On Syntactic and Semantic Action Refinement*

Ursula Goltz\(^1\), Roberto Gorrieri\(^2\), Arend Rensink\(^3\)

\(^1\) Institut für Informatik, University of Hildesheim  
P.O.Box 101363, D–31113 Hildesheim, Germany  
\(^2\) Dipartimento di Matematica, University of Bologna  
Piazza di Porta San Donato 5, I–40127 Bologna, Italy  
\(^3\) Tele-Informatics and Open Systems, University of Twente  
P.O.Box 217, NL–7500 AE Enschede, The Netherlands  
(on leave at the University of Hildesheim while doing this research)

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 synchronisation 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) \( \text{red}(P) \) 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 \( \text{red}(P) \) have the same semantics. The results we present are essentially dependent on the question whether the refined action can be synchronised or not. In the latter case, \( P \) and \( \text{red}(P) \) 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 synchronisation up to isomorphic domains of configurations. Subsequently we also give sufficient (but not necessary) syntactic conditions for reducible terms.

---

In: Theoretical Aspects of Computer Software, Sendai, Japan, April 1994; M. Hagiya and J.C. Mitchell (Eds.), LNCS 789, pp. 385–404

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 \( P[a \sim Q] \) 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 formalise it effectively is proving to be a complex issue, and consequently research on this subject has taken various different approaches.

*This work has been partially supported by Esprit Basic Research Working Group 6067 CALIBAN (Causal Calculi Based on Nets); the second author also 40% by the Italian CNR.*
Two lines of research can be recognised. On the one hand there is atomic refinement [4, 12, 13], 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.

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 $P[a \sim Q]$ can be defined using this operator. For example, when using event structures as semantic domains, an event structure $\mathcal{F} = \llbracket Q \rrbracket$, representing the semantics of $Q$, would be substituted for every $a$-labelled event $e$ in the event structure $\mathcal{E} = \llbracket P \rrbracket$. 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, 19, 21, 22, 23] 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 $P[a \sim Q]$ 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 $P[a \sim Q]$ is, by definition, the semantics of the term $P \{Q/a\}$. This line of research has been pursued in [1, 2, 16].

Simple examples (see below) show that the two approaches do not coincide in general. This is essentially due to the interplay between refinement and synchronisation. 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 synchronisation. 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 synchronisation. We provide the language with a flow event structure semantics, as defined in [21], 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:

\[
\begin{array}{ccc}
P[a \sim Q] & \xrightarrow{\text{syntactic ref.}} & P\{Q/a\} \\
\downarrow & & \downarrow \\
[P][a \sim [Q]] & \xrightarrow{\text{semantic ref.}} & [P\{Q/a\}]
\end{array}
\]

(1)

It turns out that the problem can be reduced to the following question: under which conditions does refinement distribute over parallel composition with synchronisa-
In this paper we use a TCSP-like synchronisation operator,\(^4\) which takes the form \(P_1 \mid_A P_2\), where \(A\) denotes the set of communication actions, i.e., those actions on which both \(P_1\) and \(P_2\) are forced to synchronise. Distribution of refinement over parallel composition then means that the semantic equation
\[
(P_1 \mid_A P_2)[a \sim Q] \cong P_1[a \sim Q] \mid_A P_2[a \sim Q]
\]
holds, where \(A'\) may be some modification of the synchronisation set \(A\), in case synchronising actions are refined. This equation however does not hold in general. The terms \((a \mid_{\{b\}} b; c)[a \sim b]\) and \((a[a \sim b] \mid_{\{b\}} b; c[a \sim b])\) for instance are not equivalent: intuitively, in the first term, \(c\) on the right hand side is prevented from occurring since the preceding \(b\) cannot synchronise 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 synchronisation, 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 synchronise with “old” ones (the \(b\) already occurring before refinement). This is in contrast with the intuition that, in \(P[a \sim Q]\), 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: \(P[a \sim Q]\) is well-formed if \(L(P) \cap L(Q) = \emptyset\), where \(L(P)\) denotes the alphabet of \(P\). We first consider the case that synchronising actions are not refined, that is, \(a \notin A\) for \((P_1 \mid_A P_2)[a \sim Q]\). 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 the full version of this paper.

The situation becomes much more difficult if we allow to refine synchronising actions, that is, \(a \in A\) for \((P_1 \mid_A P_2)[a \sim Q]\). 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 \((P_1 \mid_A P_2)[a \sim Q]\) to a characterisation of the sublanguage in which refinement may be replaced by substitution.

Nota bene. Because of lack of space, most of the proofs have been omitted from this paper. They can be found in the full version [11].

## 2 Syntax and Semantics of the Language

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

\[
P ::\ a \mid P + P \mid P \parallel P \mid P[a \sim P].
\]

\(^4\) This choice does not affect the central problem essentially. In Sect. 6 we briefly discuss the CCS setting of [1, 2].
Most of the operators are standard. We use a family of synchronisation operators \( \{ \parallel_A \}_{A \subseteq A} \) corresponding to the TCSP approach. The refinement operator \( P[a \leadsto Q] \) acts on single actions at a time. The behaviour of \( P[a \leadsto Q] \) is derived from the behaviour of \( P \) by replacing every execution of the action \( a \) by the behaviour of \( Q \). \( \Sigma \) 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 \( \Sigma \); to improve the readability, we will let sequential composition bind stronger than choice and synchronisation, and refinement stronger than any of the binary operators.

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 synchronising 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 \( P \in \Sigma \).

| \( L(a) \) | := | \{ a \} |
| \( L(P + Q) \) | := | \( L(P) \cup L(Q) \) |
| \( L(P; Q) \) | := | \( L(P) \cup L(Q) \) |
| \( L(P \parallel Q) \) | := | \( L(P) \cup L(Q) \cup A \) |
| \( L(P[a \leadsto Q]) \) | := | \( \begin{cases} (L(P) \setminus \{ a \}) \cup L(Q) & \text{if } a \in L(P) \\ L(P) & \text{otherwise} \end{cases} \) |

| \( S(a) \) | := | \emptyset |
| \( S(P + Q) \) | := | \( S(P) \cup S(Q) \) |
| \( S(P; Q) \) | := | \( S(P) \cup S(Q) \) |
| \( S(P \parallel Q) \) | := | \( S(P) \cup S(Q) \cup (L(P) \cup L(Q)) \cap A \) |
| \( S(P[a \leadsto Q]) \) | := | \( \begin{cases} (S(P) \setminus \{ a \}) \cup L(Q) & \text{if } a \in S(P) \\ S(P) \cup S(Q) & \text{if } a \in L(P) \setminus S(P) \\ S(P) & \text{otherwise.} \end{cases} \) |

Table 1. Label set and synchronising set

We now argue that it makes sense to restrict the refinements under consideration to a certain format. Consider a term of the form \( P[a \leadsto Q] \). 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 in the specification \( P \). This makes it impossible for actions in \( P \) to synchronise with those in \( Q \) (after refinement) and hence rules out a kind of confusion of abstraction levels. In other words we assume

\[
L(P) \cap L(Q) = \emptyset
\]
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 relabelling to the syntax— this assumption can be dropped and our results can be generalised to the entire $\Sigma$, as shown in [11].

| $\vdash a$ | $\vdash P, Q \ast \in \{:, \|, \_\}$ | $\vdash P, Q \quad L(P) \cap L(Q) = \emptyset$ |
| $\vdash P \ast Q$ | $\vdash P[a \sim Q]$ |

Table 2. Well-formedness

2.2 Flow Event Structure Semantics

We interpret the terms of $\Sigma$ in the model of flow event structures proposed by Boudol and Castellani [5]. The interpretation is standard; see for instance [21].

**Definition 1.** A flow event structure is a tuple $E = \langle E, \prec, \#_E, \ell \rangle$ where
- $E$ is a set of events;
- $\prec \subseteq E \times E$ is an irreflexive flow relation;
- $\#_E \subseteq E \times E$ is a symmetric conflict relation;
- $\ell : E \rightarrow A$ is a labelling function.

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

**Definition 2.** Let $E$ be a flow event structure and $F \subseteq E_{\prec}$ a subset of the events of $E$. $F$ is a configuration of $E$ if it satisfies the following conditions:
- $F$ does not contain flow-cycles, i.e. $(\prec_E \cap (F \times F))^+$ (the transitive closure of $\prec_E$) is irreflexive;
- $F$ is conflict-free, i.e. $\exists \forall d, e \in F. d \#_E e$;
- $F$ is closed under non-conflicting causes: $\forall d \in (E_{\prec} \setminus F). \forall e \in F. d \prec_E e \implies \exists d' \in F. d \#_E d' \prec_E e$.

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

- $d \prec_E e \iff f(d) \prec_F f(e)$
- $d \#_E e \iff f(d) \#_F f(e)$
- $\ell_E (e) = \ell_F (f(e))$.

A number of operations over $E$, 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 $\Sigma$) and their semantic counterparts (to be defined below over $E$); the context will clarify which of the two we mean.
Definition 3. 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

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

Definition 4. 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

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

Definition 5. Let $\mathcal{E}, \mathcal{F} \in \mathbf{E}$ and $A \subseteq A$; let $a \notin E_\mathcal{E} \cup E_\mathcal{F}$. The synchronisation of $\mathcal{E}$ and $\mathcal{F}$ over $A$ is defined by $\mathcal{E} \parallel_A \mathcal{F} := \langle \{a\}, \preceq, \# \rangle$, where:

- $E = \{ (d, e) \in E_\mathcal{E} \times \{a\} \cup \{ (a, e) \in E_\mathcal{F} \mid \ell_\mathcal{F}(d) = \ell_\mathcal{F}(e) \in A \} ;$
- $(d, e) \preceq (d', e')$ iff $d \preceq_\mathcal{E} d' \land e \preceq_\mathcal{F} e'$;
- $(d, e) \# (d', e')$ iff $d \#_\mathcal{E} d' \land e \#_\mathcal{F} e' \lor (d = d' \land e \neq e') \lor (e = e' \land d \neq d')$.

- $\ell(d, e) = \begin{cases} \ell_\mathcal{E}(d) & \text{if } e = \star; \\ \ell_\mathcal{F}(e) & \text{otherwise.} \end{cases}$

Definition 6. Let $\mathcal{E}, \mathcal{F} \in \mathbf{E}$, $E_\mathcal{F} \neq \emptyset$ and $a \in A$; let $a \notin E_\mathcal{E} \cup E_\mathcal{F}$. The refinement of $a$ by $\mathcal{F}$ in $\mathcal{E}$ is defined by $\mathcal{E}[a \sim \mathcal{F}] := \langle \{a\}, \preceq, \# \rangle$, where:

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

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

$$[a] := \langle \{a\}, 0, 0, (e, a) \rangle$$

$$[P + Q] := [P] + [Q]$$

$$[P; Q] := [P]; [Q]$$

$$[P \parallel_A Q] := [P] \parallel_A [Q]$$

$$[P[a \sim Q]] := [P][a \sim [Q]] .$$

This semantics then induces an equivalence relation $\cong_\mathcal{E}$ over $\Sigma$:

$$P \cong_\mathcal{E} Q \iff [P] \cong [Q] .$$

Because $\cong$ is a congruence over all the operators we have defined, $\cong_\mathcal{E}$ is a congruence over $\Sigma$.

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 $\cong_\mathcal{E}$ is too strong; it is for instance easy to see that $a \parallel_{[a]} a \not\cong_\mathcal{E} a$ whereas in many cases these terms are considered equivalent. In Sect. 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:

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

These sets of configurations are called families of configurations or configuration structures.\(^5\) Configuration structures form the standard underlying semantic model for event structures; cf. [24, 25]. 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. [20, 23]. Hence proving equality modulo configuration structure isomorphism immediately implies that it holds under those weaker equivalences, too.

**Definition 7.** Let $E$ be a set of events. A stable configuration structure over $E$ is a tuple $(C, \sqrt{}, \ell)$ where

- $C \subseteq \text{Fin}(E)$ is a set of finite configurations such that
  - $\emptyset \in C$;
  - $\forall F, G, H \in C, (F \cup G) \subseteq H \implies (F \cup G) \in C \wedge (F \cap G) \in C$;
  - $\forall F \in C, \forall d, e \in F, d \neq e \implies \exists G \in C, d \in G \iff e \notin G$.

- $\sqrt{} \subseteq C$ is a set of terminated configurations, sometimes treated as a predicate in postfix notation, such that $F \subseteq G$ implies $\neg F \sqrt{}$ (i.e. terminated configurations must be maximal w.r.t. $\subseteq$);

- $\ell: E \rightarrow A$ is a labelling function.

The class of configuration structures is denoted $\mathbb{C}$. We will sometimes use $C$ to denote the entire structure, and $\sqrt{C}$ and $\ell_C$ 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 standard; cf. [5].

**Proposition 8.** 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 $f: E_C \rightarrow E_D$ which preserves event labelling, such that the pointwise extension of $f$ to sets of events maps $C$ to $D$ and $\sqrt{C}$ to $\sqrt{D}$:

$$f(C) = D,$$

$$\sqrt{D} = f(\sqrt{C})$$

$$\ell_C = \ell_D \circ f.$$  

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 $\Sigma$:

$$P \cong Q :\iff CS[P] \cong CS[Q].$$

\(^5\) There is a subtle difference between these two concerning the presence of certain infinite objects; this is however irrelevant to the discussion at hand.
As mentioned above, this is weaker than flow event structure isomorphism; that is,
\[ P \cong_e Q \implies P \cong_e Q \, . \]

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 synchronisation and refinement over \( C \). A compositional semantics does however exist; see e.g. [7]. Moreover, the following is known to hold.

**Proposition 9.** \( \cong_e \) is a congruence for the operators of \( \Sigma \).

## 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 \( P \{ Q/a \} \) 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.

**Definition 10.** Let \( P, Q \in \Sigma_{flat} \) be two flat terms. The operation of syntactic substitution, denoted \( P \{ Q/a \} \), is defined by induction on the syntactical structure of \( P \) as follows:

\[
\begin{align*}
b \{ Q/a \} & := \begin{cases} Q & \text{if } b = a \\ b & \text{otherwise} \end{cases} \\
(P_1 \cdot P_2) \{ Q/a \} & := (P_1 \{ Q/a \}) \cdot (P_2 \{ Q/a \}) & \text{where } * \in \{ +, ; \} \\
(P_1 \parallel_A P_2) \{ Q/a \} & := (P_1 \{ Q/a \}) \parallel_A (P_2 \{ Q/a \}) .
\end{align*}
\]

The rule for parallel composition uses a special construct \( A \{ Q/a \} \), defined as follows:

\[
A \{ Q/a \} := \begin{cases} (A \setminus \{ a \}) \cup L(Q) & \text{if } a \in A \\ A & \text{otherwise.} \end{cases}
\]

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

**Proposition 11.** *If* \( P, Q \in \Sigma_{flat} \) *and* \( a \in A \) *then* \( P \{ Q/a \} \in \Sigma_{flat} \).

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.

**Definition 12.** The reduction of a term \( P \in \Sigma \), denoted \( \text{red}(P) \), is defined inductively on the structure of \( P \) as follows:

\[
\begin{align*}
\text{red}(a) & := a \\
\text{red}(P \cdot Q) & := \text{red}(P) \cdot \text{red}(Q) & \text{where } * \in \{ +, ;, \parallel_A \} \\
\text{red}(P[a \sim Q]) & := \text{red}(P) \{ \text{red}(Q)/a \} .
\end{align*}
\]
Note that in the rule for refinement, we have \( I(P) \cap I(Q) = \emptyset \) because we only consider well-formed terms. Due to Prop. 11, \( \text{red}(P) \{ \text{red}(Q) / a \} \) is always defined. The following proposition states that \text{red} is well-behaved in the sense that the alphabet and set of synchronising actions of a given term are insensitive to reduction.

**Proposition 13.** If \( P \in \Sigma \) then \( I(\text{red}(P)) = I(P) \) and \( S(\text{red}(P)) = S(P) \).

The aim of this paper is now to find general conditions for

\[
P \cong \text{red}(P)
\]

where \( \cong \) is some semantic equivalence, such as flow event structure isomorphism \( \cong_\varepsilon \).

## 4 Refinement of Non-synchronising 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 synchronisation. Recalling that \( S(P) \) denotes the set of synchronising actions, this condition can be formally stated by requiring that \( a \not\in S(P) \) for any term of the form \( P[a \sim Q] \). 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.

**Lemma 14.** Let \( P, P_1, P_2, Q \in \Sigma_{\text{flat}} \) be flat terms, let \( a, b \in A \) and \( A \subseteq A \).

1. \( a[a \sim Q] \cong_\varepsilon Q \)
2. \( b[a \sim Q] \cong_\varepsilon b \) provided that \( b \neq a \)
3. \( (P_1; P_2)[a \sim Q] \cong_\varepsilon P_1[a \sim Q]; P_2[a \sim Q] \)
4. \( (P_1 + P_2)[a \sim Q] \cong_\varepsilon P_1[a \sim Q] + P_2[a \sim Q] \)
5. \( (P_1 || P_2)[a \sim Q] \cong_\varepsilon P_1[a \sim Q] || P_2[a \sim Q] \) provided that \( a \not\in A \).

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

**Example 1.** Let \( P_1 = Q = a, A = \{a\} \) and \( P_2 = a;b \).

\[
(P_1 || P_2)[b \sim Q] \cong_\varepsilon (a ||_A a); a
\]

\[
P_1[b \sim Q] ||_A P_2[b \sim Q] \cong_\varepsilon a ||_A (a;a).
\]

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 synchronisation component wants to execute one more \( a \) in synchrony with the other, but the other component is already finished.

**Theorem 15.** Let \( P, Q \in \Sigma_{\text{flat}} \) and \( a \in A \). If \( a \not\in S(P) \) then \( P[a \sim Q] \cong_\varepsilon P \{ Q/a \} \).

**Proof.** By induction on the syntactic structure of \( P \). The base cases are when \( P \) is an action. If \( P = a \), then \( a[a \sim Q] \cong_\varepsilon Q \) by 1 of Lemma 14, and \( Q \cong_\varepsilon a \{ Q/a \} \) because of Def. 10. Analogously if \( P = b \). For the inductive case, let \( * \in \{ +, ;, \|_A \} \):

\[
(P_1 * P_2)[a \sim Q] \cong_\varepsilon P_1[a \sim Q] * P_2[a \sim Q] \quad \text{(Lemma 14)}
\]

\[
\cong_\varepsilon (P_1 \{ Q/a \}) * (P_2 \{ Q/a \}) \quad \text{(induction and congruence of \( \cong_\varepsilon \))}
\]

\[
= (P_1 + P_2) \{ Q/a \} \quad \text{(Def. 10).}
\]

If \( * = \|_A \) then Lemma 14 is applicable because \( A \subseteq S(P) \), hence \( a \not\in A \). \( \Box \)
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. Now a straightforward consequence of the theorem above is the following:

**Corollary 16.** Let $P \in \Sigma$. If $\vdash_i P$ then $P \equiv_e \text{red}(P)$.

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

$$\text{red}(P[a \sim Q]) = \text{red}(P)[\text{red}(Q)/a] \equiv_e \text{red}(P)[a \sim \text{red}(Q)] \text{ (Th. 15)}$$

$$\equiv_e \text{red}(P)$$

(induction and congruence of $\equiv_e$).

Theorem 15 is applicable because $\vdash_i P[a \sim Q]$ guarantees $a \not\in S(P)$.

---

### Table 3. Interference freedom

<table>
<thead>
<tr>
<th>$\vdash_i P, Q$</th>
<th>$a \in {+, |, a}$</th>
<th>$\vdash_i P[a \sim Q]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\vdash_i P$</td>
<td>$\vdash_i P, Q$</td>
<td>$\vdash_i P[a \sim Q]$</td>
</tr>
</tbody>
</table>

---

5 **Refinement of Synchronising 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 synchronisation actions. The following example shows that rule 5 of Lemma 14 does not hold any more.

**Example 2.** Let $P_1 = P_2 = a$, $A = \{a\}$ and $Q = b$.

$$(P_1 || A \iff P_2)[a \sim Q] \equiv_e b ||_{[a]} b$$

$$P_1[a \sim Q] || A \iff P_2[a \sim Q] \equiv_e b ||_{[a]} b$$

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 Synchronisation

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

$$(P_1 || A \iff P_2)[a \sim Q] \equiv_e P_1[a \sim Q] || (A \setminus \{a\}) \cup Q, P_2[a \sim Q].$$

(Note that there are alternative ways of distributing refinement over synchronisation. In Sect. 6 we briefly discuss one particular other choice based on a CCS-like synchronisation operator.)

Example 2 above is indeed repaired by this change, because now the second term (in which refinement is distributed over the subterms) becomes

$$P_1[a \sim Q] || (A \setminus \{a\}) \cup Q, P_2[a \sim Q] \equiv_e b ||_{[a]} b$$
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):\( (a ||_{\{a\}} a)[a \sim b; c_1 + b; c_2] \) and \( (a[a \sim b; c_1 + b; c_2]) ||_{\{b, c_1, c_2\}} (a[a \sim b; c_1 + b; c_2]) \).

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.

Example 3. Let $P_1 = P_2 = a$ and $Q = b$. One would expect the following to hold for
a reasonable equivalence relation $\equiv$:

\[ a[a \sim Q] \cong b \cong b \cong a[a \sim Q] \cong a[a \sim Q] \]

However, this is not satisfied if we replace $\cong$ by $\equiv$, due to the flow event structure
construction for synchronisation, which introduces inconsistent events (i.e. events $e$
such that $e \neq e$). In fact we have $b \not\equiv b \equiv b$ and hence

\[ (P_1 ||_{\{a\}} P_2)[a \sim Q] \not\equiv, P_1[a \sim Q] \equiv (P_2[a \sim Q]) \]

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

Let us analyse the terms in the proposed new distribution rule (5). Define

\[ C = CS[(P_1 ||_{\{a\}} P_2)[a \sim Q]] \]
\[ D = CS[P_1[a \sim Q] ||_{(A \{a\}) \cup L(Q)} P_2[a \sim Q]] \]

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

\[
X_1 := \{ (d_1, d_2, e) \in ((E_1 \times E_2) \times E_Q) \mid \ell_1(d_1) = \ell_2(d_2) = a \}
\]
\[
X_2 := \{ ((d_1, d_2), \star) \mid d_1 \in E_1 \land \ell_1(d_1) = \ell_2(d_2) \in (A \setminus \{a\}) \}
\]
\[
X_3 := \{ ((d, \star), \star) \mid d \in E_1 \land \ell_1(d) \notin A \}
\]
\[
X_4 := \{ ((\star, d), \star) \mid d \in E_2 \land \ell_2(d) \notin A \}
\]
\[
Y_1 := \{ ((d_1, e_1), (d_2, e_2)) \mid \ell_1(d_1) = \ell_2(d_2) = a \land \ell_Q(e_1) = \ell_Q(e_2) \}
\]
\[
Y_2 := \{ ((d_1, \star), (d_2, \star)) \mid d_1 \in E_1 \land \ell_1(d_1) = \ell_2(d_2) \in (A \setminus \{a\}) \}
\]
\[
Y_3 := \{ ((d, \star), \star) \mid d \in E_1 \land \ell_1(d) \notin A \}
\]
\[
Y_4 := \{ ((\star, d), \star) \mid d \in E_2 \land \ell_2(d) \notin A \}
\]

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

\[
f_1: ((d_1, d_2), e) \mapsto ((d_1, e), (d_2, e))
\]
\[
f_2: ((d_1, d_2), \star) \mapsto ((d_1, \star), (d_2, \star))
\]
\[
f_3: ((d, \star), \star) \mapsto ((d, \star), \star)
\]
\[
f_4: ((\star, d), \star) \mapsto ((\star, d), \star)
\]
$f_2 \sim f_4$ are obviously bijective. For $f_1$ this is not immediately clear; surjectivity requires 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 dearly injective.

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 14 to non-interference-free terms. Note that this result necessarily depends on the chosen semantics: in a stronger semantics our conditions will in general no longer be sufficient (Ex. 3 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

\[ F \overset{a}{\leftrightarrow} C G := \exists e \in G. F = G \setminus \{ e \} \land \ell_C(e) = a. \]

We will drop the subscript $C$ when it is clear from the context.

**Definition 17.** Let $C$ be a configuration structure; let $a$ be an action.

- $a$ is *executed* in $C$ (at $F$) if $F \overset{a}{\rightarrow} G$ for some $F, G \in C$;
- $a$ is *initial* in $C$ if $\emptyset \overset{a}{\rightarrow} F$ for some $F \in C$;
- $a$ is *noninitial* in $C$ if $\emptyset \not\overset{a}{\rightarrow} F$ 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 $F \overset{a}{\rightarrow} G$ and $F \overset{a}{\rightarrow} H \neq G$ for some $F, G, H \in C$; otherwise $a$ is *deterministic* in $C$;
- $a$ is *auto-concurrent* in $C$ (at $F$) if $F \overset{a}{\rightarrow} G$ and $F \overset{a}{\rightarrow} H \neq G$ 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.)

The following is a derived property that is defined only over structures of the form $C = C_1 \parallel_A C_2$, where $C_i \in C$ 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 synchronisation is auto-sequential in both synchronising partners. It is slightly weaker than requiring that $a$ is auto-sequential in both synchronising partners, since all $a$-autoconcurrent states in the partners may be unreachable (e.g. because of synchronisation deadlocks), in which case $a$ is still two-way sequential.

**Example 4.** If $P_1 = a; (b \parallel \| b) + b$ 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.
Theorem 18. Let $P_1, P_2, Q \in \Sigma$ and $a \in A \subseteq A$. The distributivity property

\[ (P_1 \parallel_A P_2)[a \sim Q] \cong_{\varepsilon} P_1[a \sim Q][a \setminus A] \cup L(Q), P_2[a \sim Q] \]

holds 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. We only give an outline of the proof.

1. Prove $F_C \in C \implies f(F_C) \in D$, independent of conditions C1–4.
2. Prove $F_D \in D \implies \exists F_C \in C, f(F_C) = F_D$ 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 $f: E_C \to E_D$ is bijective and $f(C) = D$.
3. Prove $F_C \sqsubseteq_C f(F_C) \sqsubseteq_D$. Because we know that (the pointwise extension of) $f$ is a bijection from $C$ to $D$ this proves $f(\sqsubseteq_C) = \sqsubseteq_D$.
4. $t_D = t_C \circ f^{-1}$ 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 $C \cong_{\varepsilon} D$ then $|C| = |D|$ because we are dealing only with finite terms; moreover $f$ is injective; hence if $C \cong_{\varepsilon} D$ then $f(C) = D$. Now prove that each of the following conditions is sufficient to construct a configuration in $D$ which is not in $f(C)$.

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. \hfill \Box

Example 5. To see the necessity of the conditions in the theorem above, consider $P_1 := a; c || a; c, P_2 := a, A := \{a\}$ 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

\[ (P_1 \parallel_A P_2)[a \sim Q] \cong_{\varepsilon} (\{a; c || a; c\}[a \sim b; b]) \]

\[ \not\cong_{\varepsilon} (b; b; c || b; b; c)[b; b] \]

\[ \cong_{\varepsilon} \left( P_1[a \sim Q][a \setminus A] \cup L(Q), P_2[a \sim Q] \right) . \]

5.2 The Language of Reducible Terms

The conditions of Th. 18 are based on the semantic properties in Def. 17. We are however also interested in a syntactic characterisation of the (sub)language in which syntactic and semantic refinement coincide, i.e. which are reducible in the sense that $P \cong_{\varepsilon} red(P)$. Such a syntactic characterisation 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. 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 \(\cong\) — which is important since as mentioned
before, most partial order equivalences found in the literature are indeed weaker.

We do not claim that our conditions are optimal in the sense that they identify the
maximal number of reducible terms under \(\cong\). 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 \(\Sigma\) to \(2^A\) inductively on the structure of
the terms. \(I\) returns the initial actions, and \(D\) the set of distributed actions which

\[
\begin{align*}
I(a) & := \{a\} \\
I(P + Q) & := I(P) \cup I(Q) \\
I(P \cdot Q) & := I(P) \\
I(P \parallel_A Q) & := (I(P) \cup I(Q)) \setminus A \cup (I(P) \cap I(Q) \cap A) \\
I(P[a \sim Q]) & := \begin{cases} 
(I(P) \setminus \{a\}) \cup I(Q) & \text{if } a \in I(P) \\
I(P) & \text{otherwise.}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
D(a) & := \emptyset \\
D(P \cdot Q) & := D(P) \cup D(Q) \setminus \{a\} , \text{ where } * \in \{+;\} \\
D(P \parallel_A Q) & := (D(P) \cap D(Q) \cap A) \cup (D(P) \cup D(Q) \cup (I(P) \cap I(Q))) \setminus A \\
D(P[a \sim Q]) & := \begin{cases} 
(D(P) \setminus \{a\}) \cup I(Q) & \text{if } a \in D(P) \\
D(P) & \text{otherwise.}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
SH(a) & := \emptyset \\
SH(P \cdot Q) & := SH(P) \cup SH(Q) \setminus \{a\} , \text{ where } * \in \{+;\} \\
SH(P \parallel_A Q) & := SH(P) \cup SH(Q) \cup ((D(P) \cup D(Q)) \setminus A) \\
SH(P[a \sim Q]) & := \begin{cases} 
(SH(P) \setminus \{a\}) \cup I(Q) & \text{if } a \in SH(P) \\
SH(P) \cup D(Q) & \text{if } a \in SH(P) \setminus SH(P) \\
SH(P) \cup SH(Q) & \text{if } a \in L(P) \setminus SH(P) \\
SH(P) & \text{otherwise.}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
SD(a) & := \emptyset \\
SD(P \cdot Q) & := SD(P) \cup SD(Q) \setminus \{a\} , \text{ where } * \in \{+;\} \\
SD(P \parallel_A Q) & := SD(P) \cup SD(Q) \cup (D(P) \cap D(Q) \cap A) \\
SD(P[a \sim Q]) & := \begin{cases} 
(SD(P) \setminus \{a\}) \cup I(Q) & \text{if } a \in SD(P) \\
SD(P) \cup D(Q) & \text{if } a \in SD(P) \setminus SD(P) \\
SD(P) \cup SD(Q) & \text{if } a \in L(P) \setminus SD(P) \\
SD(P) & \text{otherwise.}
\end{cases}
\end{align*}
\]

Table 4. Initial and distributed actions

may occur auto-concurrently. It follows that \(I(P) \subseteq I(P)\) and \(D(P) \subseteq I(P)\) 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 synchronising actions in \(A\) in such a subterm are \emph{distributed in one operand} (for \(SH\)) or \emph{distributed in both operands} (for \(SD\)). This information is used to approximate the awkward semantical property of \emph{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 characterisations for the corresponding semantic properties.

**Proposition 19.**
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 \(a \in D(P)\);

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.

**Proposition 20.** If \(P \in \Sigma\) then \(f(red(P)) = f(P)\) for all \(f = I, C, SH, SD\).

Table 5 defines a number of predicates over \(\Sigma\), intended to capture the rest of the semantic properties of Def. 17 in terms of syntax. \(\vdash_{\text{det}} \subseteq \Sigma\) captures the notion of \emph{determinism}, and \(\vdash_{\text{dis}} \subseteq \Sigma\) is intended to capture the notion of \emph{distinctness}. It follows that \(\vdash_{\text{dis}} \subseteq \vdash_{\text{det}}\). Finally, \(\vdash_{\text{red}}\) defines the notion of \emph{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.

**Proposition 21.**
1. If \(\vdash_{\text{det}} P\) then \(P\) is deterministic;
2. If \(\vdash_{\text{dis}} P\) then \(P\) is distinct;
3. If \(\vdash_{\text{dis}} P\) and \(L(P) = I(P)\) then \(P\) is atomic.

Similar to the calculation of the action sets (cf. Prop. 20), 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 \(red\), properties of the term as a whole should not be affected. This is formulated in the following proposition.

**Proposition 22.** Let \(P \in \Sigma\).
1. If \(\vdash_{\text{det}} P\) then \(\vdash_{\text{det}} red(P)\);
2. If \(\vdash_{\text{dis}} P\) then \(\vdash_{\text{dis}} red(P)\);
3. If \(\vdash_{\text{red}} P\) then \(\vdash_{\text{red}} red(P)\).

It is our intention that the syntactically decidable criterion of reducibility provides sufficient (but not necessary) conditions for the semantic properties discussed in Th. 18. This leads to the following theorem, which states that for reducible terms, semantic and syntactic refinement coincide, at least for flat terms.
In each of these cases, due to Theorem 1.8, except if $i$ for both $a$.

\[ \text{Let} \quad \text{Proof.} \]

\[ \text{Hence by Prop. 1.9,} \quad \text{and} \quad \text{then} \quad \text{for} \quad \text{atomic.} \]

\[ \text{In each of these cases, due to Th. 18 we have} \]

\[ P[a \sim Q] \cong \varepsilon P_i[a \sim Q] \mid_{A \setminus \{a\}} \cup L(Q) P_2[a \sim Q] \]

\[ \cong \varepsilon P_i \{Q/a\} \mid_{A \setminus \{a\}} \cup L(Q) P_2 \{Q/a\} \]

\[ = P \{Q/a\} . \]

**Table 5.** Determinism, distinctness and reducibility

**Theorem 23.** Let $P, Q \in \Sigma_{\text{flat}}, a \in A$. If $\vdash_{\text{red}} P[a \sim Q]$ then $P[a \sim Q] \cong \varepsilon P \{Q/a\}$.

**Proof.** By induction on the structure of $P$. The cases are analogous to the proof of Th. 15, except if $P = P_i \parallel A P_j$ such that $a \in A$. By induction $P_i \{Q/a\} \cong \varepsilon P_i[a \sim Q]$ for both $i = 1, 2$. There are three subcases.

- $a \notin SH(P)$ and $\vdash_{\text{det}} Q$. It follows, by construction of $SH$, that $a \notin D(P_1) \cup D(P_2)$; hence by Prop. 19, $a$ is not auto-concurrent in $P_1$ or $P_2$; hence $a$ is two-way sequential in $P_i \parallel_A P_j$. In addition, $Q$ is deterministic by Prop. 21.

- $a \notin SD(P)$ and $\vdash_{\text{dis}} Q$. It follows, by construction of $SD$, that $a \notin A \setminus (D(P_i) \cap D(P_j))$; hence $a \notin D(P)$ and by Prop. 19, $a$ is auto-sequential in $P_1 \parallel_A P_2$. In addition, $Q$ is distinct by Prop. 21.

- $\vdash_{\text{dis}} Q$ and $L(Q) = I(Q)$. It follows by Prop. 21 that $Q$ is atomic.
This concludes the proof. □

The following corollary extends the above result to $\Sigma$, using Propositions 20 and 22 which state that our syntactic machinery is insensitive to the application of the function $\text{red}$; by removing refinement operators from the inside out using $\text{red}$, 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.

**Corollary 24.** Let $P \in \Sigma$. If $\Downarrow_{\text{red}} P$ then $P \cong_{\text{a}} \text{red}(P)$.

### 6 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 synchronising 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 relabelling operator in the language. If we do allow synchronisation actions to be refined, the correspondence is less straightforward. For this case we establish necessary and sufficient semantic properties for the distribution of refinement over synchronisation, 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 synchronising 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 synchronisation 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:

$$a + a + a + a \not \cong a + a$$

with the consequence

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

In fact this instance of distribution is ruled out by our conditions (Theorem 18) because the refinement term $a + a$ is not deterministic, and hence not atomic. However, there are many partial order bisimulation relations weaker than $\cong_{\text{a}}$, for instance *history preserving bisimulation* [17, 20], 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 $Q; Q \cong Q; Q \parallel_{L(Q)} (Q \parallel_{Q} Q)$
- $Q$ is atomic if $(Q \parallel_{Q} Q) + (Q \parallel_{Q} Q) \cong (Q \parallel_{Q} Q) \parallel_{L(Q)} (Q \parallel_{Q} Q)$.

In fact this instance of distribution is ruled out by our conditions (Theorem 18) because the refinement term $a + a$ is not deterministic, and hence not atomic. However, there are many partial order bisimulation relations weaker than $\cong_{\text{a}}$, for instance *history preserving bisimulation* [17, 20], 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 $Q; Q \cong Q; Q \parallel_{L(Q)} (Q \parallel_{Q} Q)$
- $Q$ is atomic if $(Q \parallel_{Q} Q) + (Q \parallel_{Q} Q) \cong (Q \parallel_{Q} Q) \parallel_{L(Q)} (Q \parallel_{Q} Q)$.
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. It turns out that this extension is in principle straightforward: if we introduce process constants, defined through recursive guarded systems of equations, and extend the flow event structure semantics and the syntactic constructs (i.e. the sets $L$ etc. and the predicates $v_{det}$ etc.) to the resulting language through fixpoint constructions, then removing refinement does not present any additional difficulties.

Essentially the idea is to reduce terms as before; when however a subterm of the form $X[a \sim Q]$ is encountered, where $X$ is a process name equationally defined as $P$, this is reduced to a new process name $X'$ defined as $P[a \sim Q]$; the latter is then reduced further. When $X[a \sim Q]$ is encountered again, the same $X'$ as before is substituted, thus avoiding an infinite sequence of new action names.

It is not difficult to show that these additional reduction rules preserve the solutions of the corresponding system of equations. Since the system is guarded (and remains so during reduction) it has a unique solution; hence the extended reduction algorithm preserves the semantics, which implies that Theorem 23 and Corollary 24 remain unchanged.

Details of the above constructions are rather technical and hence outside the scope of this paper; they can however be found in the full version [11].

**Related work** The work in [16] 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 approached in the opposite direction in [15]: 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-synchronisation is enough to achieve this.

Syntactic refinement has also been investigated in depth in [1, 2]; the first paper combines it with (CCS) synchronisation. There is however no notion of semantic refinement, and consequently the relation between the two approaches is not considered. Indeed, [1] allows refinements which would contradict the commutativity of diagram (1) in the introduction under common interpretations of semantic refinement. Consider for instance the following CCS variation of Example 5; let $P := (a; c | a; c) | \tau$ and let $\rho: [a \sim b; b]$ 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

\[ P\rho \simeq P^{(\rho(a)/a)} = (b; b; c | b; b; c) | b; b. \]

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
π are defined independently. The main requirement is then that the synchronisation of those refinements satisfies certain constraints. In our setting this idea could be implemented by a rule of the form

\[ (P_1 ||_A P_2)[a \sim Q] \cong (P_1[a \sim Q_1]) ||_{(A \setminus \{a\}) \cup A'} (P_2[a \sim Q_2]) \]

where \( Q_1, Q_2 \) and \( A' \) are such that \( Q_1 ||_{A'} Q_2 \cong Q \). 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 is on finding an SOS operational semantics agreeing with the denotational one, up to history preserving bisimulation. A similar comparison of operational and denotational models, this time up to (essentially) the configuration structure isomorphism \( \cong \), is documented in [18]. 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 \( P[a \sim Q] \) is the transition system with initial state \( \text{red}(P) \{ \text{red}(Q)/a \} \), 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 refifiable symbols and synchronizable actions are explicitly kept disjoint. This means that if \( P[a \sim Q] \) is a term then \( a \) can never be synchronised within \( P \), and our well-formedness criteria are always fulfilled. Hence in this approach, (1) of the introduction commutes: syntactic and semantic refinement always coincide.

References

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

This article was processed using the \LaTeX{} macro package with LLNCS style.