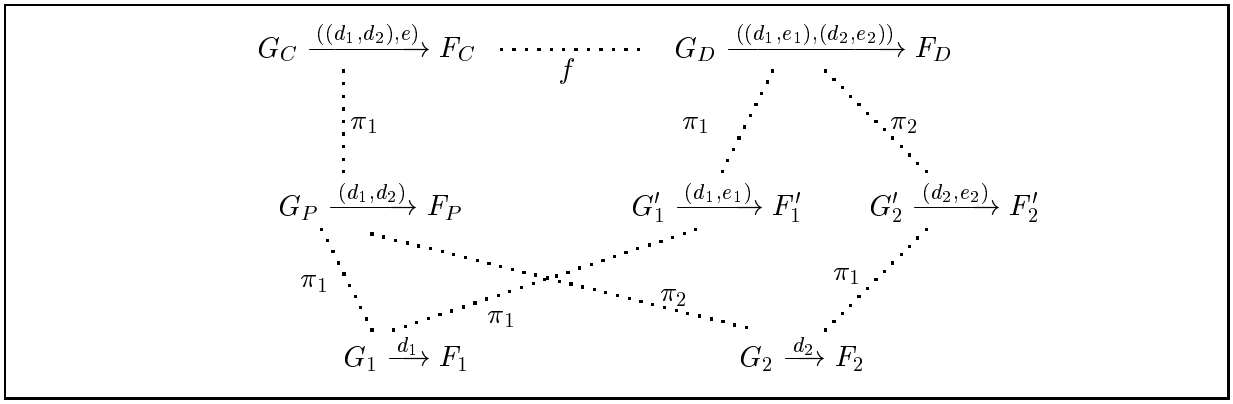

Figure 1: Names and connections used in the proofs

Some (abuse of) notation: $CS[\![P_i]\!] = \langle P_i, \surd_i, \ell_i \rangle$ and

$$\underbrace{\overbrace{(P_1 \parallel_A P_2)}^{\langle P,\surd_P,\ell_P\rangle}[a \rightsquigarrow \overbrace{Q}^{\langle Q,\surd_Q,\ell_Q\rangle}]}_{C} \qquad \underbrace{\overbrace{P_1[a \rightsquigarrow Q]}^{\langle P_1',\surd_1',\ell_1'\rangle} \parallel_{(A\setminus\{a\})\cup L(Q)} \overbrace{P_2[a \rightsquigarrow Q]}^{\langle P_2',\surd_2',\ell_2'\rangle}}_{D}$$

Moreover, if we have $G_C \xrightarrow{a}_C F_C$ then we denote

$$\begin{array}{rcll} F_P & := & \pi_1(F_C) & (\in P) \\ F_i & := & \pi_i(F_P) & (\in P_i) \\ F_D & := & f(F_C) & \\ F_i' & := & \pi_i(F_D) & \\ \\ G_P & := & \pi_1(G_C) & (\in P) \\ G_i & := & \pi_i(G_P) & (\in P_i) \\ G_D & := & f(G_C) & (\in D) \\ G_i' & := & \pi_i(G_D) & (\in P_i') \ . \end{array}$$

Note that by construction, $F_i = \pi_1(F_i')$ and $G_i = \pi_1(G_i')$ for $i = 1, 2$. The $\in$-relations between brackets partly follow from Proposition A.3, and partly from Definition A.5. Figure 1 may be of use in keeping track of the various definitions. Now we prove the points of the proof strategy of Theorem 5.5 (page 13).

A.8 Lemma. $F_C \in C \Longrightarrow f(F_C) \in D$.

Proof. By induction on the size of F.

zero step If $|F| = 0$ then $f(F) = \emptyset \in D$.

induction step Assume that $F \in C \Longrightarrow f(F) \in D$ for all F such that $|F| = n$. Now let $F_C \in C$ such that $|F_C| = n + 1$. From Definition 2.7 it follows that there is a configuration $G_C \in C$ such that $G_C \subset F_C$ and $|G_C| = n$. By the induction hypothesis it follows that $f(G_C) \in D$.

First we prove $F_i' \in P_i'$ for both $i = 1, 2$. By construction $\pi_1(F_i') = F_i \in P_i$. On the other hand, if $(d, e) \in F_i'$ then there exists exactly one $(d_1, d_2) \in \pi_1(F_C)$ such that $d_i = d$, because π_i is injective on F_P for all $F_P \in P$ (see Proposition A.3). By construction of f, it follows that $((d_1, d_2), e) \in F_C \iff (d_i, e) \in F_i'$ and hence $F_C(d_1, d_2) = F_i'(d_i)$; hence either $e = *$ or $F_i'(d_i) \in F_Q$, and furthermore if $F_i \setminus \{d_i\} \notin P_i$ then also $F_P \setminus \{(d_1, d_2)\} \notin P$, which proves $\neg F_C(d_1, d_2)\surd_Q$, hence $\neg F_i'(d)\surd_Q$. It follows by Definition A.5 that $F_i' \in P_i'$.

Next we prove $F_D \in D$. Let $((d_1, d_2), e)$ be the event uniquely determined by $((d_1, d_2), e) \in F_C \setminus G_C$. It follows that $d_i = *$ or $d_i \in F_i \setminus G_i$ for $i = 1, 2$. We use case distinction based on the index k in $((d_1, d_2), e) \in X_k$:

$k = 1, 2$ It follows that $f((d_1, d_2), e) = ((d_1, e), (d_2, e))$; hence $(d_i, e) \notin G'_i$ for $i = 1, 2$, implying $\pi_i(G_D) \neq \pi_i(G_D) \cup \{(d_i, e)\} = G'_i \cup \{(d_i, e)\} = F'_i \in P'_i$.

$k = 3$ It follows that $d_2 = *$ and $f((d_1, d_2), e) = ((d_1, e), *)$; hence $(d_1, e) \notin G'_1$, implying $\pi_1(G_D) \neq \pi_1(G_D) \cup \{(d_1, e)\} = G'_1 \cup \{(d_1, e)\} = F'_1 \in P'_1$.

$k = 4$ Symmetrical to $k = 3$.

In each of these cases it follows by Definition A.2 that $F_D \in D$. This concludes the proof.□

A.9 Lemma. If $P_1 \parallel_A P_2$ and Q satisfy one of the following conditions, then $F_D \in D \Longrightarrow \exists F_C \in C.\ f(F_C) = F_D$.

C1. a is not executed in $P_1 \parallel_A P_2$;

C2. a is two-way sequential in $P_1 \parallel_A P_2$, and Q is deterministic;

C3. a is auto-sequential in $P_1 \parallel_A P_2$, and Q is distinct;

C4. Q is atomic.

Proof. By induction on $|F|$.

zero step If $|F| = 0$ then $F_C = \emptyset$ fulfills the condition.

induction step Assume that $F \in D \Longrightarrow \exists F_C \in C.\ f(F_C) = F$ when $|F| = n$. Now let $F_D \in D$ such that $|F_D| = n+1$. From Definition 2.7 it follows that there is a configuration $G_D \subset F_D$ such that $|G| = n$. By the induction hypothesis it follows that $G_D = f(G_C)$ for some $G_C \in C$. Hence the only objects of Figure 1 that we do not have are $F_P \in P$ and F_C such that $\pi_1(F_C) = F_P$ and $f(F_C) = F_D$; the proof obligation is to construct those. This is done below on a case-by-case basis, depending on the index k in the equation $F_D \setminus G_D \subseteq Y_k$.

Given F_C such that $f(F_C) = F_D$ and $\pi_1(F_C) \in P$, we can prove $F_C \in C$ as follows. Let $((d_1, d_2), e) \in F_C$ be arbitrary such that $e \neq *$; then for both $i = 1, 2$, $F_C(d_1, d_2) = F'_i(d_i)$ according to Lemma A.7; hence $F_C(d_1, d_2) \in Q$ because $F'_1 \in P'_1$. If $\pi_1(F_C) \setminus \{(d_1, d_2)\} \notin P$ then either $F_1 \setminus \{d_1\} \notin P_1$ or $F_2 \setminus \{d_2\} \notin P_2$ (Proposition A.4), hence $\neg F'_i(d_i)\surd_Q$ for either $i = 1$ or $i = 2$, hence $\neg F_C(d_1, d_2)\surd_Q$. It follows that $F_C \in C$.

$k = 1$ $F_D \setminus G_D = \{((d_1, e_1), (d_2, e_2))\}$ where $d_i \in E_i$, $e_i \in E_Q$ such that $\ell_1(d_1) = \ell_2(d_2) = a$ and $\ell_Q(e_1) = \ell_Q(e_2)$, which is to say that $((d_1, e_1), (d_2, e_2))$ synchronizes two actions from refinements of a, i.e. two actions of Q. Now we define

$$\begin{aligned} F_P &:= G_P \cup \{(d_1, d_2)\} \\ F_C &:= G_C \cup \{((d_1, d_2), e_1)\}\ . \end{aligned}$$

First we prove $F_P \in P$. For this purpose it suffices to prove that either $F_P = G_P$, i.e. $(d_1, d_2) \in G_P$, or $d_i \notin G_i$ for both $i = 1, 2$; the other conditions of Definition A.2 are already taken care of. Hence we assume $d_1 \in G_1$ and $d_2 \notin G_1$ and show that this leads to a contradiction under all conditions Cn for $n = 1, 2, 3, 4$. First note that $d_1 \in G_1$ implies $(d_1, d'_2) \in G_P$ for some d'_2, where $\ell_P(d_1, d'_2) = \ell_1(d_1) = a$.

$n = 1$ $(d_1, d'_2) \in G$ with $\ell_P(d_1, d'_2) = a$ contradicts the assumption that a does not occur in P.

$n = 2$ Because $G'_1(d_1) \xrightarrow{\ell_Q(e_1)} F'_1(d_1)$ it follows that $G'_1(d_1)$ is not maximal in Q, and hence $\neq G'_1(d_1)\surd_Q$; hence also $\neg G'_2(d'_2)\surd_Q$ by Lemma A.7; hence $\neg F'_2(d'_2)\surd_Q$. It follows by Definition A.5 that $H_2 := F_2 \setminus \{d'_2\} \in P_2$. It follows that $H_2 \cup G_2 = F_2 \in P_2$, and hence also $H_2 \cap G_2 \in P_2$ by the definition of stable configuration structures. But this implies

$$\begin{aligned} H_2 \cap G_2 &\xrightarrow{\ell_2(d'_2)}_P H_2 \xrightarrow{\ell_2(d_2)}_P F_2 \\ H_2 \cap G_2 &\xrightarrow{\ell_2(d_2)}_P G_2 \xrightarrow{\ell_2(d'_2)}_P F_2 \ . \end{aligned}$$

This implies that $a = \ell_2(d_2) = \ell_2(d'_2)$ is auto-concurrent in P_2 at H_2; hence a is not two-way sequential in P.

$n = 3$ It follows that $\emptyset \subset G'_1(d_1) \xrightarrow{\ell_Q(e_1)}_Q F'_1(d_1)$, whereas $\emptyset = G'_2(d_2) \xrightarrow{\ell_Q(e_2)}_Q F'_2(d_2)$. Since $\ell_Q(e_1) = \ell_Q(e_2)$ this contradicts the distinctness of Q.

$n = 4$ Since $\emptyset \subset G'_1(d_1) \subset F'_1(d_1)$ it follows that Q is not atomic.

By construction it follows immediately that $\pi_1(F_C) = F_P$. Now we prove $f(F_C) = F_D$. For this we merely need $e_1 = e_2$. $(d_1, d_2) \in F_P \in P$ proves that $\ell_P(d_1, d_2) = a$ occurs in P and hence C1 does not hold. Lemma A.7 implies $G'_1(d_1) = G'_2(d_2)$; from C2–4 we know that Q is deterministic, hence from $G'_i(d_i) \xrightarrow{\ell_Q(e_i)}_Q F'_i(d_i)$ for $i = 1, 2$ and $\ell_Q(e_1) = \ell_Q(e_2)$ it follows that $F'_1(d_1) = F'_2(d_2)$, hence $e_1 = e_2$.

$k = 2$ $F_D \setminus G_D = \{((d_1, *), (d_2, *))\}$ where $d_i \in E_i$ for both $i = 1, 2$, corresponding to the synchronization of two "ordinary," i.e. non-refined events. Let

$$\begin{aligned} F_P &:= G_P \cup \{(d_1, d_2)\} \\ F_C &:= G_C \cup \{((d_1, d_2), *)\} \ . \end{aligned}$$

From the injectivity of π_i on F_D it follows that $(d_i, *) \notin G'_i$, and hence $\pi(G'_i) \neq \pi(G'_i) \cup \{d_i\} = G_i \cup \{d_i\} = F_i \in P_i$; it follows that $F_P \in P$. $\pi_1(F_C) = F_P$ and $f(F_C) = F_D$ are immediate.

$k = 3$ $F_D \setminus G_D = \{((d_1, *), *)\}$ for some $d_1 \in E_1$, corresponding to an unsynchronized event from P'_1. Let

$$\begin{aligned} F_P &:= G_P \cup \{(d_1, *)\} \\ F_C &:= G_C \cup \{((d_1, *), *)\} \ . \end{aligned}$$

From the injectivity of π_1 on F_D it follows that $(d_1, *) \notin G'_1$, and hence $\pi(G'_1) \neq \pi(G'_1) \cup \{d_1\} = G_1 \cup \{d_1\} = F_1 \in P_1$; it follows that $F_P \in P$. $\pi_1(F_C) = F_P$ and $f(F_C) = F_D$ are immediate.

$k = 4$ Symmetrical to $k = 3$. □

A.10 Lemma. $F_C\surd_C \iff f(F_C)\surd_D$.

Proof. Let $F_D := f(F_C)$. The following statements are equivalent:

- $F_C\surd_C$;
- $\pi_1(F_C)\surd_P$ and for all $((d_1, d_2), e) \in F_C$, if $e \neq *$ then $F_C(d_1, d_2)\surd_Q$ (by Definition A.5);
- for $i = 1, 2$, $\pi_1(\pi_i(F_D))\surd_i$ (by Definition A.2 and because $\pi_1(\pi_i(F_D)) = \pi_i(\pi_1(F_C))$) and for all $(d_i, e) \in \pi_i(F_D)$, if $e \neq *$ then $\pi_i(F_D)(d_i)\surd_Q$ (because $F_C(d_1, d_2) = \pi_i(F_D)(()d_i)$);
- for $i = 1, 2$, $\pi_i(F_D)\surd'_i$ (by Definition A.5);

- $F_D\surd_D$ (by Definition A.2). □

A.11 Lemma. Each of the following conditions is sufficient to construct a configuration in D which is not in $f(C)$.

D1. a is executed in $P_1 \parallel_A P_2$ and Q is nondeterministic;

D2. a is not two-way sequential in $P_1 \parallel_A P_2$ and Q is not distinct;

D3. a is autoconcurrent in $P_1 \parallel_A P_2$ and Q is not atomic.

Proof. Some general definitions first. Assume configurations $F_P, G_P \in P$ such that F_P contains no a-labelled event and $F_P \xrightarrow{a} G_P$; let $G_P \setminus F_P := \{(d_1, d_2)\}$. Furthermore let $F_Q \in Q$ be nonempty and let

$$\begin{array}{rcll} H_C & := & (F_P \times \{*\}) \cup (\{(d_1,d_2)\} \times F_Q) & \\ H_D & := & \{\, (e_1', e_2') \mid e_i' \in (\pi_i(F_P) \times \{*\}) \cup (\{d_i\} \times F_Q)\,\} & (= f(H_C)) \\ H_i' & := & (\pi_i(F_P) \times \{*\}) \cup (\{d_i\} \times F_Q) & (= \pi_i(H_D)) \\ H_i & := & \pi_i(F_P) \cup \{d_i\} & (= \pi_1(H_i')) \end{array}$$

The following argument then shows $H_C \in C$. First we show that the events of H_C are elements of X. On the one hand, $\ell_P(e) \neq a$ for all $e \in F_P$, which implies $(F_P \times \{*\}) \subseteq \bigcup_{2 \leq j \leq 4} X_j$. On the other hand, $(\{(d_1,d_2)\} \times F_Q) \subseteq X_1$ because $\ell_P(d_1,d_2) = a$. Now we show the other conditions of Definition A.5. $\pi_1(H_C) = F_P \in P$ if $F_Q = \emptyset$; otherwise $\pi_1(H_C) = F_P \cup \{d\} = G_P \in C$. If $((d_1', d_2'), e) \in H_C$ where $e \neq *$ then $(d_1', d_2') = (d_1, d_2)$, hence $H_C((d_1', d_2')) = F_Q \in Q$ and $G_P \subseteq \{(d_1', d_2')\} = F_P \in P$. It follows that $H_C \in C$, and hence $H_D \in D$, $H_i' \in P_i'$ and $H_i \in P_i$.

D1. Assume that F_P contains no a-labelled events, and that F_Q is such that there exist $e_1 \neq e_2$ with $\ell_Q(e_1) = \ell_Q(e_2) = b$ such that $F_Q \xrightarrow{b}_Q F_Q \cup \{e_1\}$ and $F_Q \xrightarrow{b}_Q F_Q \cup \{e_2\}$. Furthermore let

$$\begin{array}{rcl} K_D & := & H_D \cup \{((d_1,e_1)),(d_2,e_2))\} \\ K_i' & := & \pi_i(K_D) \quad (= H_i' \cup \{(d_i, e_i)\}) \\ K_i & := & \pi_1(K_i') \quad (= \pi_i(G_P)) \end{array}$$

We show $K_D \in D$. $(d_i, e_i) \notin H_i'$ for $i = 1, 2$ because $e_i \notin F_Q$. Furthermore, $K_i' \in P_i'$ because $\pi_1(K_i') = K_i = \pi_i(G_P) \in P_i$, and if $(d_i', e_i') \in K_i'$ such that $e_i' \neq *$ then $d_i' = d_i$ and $K_i'(d_i') = F_Q \cup \{e_i\} \in Q$; moreover $K_i \setminus \{d_i'\} = \pi_i(F_P) \in P_i$. It follows that $((d_1,e_1),(d_2,e_2)) \in E_D \setminus f(E_C)$.

D2. Assume that F_P is minimal such that (i) a is executed in P at F and (ii) a is not auto-sequential in C_i at $F_i := \pi_i(F_P)$, where either $i = 1$ or symmetrically $i = 2$. Assume $i = 1$. It follows that there exists a $d_1' \in E_1$ such that $d_1' \neq d_1$ and $F_1 \xrightarrow{a}_1 F_1 \cup \{d_1'\}$ and $F_1 \cup \{d_1, d_1'\} \in P_1$. Furthermore assume that F_Q is such that $\emptyset \xrightarrow{a}_Q \{e\}$ and $\emptyset \neq F_Q \xrightarrow{a}_Q F_Q \cup \{e'\}$. Finally let

$$\begin{array}{rcl} K_D & := & H_D \cup \{((d_1', e), (d_2, e'))\} \\ K_i' & := & \pi_i(K_D) \\ K_i & := & \pi_1(K_i') \end{array}$$

We show $K_D \in D$. On the one hand, $d_1' \notin F_1$ and $d_1' \neq d_1$, hence $d_1' \notin H_1$, which implies $(d_1', e) \notin H_1'$; moreover $\pi_1(K_1') = F_1 \cup \{d_1, d_1'\} \in P_1$ and if $(d_1'', e) \in K_1'$ such that $e \neq *$ then either $d_1'' = d_1$, in which case $K_1'(d_1'') = F_Q \in Q$ and $K_1 \setminus \{d_1''\} = F_1 \cup \{d_1'\} \in P_1$, or $d_1'' = d_1'$, in which case $K_1'(d_1'') = \{e\} \in Q$ and $K_1 \setminus \{d_1''\} = F_1 \cup \{d_1\} \in P_1$. On the other

hand, $(d_2, e') \notin H_2'$ because $e' \notin F_Q$; moreover $\pi_1(K_2') = K_2 \in P_2$, and if $(d_2', e'') \in K_2'$ such that $e'' \neq *$ then $d_2' = d_2$ and $K_2'(d_2) = F_Q \cup \{e'\} \in Q$ and $K_2 \setminus \{d_2'\} = \pi_2(F_P) \in P_2$.

At the same time, if $K_D \in f(C)$ then $e = e'$ and $K_C := f^{-1}(K_D) \in C$; then $K_P := \pi_1(K_C) \in P$ contains both (d_1, d_2) and (d_1', d_2), hence π_1 is not injective on K_P, which contradicts $K_P \in P$. It follows that $K_D \in D \setminus f(C)$.

D3. Let $F_P, F_P', F_P'' \in P$ be minimal such that $F_P \xrightarrow{a}_1 F_P'$, $F \xrightarrow{a}_1 F_P'' \neq F_P'$ and $F_P' \cup F_P'' \in P$; then $F_P' = F_P \cup \{(d_1, d_2)\}$ and $F_P'' = F_P \cup \{(d_1', d_2')\}$ where $(d_1, d_2) \neq (d_1', d_2')$; because π_i is injective on $F' \cup F''$ this implies $d_i \neq d_i'$ for both $i = 1, 2$. Assume that F_Q is such that $\emptyset \neq F_Q \xrightarrow{a}_Q F_Q \cup \{e\}$. Furthermore let

$$\begin{aligned} K_D &:= H_D \cup \{(d_1', e), (d_2, e)\} \\ K_i' &:= \pi_i(K_D) \\ K_i &:= \pi_1(K_i') \end{aligned}$$

From here one proceeds in exactly the same manner as for D2, proving $K_D \in D \setminus F(C)$. □

A.3 Proofs of the syntactic characterization

5.7 Proposition. Let $P \in \Sigma$ and $a \in Act$.

1. If a is executed in P then $a \in L(P)$;
2. a is initial in P if and only if $a \in I(P)$;
3. If a is auto-concurrent in P then $a \in D(P)$;

Proof. Let $\langle C_P, \surd_P, \ell_P \rangle := CS[\![P]\!]$.

1. Trivial.

2. The property can be reformulated as follows:

$$I(P) = \{\, \ell_P(e) \mid \{e\} \in C_P \,\} \ .$$

The proof is straightforward by induction on the structure of P. We show the case for refinement. Assume $P = P_1[a \rightsquigarrow Q]$.

$$\begin{aligned} \{\, e \mid \{e\} \in C_P \,\} &= \{\, (e, *) \mid \{e\} \in C_{P_1} \wedge \ell_P(e) \neq a \,\} \\ &\quad \cup \{\, (e, d) \mid \{e\} \in C_{P_1} \wedge \ell_{P_1}(e) = a \wedge \{d\} \in C_Q \,\} \end{aligned}$$

and hence

$$\begin{aligned} \{\, \ell(e) \mid \{e\} \in C_P \,\} &= \{\, \ell_{P_1}(e) \mid \{e\} \in C_{P_1} \wedge \ell_{P_1}(e) \neq a \,\} \\ &\quad \cup \{\, \ell_Q(d) \mid \{e\} \in C_{P_1} \wedge \ell_{P_1}(e) = a \wedge \{d\} \in C_Q \,\} \\ &= (I(P_1) \setminus \{a\}) \cup (I(Q) \text{ if } a \in I(P_1)) \\ &= I(P_1[a \rightsquigarrow Q]) \ . \end{aligned}$$

3. By induction on the structure of P. We show the cases of synchronization and refinement.

Synchronization. Assume $P = P_1 \parallel_A P_2$ and let $F \in C_P$ and $e_1 \neq e_2$ be such that $F \xrightarrow{a}_P F \cup \{e_1\}$ and $F \xrightarrow{a} F \cup \{e_2\}$ and $F \cup \{e_1, e_2\} \in C_P$. e_1 and e_2 are of the form (d_1, d_2) where $d_i = *$ implies $d_{3-i} \neq *$. If $\pi_i(e_1) \neq * \neq \pi_i(e_2)$ for some $i = 1, 2$ then $\pi_i(e_1) \neq \pi_i(e_2)$, $\pi_i(F \cup \{e_1, e_2\}) \in C_{P_i}$ and $\pi_i(F) \xrightarrow{a}_{P_i} \pi_i(F \cup \{e_j\})$ for both $j = 1, 2$; hence a is concurrent in P_i and hence $a \in D(P_i)$ by induction.

Now if $a \in A$ then $\pi_i(e_j) \neq *$ for all $i, j \in \{1,2\}$; therefore

$$a \in D(P_1) \cap D(P_2) \cap A \ .$$

On the other hand, if $a \notin A$ then either for some i then $\pi_i(e_1) = \pi_i(e_2) = *$, and hence

$$a \in D(P_{3-i}) \setminus A \ ,$$

or $\pi_1(e_j) = * = \pi_2(e_{3-j})$ for some $j \in \{1,2\}$, hence

$$a \in (L(P_1) \cap L(P_2)) \setminus A \ .$$

In each case $a \in D(P)$.

Refinement. Assume $P = P_1[a \rightsquigarrow Q]$ and let $F \in C_P$ and $e_1 \neq e_2$ be such that $F \xrightarrow{b}_P F \cup \{e_i\}$ for both $i = 1,2$ and $F \cup \{e_1, e_2\} \in C$.

If $b \in L(Q)$ then e_1 and e_2 are events obtained by refinement; hence $\ell_{P_1}(\pi_1(e_i)) = a$ for both $i = 1,2$, implying $a \in L(P_1)$. Either $\pi_1(e_1) = \pi_1(e_2)$, in which case $\pi_2(e_1) \neq \pi_2(e_2)$, implying that b is auto-concurrent in Q and hence $b \in D(Q)$ by induction, or $\pi_1(e_1) \neq \pi_1(e_2)$, implying that a is auto-concurrent in P_1 and hence $a \in D(P_1)$ by induction. Otherwise $b \in L(P_1)$, implying $\pi_2(e_1) = \pi_2(e_2) = *$; hence b is auto-concurrent in P_1, implying $b \in D(P_1)$ by induction. In each case $b \in D(P_1[a \rightsquigarrow Q])$. □

A.12 Lemma. Let $P, Q \in \Sigma_{flat}$ and $a \in Act$.

$$I\left(P\left\{{}^{Q}/_{a}\right\}\right) = \begin{cases} (I(P) \setminus \{a\}) \cup I(Q) & \text{if } a \in I(P) \\ I(P) & \text{otherwise} \end{cases}$$

$$D\left(P\left\{{}^{Q}/_{a}\right\}\right) = \begin{cases} (D(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in D(P) \\ D(P) \cup D(Q) & \text{if } a \in L(P) \setminus D(P) \\ D(P) & \text{otherwise} \end{cases}$$

$$SH\left(P\left\{{}^{Q}/_{a}\right\}\right) = \begin{cases} (SH(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in SH(P) \\ SH(P) \cup D(Q) & \text{if } a \in S(P) \setminus SH(P) \\ SH(P) \cup SH(Q) & \text{if } a \in L(P) \setminus S(P) \\ SH(P) & \text{otherwise} \end{cases}$$

$$SD\left(P\left\{{}^{Q}/_{a}\right\}\right) = \begin{cases} (SD(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in SD(P) \\ SD(P) \cup D(Q) & \text{if } a \in S(P) \setminus SD(P) \\ SD(P) \cup SD(Q) & \text{if } a \in L(P) \setminus S(P) \\ SD(P) & \text{otherwise.} \end{cases}$$

Proof. By induction on the structure of P. We only prove the case of synchronization for I, D and SH; the other cases are analogous. Assume that the lemma holds for P_1 and P_2 and let $P = P_1 \parallel_A P_2$. First assume $a \in A$; then $P\left\{{}^{Q}/_{a}\right\} = P_1\left\{{}^{Q}/_{a}\right\} \parallel_{(A \setminus \{a\}) \cup L(Q)} P_2\left\{{}^{Q}/_{a}\right\}$.
We denote $A' := (A \setminus \{a\}) \cup L(Q)$; also, we write I_P for $I(P)$ etc. to improve the readability by keeping the number of brackets down.

$$\begin{aligned} I\left(P\left\{{}^{Q}/_{a}\right\}\right) &= \left(\left(I_{P_1\{{}^{Q}/_{a}\}} \cup I_{P_2\{{}^{Q}/_{a}\}}\right) \setminus A'\right) \cup \left(I_{P_1\{{}^{Q}/_{a}\}} \cap I_{P_2\{{}^{Q}/_{a}\}} \cap A'\right) \\ &= \begin{cases} ((((I_{P_1} \setminus \{a\}) \cup I_Q) \cup ((I_{P_2} \setminus \{a\}) \cup I_Q)) \setminus A') & \\ \cup\ (((I_{P_1} \setminus \{a\}) \cup I_Q) \cap ((I_{P_2} \setminus \{a\}) \cup I_Q) \cap A') & \text{if } a \in I_{P_1} \cap I_{P_2} \\ ((I_{P_1} \cup ((I_{P_2} \setminus \{a\}) \cup I_Q)) \setminus A') \cup (I_{P_1} \cap ((I_{P_2} \setminus \{a\}) \cup I_Q) \cap A') & \text{if } a \in I_{P_2} \setminus I_{P_1} \\ ((((I_{P_1} \setminus \{a\}) \cup I_Q) \cup I_{P_2}) \setminus A') \cup (((I_{P_1} \setminus \{a\}) \cup I_Q) \cap I_{P_2} \cap A') & \text{if } a \in I_{P_1} \setminus I_{P_2} \\ ((I_{P_1} \cup I_{P_2}) \setminus A') \cup (I_{P_1} \cap I_{P_2} \cap A') & \text{otherwise} \end{cases} \end{aligned}$$

$$= \begin{cases} ((I_{P_1} \cup I_{P_2}) \setminus A) \cup (I_{P_1} \cap I_{P_2} \cap (A \setminus \{a\})) \cup I_Q & \text{if } a \in I_{P_1} \cap I_{P_2} \\ ((I_{P_1} \cup I_{P_2}) \setminus A) \cup (I_{P_1} \cap I_{P_2} \cap A) & \text{otherwise} \end{cases}$$

$$= \begin{cases} I(P) \cup I(Q) & \text{if } a \in I(P) \cap A \\ I(P) & \text{if } a \in A \setminus I(P). \end{cases}$$

The second step is by application of the induction hypothesis. In the derivations below we joint the second and third steps to save space.

$$\begin{aligned} D\left(P\{^{Q}/_{a}\}\right) &= \left(D_{P_1\{^{Q}/_{a}\}} \cup D_{P_2\{^{Q}/_{a}\}} \cup \left(L_{P_1\{^{Q}/_{a}\}} \cap L_{P_2\{^{Q}/_{a}\}}\right)\right) \setminus A' \cup \left(D_{P_1\{^{Q}/_{a}\}} \cap D_{P_2\{^{Q}/_{a}\}} \cap A'\right) \\ &= \begin{cases} ((D_{P_1} \cup D_{P_2} \cup (L_{P_1} \cap L_{P_2}) \setminus A) \cup (D_{P_1} \cap D_{P_2} \cap (A \setminus \{a\})) \cup L_Q \\ \quad \text{if } a \in D_{P_1} \cap D_{P_2} \\ ((D_{P_1} \cup D_{P_2} \cup (L_{P_1} \cap L_{P_2}) \setminus A) \cup (D_{P_1} \cap D_{P_2} \cap (A \setminus \{a\})) \cup D_Q \\ \quad \text{if } a \in (L_{P_1} \setminus D_{P_1}) \cap (L_{P_2} \setminus D_{P_2}) \\ ((D_{P_1} \cup D_{P_2} \cup (L_{P_1} \cap L_{P_2}) \setminus A) \cup (D_{P_1} \cap D_{P_2} \cap A) \\ \quad \text{otherwise} \end{cases} \\ &= \begin{cases} D(P) \cup L(Q) & \text{if } a \in D(P) \cap A \\ D(P) \cup D(Q) & \text{if } a \in (L(P) \cap A) \setminus D(P) \\ D(P) & \text{if } a \in A \setminus L(P). \end{cases} \end{aligned}$$

$$\begin{aligned} SH\left(P\{^{Q}/_{a}\}\right) &= SH_{P_1\{^{Q}/_{a}\}} \cup SH_{P_2\{^{Q}/_{a}\}} \cup \left(\left(D_{P_1\{^{Q}/_{a}\}} \cup D_{P_2\{^{Q}/_{a}\}}\right) \cap A'\right) \\ &= \begin{cases} ((SH_{P_1} \cup SH_{P_2}) \setminus \{a\}) \cup ((D_{P_1} \cup D_{P_2}) \cap (A \setminus \{a\})) \cup L_Q \\ \quad \text{if } a \in S_{P_1} \cup S_{P_2} \cup D_{P_1} \cup D_{P_2} \\ (SH_{P_1} \cup SH_{P_2}) \cup ((D_{P_1} \cup D_{P_2}) \cap A) \cup D_Q \\ \quad \text{if } a \in (S_{P_1} \setminus D_{P_1}) \cup (S_{P_2} \setminus D_{P_2}) \\ (SH_{P_1} \cup SH_{P_2}) \cup ((D_{P_1} \cup D_{P_2}) \cap A) \cup SH_Q \\ \quad \text{if } a \in (L_{P_1} \setminus S_{P_1}) \cup (L_{P_2} \setminus S_{P_2}) \\ (SH_{P_1} \cup SH_{P_2}) \cup ((D_{P_1} \cup D_{P_2}) \cap A) \\ \quad \text{otherwise} \end{cases} \\ &= \begin{cases} (SH(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in SH(P) \cap A \\ SH(P) \cup D(Q) & \text{if } a \in (S(P) \cap A) \setminus SH(P) \\ SH(P) \cup S_1(Q) & \text{if } a \in (L(P) \cap A) \setminus S(P) \\ SH(P) & \text{if } a \in A \setminus L(P). \end{cases} \end{aligned}$$

Now assume $a \notin A$; then $P\left\{^{Q}/_{a}\right\} = P_1\left\{^{Q}/_{a}\right\} \parallel_A P_2\left\{^{Q}/_{a}\right\}$.

$$\begin{aligned} I\left(P\{^{Q}/_{a}\}\right) &= \left(\left(I_{P_1\{^{Q}/_{a}\}} \cup I_{P_2\{^{Q}/_{a}\}}\right) \setminus A\right) \cup \left(I_{P_1\{^{Q}/_{a}\}} \cap I_{P_2\{^{Q}/_{a}\}} \cap A\right) \\ &= \begin{cases} ((I_{P_1} \cup I_{P_2}) \setminus A) \cup I_Q \cup ((I_{P_1} \cap I_{P_2} \cap A) & \text{if } a \in I_{P_1} \cup I_{P_2} \\ ((I_{P_1} \cup I_{P_2}) \setminus A) \cup ((I_{P_1} \cap I_{P_2} \cap A) & \text{otherwise} \end{cases} \\ &= \begin{cases} I(P) \cup I(Q) & \text{if } a \in I(P) \setminus A \\ I(P) & \text{if } a \notin I(P) \cup A. \end{cases} \end{aligned}$$

$$\begin{aligned} D\left(P\{^{Q}/_{a}\}\right) &= \left(D_{P_1\{^{Q}/_{a}\}} \cup D_{P_2\{^{Q}/_{a}\}} \cup \left(L_{P_1\{^{Q}/_{a}\}} \cap L_{P_2\{^{Q}/_{a}\}}\right)\right) \setminus A \\ &\quad \cup \left(D_{P_1\{^{Q}/_{a}\}} \cap D_{P_2\{^{Q}/_{a}\}} \cap A\right) \\ &= \begin{cases} (((D_{P_1} \cup D_{P_2} \cup (L_{P_1} \cap L_{P_2})) \setminus A) \cup (D_{P_1} \cap D_{P_2} \cap A)) \setminus \{a\} \cup L_Q \\ \quad \text{if } a \in D_{P_1} \cup D_{P_2} \cup (L_{P_1} \cap L_{P_2}) \\ ((D_{P_1} \cup D_{P_2} \cup (L_{P_1} \cap L_{P_2})) \setminus A) \cup D_Q \cup (D_{P_1} \cap D_{P_2} \cap A) \\ \quad \text{if } a \in (L_{P_1} \setminus (D_{P_1} \cup L_{P_2}) \cup (L_{P_2} \setminus (D_{P_2} \cup L_{P_1}) \\ ((D_{P_1} \cup D_{P_2} \cup (L_{P_1} \cap L_{P_2})) \setminus A) \cup (D_{P_1} \cap D_{P_2} \cap A) \\ \quad \text{otherwise} \end{cases} \\ &= \begin{cases} (D(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in D(P) \setminus A \\ D(P) \cup D(Q) & \text{if } a \in L(P) \setminus (D(P) \cup A) \\ D(P) & \text{if } a \notin L(P) \cup A. \end{cases} \end{aligned}$$

$$SH\left(P\{^{Q}/_{a}\}\right) = SH_{P_1\{^{Q}/_{a}\}} \cup SH_{P_2\{^{Q}/_{a}\}} \cup \left(\left(D_{P_1\{^{Q}/_{a}\}} \cup D_{P_2\{^{Q}/_{a}\}}\right) \cap A\right)$$

$$= \begin{cases} ((SH_{P_1} \cup SH_{P_2}) \setminus \{a\}) \cup L_Q \cup ((D_{P_1} \cup D_{P_2}) \cap (A \setminus \{a\})) \\ \quad \text{if } a \in S_{P_1} \cup S_{P_2} \\ (SH_{P_1} \cup SH_{P_2}) \cup D_Q \cup ((D_{P_1} \cup D_{P_2}) \cap A) \\ \quad \text{if } a \in (S_{P_1} \setminus SH_{P_1}) \cup (S_{P_2} \setminus SH_{P_2}) \\ (SH_{P_1} \cup SH_{P_2}) \cup SH_Q \cup ((D_{P_1} \cup D_{P_2}) \cap A) \\ \quad \text{if } a \in (L_{P_1} \setminus S_{P_1}) \cup (L_{P_2} \setminus S_{P_2}) \\ (SH_{P_1} \cup SH_{P_2}) \cup ((D_{P_1} \cup D_{P_2}) \cap A) \\ \quad \text{otherwise} \end{cases}$$

$$= \begin{cases} (SH(P) \setminus \{a\}) \cup L(Q) & \text{if } a \in SH(P) \setminus A \\ SH(P) \cup D(Q) & \text{if } a \in S(P) \setminus (SH(P) \cup A) \\ SH(P) \cup SH(Q) & \text{if } a \in L(P) \setminus (S(P) \cup A) \\ SH(P) & \text{if } a \notin A \cup L(P). \end{cases}$$

These two cases together imply the lemma. □

5.9 Proposition. Let $P \in \Sigma$.

1. If $\vdash_{det} P$ then P is deterministic;
2. If $\vdash_{dis} P$ then P is distinct;
3. If $\vdash_{dis} P$ and $L(P) = I(P)$ then P is atomic.

Proof. Let $\langle C_P, \surd_P, \ell_P \rangle := CS[\![P]\!]$.

1. By induction on the structure of P. Assume that $\vdash_{det} P$ and is not deterministic; we will show that this leads to contradictions in each case. There is a configuration $F \in C_P$ and events $e_1 \neq e_2$ such that $F \xrightarrow{a} F \cup \{e_i\}$ for both $i = 1, 2$, for some action $a = \ell_P(e_1) = \ell_P(e_2)$.

   Actions. Assume $P = a$. There can be no such $e_1 \neq e_2$; contradiction.

   Choice. Assume $P = P_1 + P_2$. It follows that $I(P_1) \cap I(P_2) = \emptyset$ and by induction that P_1 and P_2 are deterministic. If $e_1, e_2 \in E_{P_i}$ for some $i \in \{1, 2\}$ then $F \in C_{P_i}$; hence P_i is nondeterministic; contradiction. If $e_1 \in C_{P_i}$ and $e_2 \in C_{P_{3-i}}$ then $F = \emptyset$ and $a \in I(P_1) \cap I(P_2)$ according to Proposition 5.7; contradiction.

   Sequential composition. Assume $P = P_1; P_2$. It follows by induction that P_1 and P_2 are deterministic. If $F \in C_{P_1}$ and $\neg F \surd_{P_1}$ then $F, F \cup \{e_i\} \in C_{P_1}$ for $i = 1, 2$, contradicting the determinism of P_1; otherwise

   $$F \setminus E_{P_1} \xrightarrow{a}_{P_2} (F \cup \{e_i\}) \setminus E_{P_1} \quad ,$$

   contradicting the determinism of P_2.

   Synchronization. Assume $P_1 \|_A P_2$. It follows that $L(P) \cap L(Q) \subseteq A$ and by induction that P_1 and P_2 are deterministic. If $a \in A$ then $\pi_i(F) \xrightarrow{a}_{P_i} \pi_i(F \cup \{e_j\})$ for all $i, j \in \{1, 2\}$; contradiction. Otherwise either $a \in L(P_1)$ or $a \in L(P_2)$; assume $a \in L(P_1)$. It follows that $\pi_2(e_1) = \pi_2(e_2) = *$ and $\pi_1(F) \xrightarrow{a}_{P_1} \pi_1(F \cup \{e_i\})$ for both $i = 1, 2$, contradicting the determinism of P_1.

   Refinement. Assume $P = P_1[b \rightsquigarrow Q]$. There are two cases.

   - $a \notin L(P_1)$ and P_1 is deterministic. It follows that $\pi_2(e_1) = \pi_2(e_2) = *$, hence $\pi_1(e_1) \neq \pi_2(e_2)$ and $\pi_1(F) \xrightarrow{a}_{P_1} \pi_1(F \cup \{e_i\})$ for both $i = 1, 2$. This contradicts the determinism of P_1.
   - P_1 and Q are deterministic. If $a \in L(P)$ then proceed as above. Otherwise $a \in L(Q)$ and $\ell_{P_1}(\pi_1(e_i)) = b$ for both $i = 1, 2$. If $\pi_1(e_1) \neq \pi_1(e_2)$ then proceed as above. Otherwise $\pi_2(e_1) \neq \pi_2(e_2)$ and $\pi_2(F) \xrightarrow{a}_Q \pi_2(F \cup \{e_i\})$ for both $i = 1, 2$; this contradicts the determinism of Q.

2. Assume $\vdash_{dis} P$. From an analysis of the rules it follows that $\vdash_{det} P$; hence we just have to prove that every initial action of P is initial-only in P; due to item 2 of Proposition 5.7 this implies proving

$$\ell_P(e) \in I(P) \Longrightarrow \{e\} \in C_P$$

for all $e \in E_P$. This is proved by induction on the structure of P.

Actions. Assume $P = a$. If $e \in E_P$ then automatically $\{e\} \in C_P$.

Choice. Assume $P = P_1 + P_2$. If $e \in E_P$ such that $\ell_P(e) = a \in I(P)$ then $a \in I(P_i)$ for some $i \in \{1,2\}$; hence $a \notin L(P_{3-i})$, which implies $e \in E_{P_i}$. Hence by induction $\{e\} \in C_{P_i} \subseteq C_P$.

Sequential composition. Assume $P = P_1; P_2$. If $e \in E_P$ such that $\ell_P(e) = a \in I(P)$ then $a \in I(P_1)$ and $a \notin L(P_2)$; hence $e \in E_{P_1}$, and by induction $\{e\} \in C_{P_1} \subseteq C_P$.

Synchronization. Assume $P = P_1 \|_A P_2$ and let $e = (e_1, e_2) \in E_P$ be such that $\ell_P(e) = a \in I(P)$. If $a \in A$ then $e_1 \neq * \neq e_2$ and $a \in I(P_1) \cap I(P_2)$; hence $\{e_i\} \in C_{P_i}$ for both $i = 1,2$, implying $\{e\} \in C_P$. Otherwise $a \in I(P_i)$ and $a \notin L(P_{3-i})$ for some $i \in \{1,2\}$; it follows that $\{e_i\} \in C_{P_i}$ and $e_{3-i} = *$; hence $\{e\} \in C_P$.

Refinement. Assume $P = P_1[a \rightsquigarrow Q]$ and let $e = (e_1, e_2) \in E_P$ be such that $\ell_P(e) = b \in I(P)$. If $b \in L(P_1)$ then $e_2 = *$ and $b \in I(P_1)$, whence $\{e_1\} \in C_{P_1}$ and $\{e\} \in C_P$. Otherwise $b \in L(Q)$; in that case $\ell_{P_1}(e_1) = a \in I(P_1)$ and $b \in I(Q)$, implying $\{e_1\} \in C_{P_1}$ and $\{e_2\} \in C_Q$, and hence $\{e\} \in C_P$.

3. Immediate. □

A.13 Lemma. Let $P, Q \in \Sigma_{flat}$ and $a \in Act$.

1. $\vdash_{det} P\left\{{}^{Q}/_{a}\right\}$ if $\vdash_{det} P$ and either $a \notin L(P)$ or $\vdash_{det} Q$.
2. $\vdash_{dis} P\left\{{}^{Q}/_{a}\right\}$ if $\vdash_{dis} P$ and either $a \notin L(P)$, or $a \in L(P) \setminus I(P)$ and $\vdash_{det} Q$, or $\vdash_{dis} Q$.
3. $\vdash_{red} P\left\{{}^{Q}/_{a}\right\}$ if $\vdash_{red} P, Q$ and in addition one of the following holds:
   (a) $a \notin S(P)$;
   (b) $a \notin SH(P)$ and $\vdash_{det} Q$;
   (c) $a \notin SD(P)$ and $\vdash_{dis} Q$;
   (d) $\vdash_{dis} Q$ and $L(Q) = I(Q)$.

Proof. Each of the statements is proved by induction on the structure of P. Note that we do not need the case for refinement, since P is assumed to be flat.

1. Immediate if $\vdash_{det} Q$. If $a \notin L(P)$ then $a \notin L(P_i)$ for both $i = 1,2$ whenever $P = P_1 * P_2$, where $* \in \{ +, ;, \|_A \}$.

2. **Actions.** Assume $P = b$; then either $b = a$, whence $a \in I(P)$ and $P\left\{{}^{Q}/_{a}\right\} = Q$, or $a \notin I(P)$ and $P\left\{{}^{Q}/_{a}\right\} = P$. In either case $\vdash_{dis} P\left\{{}^{Q}/_{a}\right\}$.

   Choice. Assume $P = P_1 + P_2$. From $\vdash_{dis} P$ it follows that $\vdash_{dis} P_i$, $L(P) \supseteq L(P_i)$ and $I(P) \supseteq I(P_i)$ for both $i = 1,2$, and hence $\vdash_{dis} P_i\left\{{}^{Q}/_{a}\right\}$ by induction. If $b \in I\left(P_1\left\{{}^{Q}/_{a}\right\}\right)$ then either $b \in I(P_1)$, in which case $b \notin L(P_2)$ because $\vdash_{dis} P_1 + P_2$ and certainly $b \notin L\left(P_2\left\{{}^{Q}/_{a}\right\}\right)$; or $a \in I(P_1)$ and $b \in I(Q)$, in which case $a \notin L(P_2)$ since $\vdash_{dis} P_1 + P_2$; hence $b \notin L\left(P_2\left\{{}^{Q}/_{a}\right\}\right)$. It follows that

$$I\left(P_1\left\{{}^{Q}/_{a}\right\}\right) \cap L\left(P_2\left\{{}^{Q}/_{a}\right\}\right) = \emptyset \ .$$

The other case follows symmetrically. We can conclude $\vdash_{dis} P\left\{{}^{Q}/_{a}\right\}$.

Sequential composition. Assume $P = P_1; P_2$. $\vdash_{dis} P_1\left\{{}^{Q}/_{a}\right\}$ because $L(P) \supseteq L(P_1)$ and $I(P) \supseteq I(P_1)$, whereas $\vdash_{det} P_2\left\{{}^{Q}/_{a}\right\}$ follows from part 1 above.

$$I\left(P_1\left\{{}^{Q}/_{a}\right\}\right) \cap L\left(P_2\left\{{}^{Q}/_{a}\right\}\right) = \emptyset$$

is proved in the same way as for the case of choice. It follows that $\vdash_{dis} P\left\{{}^{Q}/_{a}\right\}$.

Synchronization. Assume $P = P_1 \parallel_A P_2$. From $\vdash_{dis} P$ it follows that $\vdash_{dis} P_i$, $L(P) \supseteq L(P_i)$ and $I(P) \supseteq I(P_i)$ for both $i = 1, 2$; hence $\vdash_{dis} P_i\left\{{}^{Q}/_{a}\right\}$. We distinguish two cases.

- If $a \in A$ then $P\left\{{}^{Q}/_{a}\right\} = P_1\left\{{}^{Q}/_{a}\right\} \parallel_{(A\setminus\{a\})\cup L(Q)} P_2\left\{{}^{Q}/_{a}\right\}$. If $b \in L\left(P_1\left\{{}^{Q}/_{a}\right\}\right) \cap L\left(P_2\left\{{}^{Q}/_{a}\right\}\right)$ then either $b \neq a$ and $b \in L(P_1) \cap L(P_2)$, hence $b \in A \setminus \{a\}$ because $\vdash_{dis} P_1 \parallel_A P_2$, or $a \in L(P_1) \cap L(P_2)$ and $b \in L(Q)$. In both cases $b \in (A \setminus \{a\}) \cup L(Q)$.
- If $a \notin A$ then $P\left\{{}^{Q}/_{a}\right\} = P_1\left\{{}^{Q}/_{a}\right\} \parallel_A P_2\left\{{}^{Q}/_{a}\right\}$. Assume $b \in L\left(P_2\left\{{}^{Q}/_{a}\right\}\right) \cap L\left(P_2\left\{{}^{Q}/_{a}\right\}\right)$. Because $L(P_1) \cap L(P_2) \subseteq A$ due to $\vdash_{dis} P_1 \parallel_A P_2$ it follows that $b \notin L(Q)$; hence $b \in L(P_1) \cap L(P_2) \subseteq A$.

In either case it follows that $\vdash_{dis} P\left\{{}^{Q}/_{a}\right\}$. □

3. We show only the case where $P = P_1 * P_2$, where $* \in \{ +, ;, \parallel_A \}$ and $a \notin A$. It follows that $\vdash_{red} P_i$ for both $i = 1, 2$ and $P\left\{{}^{Q}/_{a}\right\} = P_1\left\{{}^{Q}/_{a}\right\} * P_2\left\{{}^{Q}/_{a}\right\}$.

   (a) It follows that $a \notin S(P_i)$ and hence by induction, $\vdash_{red} P_i\left\{{}^{red(Q)}/_{a}\right\}$ for both $i = 1, 2$.

   (b) It follows that $a \notin SH(P_i)$ and hence by induction, $\vdash_{red} P_i\left\{{}^{red(Q)}/_{a}\right\}$ for both $i = 1, 2$.

   (c) It follows that $a \notin SD(P_i)$ and hence by induction, $\vdash_{red} P_i\left\{{}^{red(Q)}/_{a}\right\}$ for both $i = 1, 2$.

   (d) By induction, $\vdash_{red} P_i\left\{{}^{red(Q)}/_{a}\right\}$ for both $i = 1, 2$.

   In each of these cases we can conclude $\vdash_{red} P\left\{{}^{Q}/_{a}\right\}$ according to Table 5.

References

[1] L. Aceto and M. Hennessy. Towards action-refinement in process algebras. Computer Science Report 3/88, University of Sussex, Apr. 1988.

[2] L. Aceto and M. Hennessy. Adding action refinement to a finite process algebra. In J. Leach Albert, B. Monien, and M. R. Artalejo, editors, *Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*, pages 506–519. Springer-Verlag, 1991. To apear in Information and Computation.

[3] E. Best, R. Devillers, and J. Esparza. General refinement and recursion operators for the Petri box calculus. To appear in STACS '93, Aug. 1992.

[4] G. Boudol. Atomic actions. *Bull. Eur. Ass. Theoret. Comput. Sci.*, 38:136–144, June 1989. note.

[5] G. Boudol and I. Castellani. Permutations of transitions: An event structure semantics for CCS and SCCS. In de Bakker et al. [9], pages 411–427.

[6] L. Castellano, G. De Michelis, and L. Pomello. Concurrency vs. interleaving: An instructive example. *Bull. Eur. Ass. Theoret. Comput. Sci.*, 31:12–15, 1987.

[7] R. Costantini. Eine kompositionelle Semantik für eine CCSP-artige Sprache auf der Grundlage von Konfigurationsstrukturen. Universität Hildesheim; draft, 1992.

[8] P. Darondeau and P. Degano. Event structures, causal trees, and refinement. In B. Rovan, editor, *Mathematical Foundations of Computer Science 1990*, volume 452 of *Lecture Notes in Computer Science*, pages 239–245. Springer-Verlag, 1990. To appear in Theoretical Comput. Sci.

[9] J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors. *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[10] P. Degano and R. Gorrieri. An operational definition of action refinement. Technical Report TR–28/92, Università di Pisa, 1992.

[11] R. Gorrieri. *Refinement, Atomicity and Transactions for Process Description Languages.* PhD thesis, Università di Pisa, 1991. report no. TD–2/91.

[12] R. Gorrieri, S. Marchetti, and U. Montanari. A^2CSS: Atomic actions for CCS. *Theoretical Comput. Sci.*, 72:203–223, 1990.

[13] L. Jategaonkar and A. Meyer. Self-synchronization of concurrent processes. Preliminary Report — to appear in LICS'93, 1992.

[14] L. Jategaonkar and A. Meyer. Testing equivalences for Petri nets with action refinement. In W. R. Cleaveland, editor, *Concur '92*, volume 630 of *Lecture Notes in Computer Science*, pages 17–31. Springer-Verlag, 1992.

[15] M. Nielsen, U. Engberg, and K. G. Larsen. Fully abstract models for a process language with refinement. In de Bakker et al. [9], pages 523–549.

[16] A. Rabinovich and B. A. Trakhtenbrot. Behaviour structure and nets. *Fundamenta Informaticae*, XI(4):357–404, Dec. 1988.

[17] R. van Glabbeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions. In A. Kreczmar and G. Mirkowska, editors, *Mathematical Foundations of Computer Science 1989*, volume 379 of *Lecture Notes in Computer Science*, pages 237–248. Springer-Verlag, 1989.

[18] R. van Glabbeek and U. Goltz. Equivalences and refinement. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes*, volume 469 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[19] R. van Glabbeek and U. Goltz. Refinement of actions in causality based models. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems — Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 267–300. Springer-Verlag, 1990.

[20] W. Vogler. Bisimulation and action refinement. SFB-Bericht 342/10/90 A, Technische Universität München, May 1990.

[21] W. Vogler. Failures semantics based on interval semiwords is a congruence for refinement. In C. Choffrut and T. Lengauer, editors, *STACS 90*, volume 415 of *Lecture Notes in Computer Science*, pages 285–297. Springer-Verlag, 1990.

[22] G. Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer-Verlag, 1987.

[23] G. Winskel. An introduction to event structures. In de Bakker et al. [9], pages 364–397.