The difference between splitting in *n* and *n+1*

R.J. van Glabbeek, F. Vaandrager

The Difference between Splitting in n and $n+1$

Rob van Glabbeek
Computer Science Department
Stanford University
Stanford, California 94305, USA
rvg@cs.stanford.edu

Frits Vaandrager
CWI
P.O.Box 94079, 1090 GB Amsterdam, The Netherlands
fritsv@cwi.nl

It is established that durational and structural aspects of actions can in general not be modeled in interleaving semantics, even when a time-consuming action is represented by a pair of instantaneous events denoting its start and finish. By means of a series of counterexamples it is shown that, for any n, it makes a difference whether actions are split in n or in $n+1$ parts.

1. INTRODUCTION

In event based models of concurrency which interleave concurrent actions, and thus reduce parallelism to nondeterminism, some rather strong assumptions have to be made about the nature of events. The following formulation of these assumptions is taken from HOARE [15]:

> 'The actual occurrence of each event in the life of an object should be regarded as an instantaneous or an atomic action without duration. Extended or time-consuming actions should be represented by a pair of events, the first denoting its start and the second denoting its finish.' (p. 24)

In interleaving semantics the behavior of a system that performs two actions a and b in parallel is considered the same as the behavior of a system that either does an a followed by a b, or a b followed by an a. Algebraically this leads to equations like

$$a\|b = a{\cdot}b + b{\cdot}a.$$

Such identifications are reasonable under the assumption that the actions a and b are instantaneous, but become highly problematic if a and b are durational or time consuming. Any intuition that in $a\|b$ the a and the b are causally independent and can occur simultaneously, or that $a\|b$ is faster than $a{\cdot}b + b{\cdot}a$, cannot be captured in interleaving semantics in terms of primitive notions. The postulate of interleaving forces one to consider instantaneous actions only. Hoare however, and with him many other advocates of interleaving semantics, argues that this is not really a restriction because one can still deal with durational actions, by *splitting* such actions into a beginning and an end. In interleaving

2

semantics

$$a^+ \cdot a^- \parallel b^+ \cdot b^- \neq a^+ \cdot a^- \cdot b^+ \cdot b^- + b^+ \cdot b^- \cdot a^+ \cdot a^-,$$

because the process on the left has a trace $a^+ b^+ a^- b^-$ which the process on the right has not. Observations like this make many people believe that at least in theory the idea of splitting offers a viable approach to the concept of durational actions.

In this paper we show that the representation, in interleaving models, of durational actions by pairs of actions is *impossible* in a general setting of concurrent and nondeterministic systems. Basically, the problem is that in the presence of a sufficient amount of nondeterminism and concurrency, it is not always clear how start actions match up with finish actions. This confusion leads to the identification of certain processes that should be distinguished by any semantic equivalence which claims to capture the notion of actions with duration. Our counterexamples that illustrate these phenomena are rather complex and don't leap out at the casual observer. PRATT [22] compared this situation with the 19th century engineers who did not notice discrepancies in their day-to-day work due to relativity and quantum mechanics. And indeed, splitting works in all practical situations that we know of. The point which we try to make is that when applications become more complex, this situation may change because splitting is unable to capture the notion of durational actions in general. Given the widespread belief in the universal power of splitting, we proceed in a very careful manner, with the aim to eradicate this misconception in all its possible forms. However, we would like to stress in advance that our results by no means imply that in a setting with durational actions the use of partial order or ST-semantics is 'superior' to a semantics based on interleaving of split actions. In fact we think that in many specific areas of application interleaving+splitting can lead to simpler and more satisfactory solutions. A typical example here is the area of wait-free atomic registers, where apparently the use of interleaving based models in combination with splitting of the durational read and write operations (see for instance [3]), is more successful than the use of the partial order based model of [16].

As it is more general and much simpler, the discussion of this paper will take place at the level of semantic models and not at the level of the languages that are interpreted in those models. We use semantic models which are based on partial orders of events, because in these models we can easily express that actions have duration or structure. In order to argue that a combination of interleaving and splitting cannot capture these features, we need a model which can! A simple way to capture the notion of durational actions in a partial order model is to introduce a mapping which associates a duration to each action. For instance, one can assume that action a takes one hour and action b two hours. Then in $a\|b$ the two actions can be completed in two hours; this is not possible in $a \cdot b + b \cdot a$. Given an execution sequence, one can construct a *real-time execution sequence* by labeling each event with a start time and a finish time, in such a way that (1) the starting times of consecutive events in the sequence are nondecreasing, (2) the start time of each event is greater or equal than the finish time of all its predecessors in the partial order, and (3) for each individual event the difference between finish and start time equals the duration of the action which is associated to the event. We postulate that a treatment of concurrency that claims to capture durational aspects of actions should distinguish between processes with different real-time execution sequences. Partial order models can also capture the concept of actions having structure, via *action refinement*: the operation of refining actions into more complicated processes. Based on arguments from [5, 16, 21], refinement of actions is advocated in [11] as a natural operation on processes, allowing to decrease the level of abstraction at which systems are described. Splitting of actions can be viewed as a simple form of action refinement in which each action a is refined into a process which first does a^+ and then a^-. We postulate that a treatment of concurrency that claims to capture structural aspects of actions should distinguish between processes such as $a\|b$ and $a \cdot b + b \cdot a$ that turn out to be different after refinement of actions. Both requirements are met by suitable approaches based on partial orders [10], but not by those based on interleaving.

Given the complexity of our counterexamples, we decided to split the technical part of this paper

into two parts. In Sections 2 and 3 we discuss the issue in the linear time model of *pomset processes*, and in Sections 4 and 5 we discuss it again in the branching time model of *labeled prime event structures*. We establish that labeled prime event structures can without loss of information be represented by *process graphs*. This representation turns out to be helpful in the analysis of our counterexamples. The results of the latter sections are more general, but also more difficult to understand because the counterexamples have to preserve branching structure too. Each counterexample consists of two systems (represented either as pomset processes or as event structures) which are clearly different if we assume that actions have duration or structure, but which are identified if we first split the events in each system and then compare the interleaving behaviors of the resulting systems.

The first counterexample that we found (the *owl example*) consists of two such systems which are identified in interleaving semantics after splitting each action into two parts, but which have different real-time execution sequences and can be distinguished by splitting each action into three parts. It settles the open problem 'whether digram languages of unit cardinality suffice' posed in GISCHER [6] (p. 11). This may lead a fervent splitter to speculate that although durational and structural properties of actions cannot be modeled in interleaving semantics by simply splitting actions into a beginning and an end, it might still be possible to do so by splitting actions in three or more parts. Therefore we also consider *n*-splitting, where a sequence $a_1 \cdots a_n$ is substituted for each action a, and show that for any $n \geqslant 2$ there are processes with the same interleaving behavior after splitting each action in n parts, but with different real-time execution sequences and different interleaving behavior after each action is split into $n+1$ parts. These examples show that it is impossible to capture durational or structural properties of actions by splitting them into sequences of any finite number of parts.

A possible alternative to splitting actions into sequences is the refinement of actions by processes with a more complicated internal structure, possibly involving conflicts ($+$). We review an example due to VOGLER [24] (a variant of which appears already in K.S. LARSEN [17]) involving two processes that are interleaving equivalent after splitting actions in sequences of arbitrary length, but can be distinguished by refining actions into processes of the form $a_1 \cdot a_2 + a_3 \cdot a_4$. We show that this example works in linear time semantics (in a slightly modified form up to the level of *ready trace semantics* [9]), but does not generalize to branching time semantics. GORRIERI & LANEVE [13] furthermore establish that in *bisimulation semantics* no such example exists.

The auxiliary power of *conflict refinements* (at least in linear time semantics) may be taken as an incentive to propose extensions of the splitting doctrine beyond the level of *sequence refinement*. In fact, GISCHER [6] shows, in essence, that any systems that can be distinguished in linear time interleaving semantics after arbitrary refinements, can already be distinguished by refinements of the form $\Sigma_{i \in I} a_i b_i$, involving an infinite choice. However, our counterexamples show that applying interleaving semantics after any given refinement of actions into processes with a finite number of internal states leads to the identification of systems with a different real-time behavior and a different interleaving semantics after splitting actions in sequences with a greater number of internal states (Section 5.9).

Even though we will not discuss this issue in any detail, we claim that our counterexamples can be recreated on the language level in all major process algebras. In the language *Rec* of [20] for instance, any finite prime event structure can be denoted up to history preserving bisimulation equivalence, which is one of the finest partial order equivalences found in the literature. Since all our counterexamples involve finite structures only, this observation immediately implies that they can be translated to *Rec*. ACETO & HENNESSY [2] discuss in detail the translation of our owl example into Milner's CCS. In [1] they investigated the largest congruence contained in (interleaved) bisimulation equivalence for a subset of CCS without communication and restriction, but with a general refinement operator. They characterized this congruence as the equivalence which declares two processes equal iff after splitting actions into begin and end, the results are (interleaved) bisimilar. That equivalence was originally proposed on a slightly different subset of CCS in HENNESSY [14]; we call it *split bisimulation equivalence*. It follows that our counterexamples do not carry over to the subset of CCS considered by Aceto and Hennessy, and that their characterization does not carry over to full CCS with

refinement. In fact, on the domain of processes that can be expressed in Aceto and Hennessy's language, split bisimulation equivalence coincides with history preserving bisimulation equivalence.

In VAANDRAGER [23] it is proved that for deterministic systems, the ability to observe the beginning and end of events is sufficient for retrieving the causal structure of the systems. Therefore our counterexamples have to be nondeterministic. All counterexamples necessarily involve *autoconcurrency* as well, the possibility that two instances of the same action occur simultaneously. If there is no autoconcurrency, then there can be no confusion about how finish actions match up with start actions, and splitting becomes a consistent way of dealing with durational actions. This is done in [17, 18] on a simple subset of CCS were autoconcurrency is explicitly excluded. The subset of the language *Rec* of [20] which contains injective morphisms only, is a nontrivial example of a language without autoconcurrency. Although we do not exclude that such a restricted language can be useful in practice, our feeling is that autoconcurrency, just like nondeterminism, is a feature which arises naturally if one tries to give abstract descriptions of concurrent systems. Therefore we think that a theory of durational actions will be more useful if it allows for autoconcurrency.

For each interleaving equivalence one can define a *split-n* equivalence, identifying two processes exactly when they are interleaving equivalent after splitting each action into n parts. Our counterexamples show that the split-n equivalences are too course to capture durational and structural aspects of actions. This means that the interleaving equivalences are too course for this purpose even if occurrences of actions are represented by a sequence of two or more events. In a previous paper [12] we introduced a type of equivalence that is very similar to split-2 equivalence, but which in addition requires that the way start actions match up with finish actions is the same for equivalent processes. These so-called *ST-equivalences* are finer than the split-n equivalences but courser then the main partial order equivalences. In [12] and [8] is was established that they do not suffer from the problems we here encounter with split equivalences, and are as suitable for modeling durational actions as are the main partial order equivalences. In ST-semantics as well as in partial order semantics no additional discriminatory power is obtained when actions are split into a beginning and an end (or in more than two parts) before their behaviors are compared. So whereas in interleaving semantics splitting is insufficient, in ST- and in partial order semantics it is unnecessary. In order to capture durational and structural properties of actions one needs to make the distinctions of ST-semantics. The additional distinctions made by partial order equivalences do not harm, but are inessential for this purpose.

2. POMSET PROCESSES

2.1. DEFINITION. A *labeled partial order* or *lpo* over an alphabet A of *actions* is a structure $p = (E, <, l)$ where
- E is a set of *events*;
- $<$ is a partial ordering on E, sometimes referred to as the *causality relation*;
- $l : E \rightarrow A$ is a labeling function which assigns an action to each element of E.

Sometimes we will refer to the components of an lpo p as E_p, $<_p$ and l_p.

Two lpo's p and q are *isomorphic* if there exists an *isomorphism* from p to q, i.e. a bijective mapping f from E_p to E_q which preserves ordering and labels (so $e <_p e' \Leftrightarrow f(e) <_q f(e')$ and $l_p(e) = l_q(f(e))$.

A *partially ordered multiset* or *pomset* is the isomorphism class $[p]$ of an lpo p. We call a pomset *finite* if the lpo's contained in it have a finite number of events.

Let p be an lpo. A set $E \subseteq E_p$ of events is *left-closed* in p if for all $e \in E$ and $e' \in E_p$, $e' <_p e$ implies $e' \in E$.

An lpo p is a *prefix* of an lpo q if E_p is a subset of E_q which is left-closed in q, and moreover $<_p$ and l_p are the restrictions of $<_q$ resp. l_q to E_p. A pomset P is a *prefix* of a pomset Q if some lpo in P is a prefix of some lpo in Q.

The *prefix-closure* $PREF(X)$ of a set of pomsets X is the set consisting of all prefixes of elements of X. We say X is *prefix-closed* if $PREF(X)=X$ (note that, since any pomset is a prefix of itself, it is always the case that $X \subseteq PREF(X)$).

A *pomset process* is a non-empty, prefix-closed set of finite pomsets.

If S is a set of lpo's, then $POMPROC(S)$ denotes the pomset process obtained by taking the prefix-closure of $\{[p] \mid p \in S\}$.

Pomset processes were introduced by PRATT [21]. However, his processes were not required to be nonempty and prefix-closed, and the underlying pomsets did not have to be finite. A pomset process can be used to record the causal relationships between occurrences of actions in a concurrent system. The identity of events is not preserved. Also, due to the prefix-closure property, a pomset process provides no information about the presence or absence of deadlock and infinite behaviors.

2.2. DEFINITION. Let p be an lpo over an alphabet A. A sequence $e_1 \cdots e_n \in (E_p)^*$ is an *event sequence* of p if all events in the sequence are different and for all $i \geqslant 1$ the set $\{e_1, \ldots, e_i\}$ is left-closed.

A sequence $a_1 \cdots a_n \in A^*$ is a *(sequential) trace* of p if there exists an event sequence $e_1 \cdots e_n$ of p with for all i, $l_p(e_i) = a_i$. A sequence σ is a *(sequential) trace* of a pomset P if, for some lpo p in P, σ is a trace of p. Similarly, σ is a *(sequential) trace* of a pomset process X if, for some pomset P of X, σ is a trace of P.

Two pomset processes X and Y are *interleaving trace equivalent*, notation $X \approx_{it} Y$, if they have the same sequential traces.

Notice that if σ is a trace of some lpo in a pomset, it is a trace of all lpo's in that pomset. Also, if σ is a trace of a pomset P which is a prefix of a pomset Q, then σ is a trace of Q. Thus, if S is a set of lpo's, the traces of the associated pomset process $POMPROC(S)$ are exactly the traces of the lpo's in S.

2.3. DEFINITION. Let p be an lpo over A and let $n \geqslant 1$ be a positive natural number. Then $q = split_n(p)$ is the lpo over alphabet $A_n = \{a_i \mid a \in A \text{ and } 1 \leqslant i \leqslant n\}$ given by

$$E_q = \{e_i \mid e \in E_p \text{ and } 1 \leqslant i \leqslant n\},$$

$$e_i <_q f_j \quad \text{iff} \quad e <_p f \text{ or } (e = f \text{ and } i < j),$$

$$l_q(e_i) = (l_p(e))_i.$$

We say that an event e of p has *started* in an event sequence ρ of $split_n(p)$, if e_1 occurs in the sequence. The event e has *finished* if e_n occurs in the sequence. If e has started but not finished, then we say that e is *active* in ρ.

The operation $split_n$ generalizes in the obvious way to pomsets $[p]$ and pomset processes X:

$$split_n([p]) = [split_n(p)],$$

$$split_n(X) = PREF(\{split_n(P) \mid P \in X\}).$$

Two pomset processes X and Y are *split-n trace equivalent*, notation $X \approx_{it}^n Y$, if $split_n(X)$ and $split_n(Y)$ are trace equivalent. We call X and Y *split-ω trace equivalent*, notation $X \approx_{it}^\omega Y$, if they are split-n trace equivalent for all n.

It is immediate from the definitions that $\approx_{it} = \approx_{it}^1$.

2.4. DEFINITION. An *ST-configuration* of an lpo p is a pair (S,T) with $T \subseteq S \subseteq E_p$, and for all $e \in S$ and $e' \in E_p$, $e' <_p e$ implies $e' \in T$.

Intuitively, in an ST-configuration (S,T), the events in S are the ones that have started and the events in T are those that are terminated. Since an event always starts before it terminates, $T \subseteq S$. Moreover an event cannot start before all its causal predecessors are terminated.

2.5. DEFINITION. Let p be an lpo over A. An *ST-sequence* of p is a sequence $e_1 \cdots e_n \in (E_p)^*$ in which each event occurs at most twice, and for all $i \geqslant 0$ the pair

$$(\{e_1, \ldots, e_i\}, \{e \mid e \text{ occurs twice in } e_1 \cdots e_i\})$$

is an ST-configuration of p.

A sequence $(a_1, u_1) \cdots (a_n, u_n) \in (A \times \mathbb{N})^*$ is an *ST-trace* of p if p has an ST-sequence $e_1 \cdots e_n$ such that for all i: $l_p(e_i) = a_i$ and $u_i = \max(\{0\} \cup \{j < i \mid e_j = e_i\})$. A sequence σ is an *ST-trace* of a pomset process X if, for some lpo p in a pomset of X, σ is an ST-trace of p.

Two pomset processes X and Y are *ST-trace equivalent*, notation $X \approx_{STt} Y$, if they have the same sets of ST-traces.

In an ST-sequence $e_1 \cdots e_n$, the first occurrence of an event e denotes that e starts, and the second occurrence denotes that e terminates. If we would only consider the labels of the events in an ST-sequence then, since the identity of the events is not preserved, too much information would get lost. Therefore the actions in an ST-trace are labeled with an additional number. If this number is 0, then the action corresponds to the start of an event. If the number is nonzero, say j, then this means that the action corresponds with the termination of the event whose start is referred to by the j-th action in the sequence.

ST-trace equivalence is the linear time version of the ST-bisimulation equivalence which we introduced in [12] on the domain of Petri nets. The name *ST-bisimulation* originally referred to the fact that this equivalence was defined in terms of markings which incorporate both places (Stellen) and transitions (Transitionen) of a Petri net.

3. CLASSIFICATION OF LINEAR TIME EQUIVALENCES

In this section we establish the relationships between the various notions of equivalence on pomset processes that were introduced in the previous section.

3.1. THEOREM. *For all $n \geqslant 1$, $\approx_{it}^{n+1} \subseteq \approx_{it}^n$.*

PROOF. Suppose X and Y are pomset processes with $X \approx_{it}^{n+1} Y$. We prove $X \approx_{it}^n Y$. For reasons of symmetry it is sufficient to show that any trace of $split_n(X)$ is also a trace of $split_n(Y)$. So let σ be a trace of $split_n(X)$. Then σ is a trace of some lpo $split_n(p)$, where p is an lpo in a pomset of X. Let ρ be an event sequence of $split_n(p)$ which has σ as associated trace. Consider the sequence ρ' which is obtained from ρ by replacing each e_n in ρ (where $e \in E_p$) by the sequence $e_n e_{n+1}$. One can check that ρ' is an event sequence of $split_{n+1}(p)$ and that its associated trace σ' can be obtained from σ by replacing each occurrence of an action a_n by the sequence $a_n a_{n+1}$. Next observe that σ' is a trace of $split_{n+1}(X)$ and thus also of $split_{n+1}(Y)$. This means that σ' is a trace of some lpo $split_{n+1}(q)$, where q is an lpo in a pomset of Y. Let θ' be an event sequence of $split_{n+1}(q)$ which has σ' as associated trace. Since in σ' each a_n is followed immediately by a label a_{n+1}, it must be that in θ' each event e_n is followed immediately by the event e_{n+1}: after e_n, e_{n+1} is the only possible event with label a_{n+1}. The sequence θ which is obtained from θ' by replacing each subsequence $e_n e_{n+1}$ by e_n is an event sequence of $split_n(q)$ and its associated trace is σ. Thus σ is a trace of $split_n(Y)$. $\square$

3.2. Our next aim is to show that all the inclusions of Theorem 3.1 are strict. Figure 1 shows the obvious, trivial and well-known counterexample that illustrates the difference between (split-1) trace equivalence and split-2 trace equivalence.

$$PREF(\{ \quad a \qquad b \quad \}) \quad \approx^{1}_{it} \quad PREF(\{ \quad a \quad , \quad b \quad \})$$

$$\not\approx^{2}_{it} \qquad\qquad\qquad\qquad b \qquad a$$

FIGURE 1: *The difference between splitting in 1 and 2*

Less trivial is the claim that the pomset processes X_2^{even} and X_2^{odd}, which are displayed in Figure 2, are split-2 trace equivalent but not split-3 trace equivalent.

$$X_2^{even} = PREF(\{ \cdots \})$$
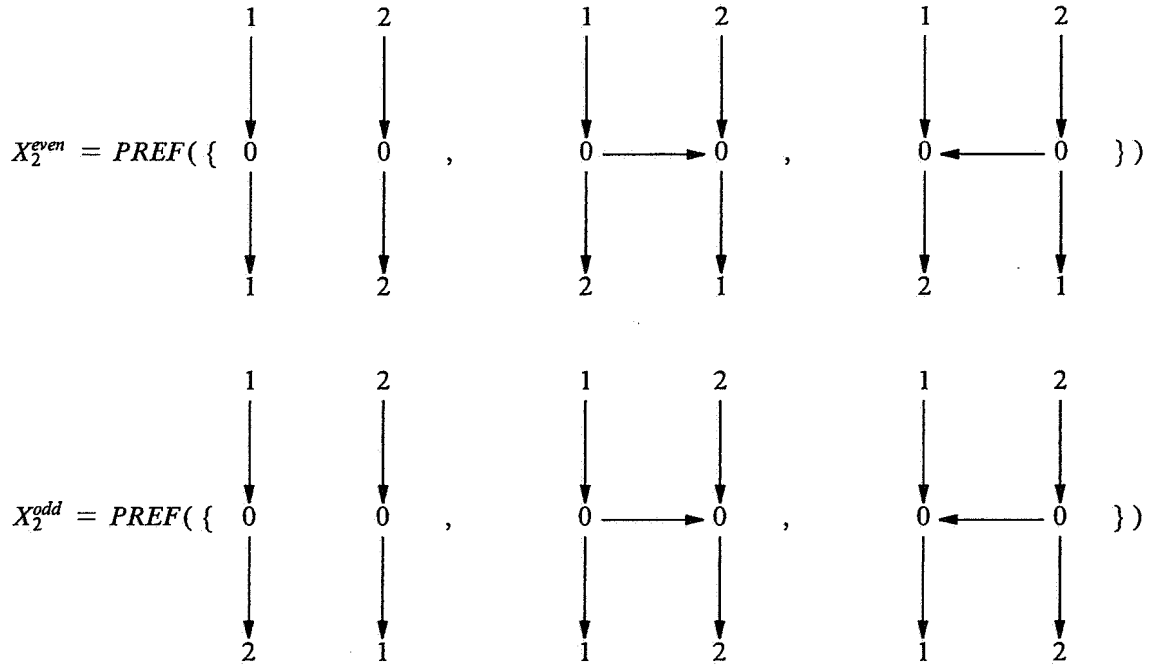
$$X_2^{odd} = PREF(\{ \cdots \})$$

FIGURE 2: *The difference between splitting in 2 and 3*

In fact, we we will show that this example generalizes to a counterexample which distinguishes split-n trace equivalence from split-$n+1$ trace equivalence for all $n \geq 2$. In Definition 3.3 we will define pomset processes X_n^{even} and X_n^{odd}, and in Proposition 3.4 and Theorem 3.5 we will prove that for any $n \geq 2$, X_n^{even} and X_n^{odd} are split-n trace equivalent but not split-$n+1$ trace equivalent.

In order to see that the pomset processes X_2^{even} and X_2^{odd} are different in a setting where actions have duration, assume that action 1 takes 1 time unit, action 2 takes 2 time units, and action 0 takes 3 time units. By choosing the leftmost pomset, process X_2^{odd} can complete the execution of 6 actions in 6 time units. One can easily check that the pomset process X_2^{even} has no possibility to obtain a similar performance.

3.3. DEFINITION. Let $n \geqslant 2$ and let $(s_1, \ldots, s_n)$ be a permutation of $(1, \ldots, n)$. The lpo $p(s_1, \ldots, s_n)$ is defined as follows. Its set of events is

$$\{(i, 0), (0, i), (s_i, i) \mid 1 \leqslant i \leqslant n\},$$

its causality relation $<$ is the least partial ordering relation satisfying for $1 \leqslant i \leqslant n$:

$$(i, 0) < (0, i) < (s_i, i),$$

and its labeling function associates label j to event (j, k).

For $j, k \in \{1, \ldots, n\}$ with with $j \neq k$, the lpo $p_{jk}(s_1, \ldots, s_n)$ is defined in exactly the same way as $p(s_1, \ldots, s_n)$, only now there is one additional causal link:

$$(0, j) < (0, k).$$

We define the pomset process X_n^{even} by

$$X_n^{even} = POMPROC(\{p(s_1, \ldots, s_n) \mid (s_1, \ldots, s_n) \text{ is an even permutation}\} \cup$$

$$\{p_{jk}(s_1, \ldots, s_n) \mid (s_1, \ldots, s_n) \text{ is an odd permutation and } j \neq k\}).$$

Symmetrically, the pomset process X_n^{odd} is defined by

$$X_n^{odd} = POMPROC(\{p(s_1, \ldots, s_n) \mid (s_1, \ldots, s_n) \text{ is an odd permutation}\} \cup$$

$$\{p_{jk}(s_1, \ldots, s_n) \mid (s_1, \ldots, s_n) \text{ is an even permutation and } j \neq k\}).$$

3.4. PROPOSITION. *For all $n \geqslant 2$, the pomset processes X_n^{even} and X_n^{odd} are not split-$n+1$ trace equivalent.*

PROOF. Since $(1, \ldots, n)$ is an even permutation, the following sequence is a trace of $split_{n+1}(p(1, \ldots, n))$, and hence also of $split_{n+1}(X_n^{even})$. Here, as in the rest of this paper, we abbreviate a sequence $a_1 \cdots a_{n+1}$ by a, for a an action.

$$1\ 0_1\ \ldots\ 0_n\ 2\ 0_1\ \ldots\ 0_{n-1}\ 3\ 0_1\ \ldots\ 0_{n-2} \cdots n\ 0_1\ 0_{n+1}\ 1\ 0_n\ 0_{n+1}\ 2\ 0_{n-1}\ 0_n\ 0_{n+1}3 \cdots 0_2\ \ldots\ 0_{n+1}\ n.$$

In the sequence both occurrences of the action i (for $i = 1, \ldots, n$) must stem from the same (i^{th}) component. Therefore the sequence is not a trace of $split_{n+1}(X_n^{odd})$. □

3.5. THEOREM. *For all $n \geqslant 2$, the pomset processes X_n^{even} and X_n^{odd} are split-n trace equivalent.*

PROOF. For reasons of symmetry it is sufficient to show that each trace of $split_n(X_n^{even})$ is also a trace of $split_n(X_n^{odd})$. So suppose σ is a trace of $split_n(X_n^{even})$. If σ is a trace of an lpo $split_n(p_{jk}(s_1, \ldots, s_n))$ for some odd permutation $(s_1, \ldots, s_n)$ and indices j and k, then σ will also be a trace of the lpo $split_n(p(s_1, \ldots, s_n))$ and thus of X_n^{odd}. Otherwise, σ is a trace of an lpo $split_n(p(s_1, \ldots, s_n))$ for some even permutation $(s_1, \ldots, s_n)$. Let ρ be an event sequence of $split_n(p(s_1, \ldots, s_n))$ which has σ as associated trace. If, in ρ, an event $(0, j)$ ends before another event $(0, k)$ starts, then ρ is also an event sequence of the lpo $split_n(p_{jk}(s_1, \ldots, s_n))$ and consequently σ is a trace of $split_n(X_n^{odd})$, and we are done. Also, if ρ is a 'short' event sequence on which not even parts of maximal events (i.e. events (i, j) with both i and j nonzero) occur, then we can easily prove that σ is a trace of $split_n(X_n^{odd})$, by observing that ρ is also an event sequence of $split_n(p(s_n, s_2, \ldots, s_{n-1}, s_1))$; and since swapping two elements of an even permutation leads to an odd permutation this lpo underlies a pomset of $split_n(X_n^{odd})$. The above observations leave as the remaining case one in which ρ is of the form $\theta\eta$ and after θ all events $(0, i)$ (for $1 \leqslant i \leqslant n$) are active. Since each event is split into n parts, and since, after θ, the n events $(0, 1), \ldots, (0, n)$ have all started but not yet finished, it must be that two of these events, say $(0, j)$ and $(0, k)$ with $j < k$, are in the same stage of development. Now let η' be the sequence obtained from η by replacing each sub-event $(i, j)_u$ by $(i, k)_u$ and vice versa. We observe that $\theta\eta'$ is an event sequence of the lpo

$$split_n(p(s_1, \ldots, s_{j-1}, s_k, s_{j+1}, \ldots, s_{k-1}, s_j, s_{k+1}, \ldots, s_n)),$$

which has an associated trace σ. Using again that swapping two elements in an even permutation leads to an odd permutation gives that this lpo underlies a pomset of $split_n(X_n^{odd})$. Thus σ is a trace of $split_n(X_n^{odd})$. $\square$

3.6. Note that similar examples, also validating Proposition 3.4 and Theorem 3.5, are obtained using the pomset processes Y_n^{even} and Y_n^{odd}, given by:

$$Y_n^{even} = POMPROC(\{p(s_1, \ldots, s_n) \mid (s_1, \ldots, s_n) \text{ is an even permutation}\} \cup$$

$$\{p_{jk}(s_1, \ldots, s_n) \mid (s_1, \ldots, s_n) \text{ is any permutation and } j \neq k\}),$$

$$Y_n^{odd} = POMPROC(\{p(s_1, \ldots, s_n) \mid (s_1, \ldots, s_n) \text{ is an odd permutation}\} \cup$$

$$\{p_{jk}(s_1, \ldots, s_n) \mid (s_1, \ldots, s_n) \text{ is any permutation and } j \neq k\});$$

or Z_n and Z'_n, given by:

$$Z_n = POMPROC(\{p(s_1, \ldots, s_n) \mid (s_1, \ldots, s_n) \text{ is any permutation}\},$$

$$Z'_n = POMPROC(\{p(s_1, \ldots, s_n) \mid (s_1, \ldots, s_n) \neq (1, \ldots, n)\} \cup \{p_{jk}(1, \ldots, n) \mid j \neq k\}).$$

3.7. THEOREM. *For all* $n \geq 1$, $\approx_{STt} \subseteq \approx_{it}^n$.
PROOF. Using Theorem 3.1, we can assume w.l.o.g. that $n \geq 2$. Suppose X and Y are pomset processes with $X \approx_{STt} Y$. We prove $X \approx_{it}^n Y$. For reasons of symmetry it is sufficient to show that any trace of $split_n(X)$ is also a trace of $split_n(Y)$. So let σ be a trace of $split_n(X)$. Then σ is a trace of some lpo $split_n(p)$, where p is an lpo in a pomset of X. Let ρ be a corresponding event sequence of $split_n(p)$. Consider the sequence ρ' which is obtained from ρ by replacing events e_1 and e_n in ρ (where $e \in E_p$) by e, and removing all events e_i for $1 < i < n$. Then ρ' is an ST-sequence of p. Let σ' be its associated ST-trace. Then σ' is an ST-trace of X and, consequently, also of Y and of some lpo q in a pomset of Y. Let θ' be an ST-sequence of q which has σ' as associated trace. Then there exists a unique bijective mapping f from the events occurring in ρ' to the events occurring in θ' with the property that the homomorphic extension of this mapping to sequences maps ρ' to θ'. Now let θ be the sequence obtained from ρ by replacing each event e_u by $f(e)_u$. Then θ will be an event sequence of $split_n(q)$. Since f preserves labels, the associated trace of θ is just the associated trace of ρ, i.e. σ. Thus σ is a trace of $split_n(q)$ and hence also of $split_n(Y)$. $\square$

As $\approx_{it}^\omega$ is the limit of the split-n equivalences, it follows immediately from Theorem 3.7 that $\approx_{STt} \subseteq \approx_{it}^\omega$. That also this inclusion is strict follows from the following example, due to VOGLER [24]. A similar example was described earlier by K.S. LARSEN [17].

3.8. DEFINITION. Let p be the lpo with events $\{e, f, g, h\}$, ordering relation $\{(e,g), (e,h), (g,h)\}$, and labeling $\{(e,a), (f,b), (g,b), (h,c)\}$. Let q be the lpo which is identical to p except that (g,h) is replaced by (f,h) in the ordering. Both lpo's p and q are depicted in Figure 3. Here the labels occur as subscripts of the events. The pomset processes X and Y are defined by

$$X = POMPROC(\{p,q\}) \qquad Y = POMPROC(\{q\}).$$

3.9. PROPOSITION. *The pomset processes X and Y are not ST-trace equivalent.*
PROOF. The following sequence is an ST-trace of p but not of q, and hence of X but not of Y:

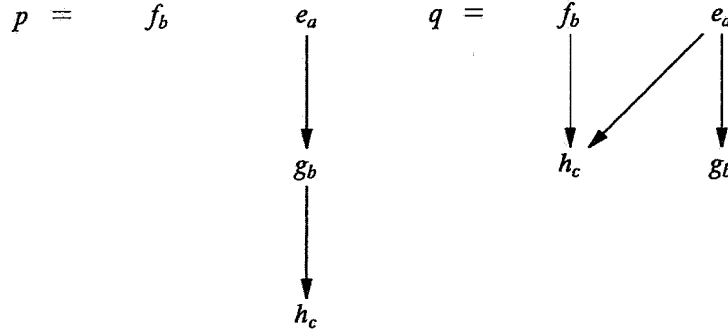$$(a, 0)\ (b, 0)\ (a, 1)\ (b, 0)\ (b, 4)\ (c, 0).$$ $\square$

$$p \;=\; f_b \qquad\qquad e_a \qquad\qquad q \;=\; f_b \qquad\qquad e_a$$

$$g_b \qquad\qquad\qquad\qquad h_c \qquad\qquad g_b$$

$$h_c$$

FIGURE 3: *The difference between ST-trace semantics and arbitrary splitting*

3.10. THEOREM. *For all* $n \geq 1$, *the pomset processes* X *and* Y *are split-n trace equivalent.*

PROOF. It is sufficient to show that any trace of $split_n(p)$ is also a trace of $split_n(q)$. So let σ be a trace of $split_n(p)$ and let ρ be a corresponding event sequence. If it is not the case that in ρ event h starts before event f terminates, then ρ is also an event sequence of $split_n(q)$ and consequently σ is a trace of $split_n(q)$ and we are done. Otherwise, there is a point in ρ where h has started and f is not yet finished. The key observation we can make in this case is that when h starts it must be that g has already finished. Thus, somewhere in ρ the event g 'overtakes' f, in the sense that ρ is of the form $\theta\eta$ and there is a $0 \leq k < n$ such that θ contains e_n, f_i and g_i for all $1 \leq i \leq k$ but not f_{k+1} or g_{k+1}. Therefore, if we swap in ρ, for $k < u \leq n$, the subevents f_u and g_u, we obtain a valid event sequence of lpo $split_n(q)$ which, since f and g both have the same label, corresponds with a trace σ of $split_n(q)$. $\square$

3.11. In order to complete our classification of the equivalences on pomset processes, we recall the counterexample which shows that ST-trace equivalence is different from pomset process equality. Let p_0 be the lpo with two events with labels a resp. b, which are causally unrelated. Let p_1 be identical to p_0, except for the presence of a causal link between the a-event and the b-event. The reader can easily check that $POMPROC(\{p_0\})$ and $POMPROC(\{p_0, p_1\})$ have the same ST-traces, even though they are different as pomset processes.

The final theorem of this section summarizes the relationships between the equivalences on pomset processes.

3.12. THEOREM.

$$\approx_{it} \;=\; \approx_{it}^1 \;\supset\; \approx_{it}^2 \;\supset\; \approx_{it}^3 \;\supset\; \cdots \;\supset\; \approx_{it}^n \;\supset\; \cdots \;\supset\; \approx_{it}^\omega \;\supset\; \approx_{STt} \;\supset\; =.$$

PROOF. By combination of the results of this section. $\square$

4. EVENT STRUCTURES

In order to present the branching time version of our counterexamples we use the model of labeled prime event structures to represent concurrent systems. Prime event structures are introduced in WINSKEL [25] and generalize the better known *prime event structures with a binary conflict relation*, introduced earlier under the name 'event structures' in NIELSEN, PLOTKIN & WINSKEL [19]. The definition of a prime event structure given below is consistent with the one in [25] apart from the fact that in [25] instead of the conflict relation $\sharp$ its complement *Con* (the *consistency relation*) is used.

4.1. DEFINITION. A *labeled prime event structure* (over an alphabet A) is a 4-tuple $\mathbf{E} = (E, <, \sharp, l)$, where

- E is a set of *events*;
- $< \subseteq E \times E$ is a partial order (the *causality relation*) satisfying the principle of *finite causes*:

$$\{e' \in E \mid e' < e\} \text{ is finite for } e \in E;$$

- $\sharp \subseteq \mathcal{P}(E)$ is a set of finite, nonempty, nonsingleton subsets of E (the *conflict relation*) satisfying the principles of *extension*:

$$X \in \sharp \wedge Y \subseteq E \text{ finite} \Rightarrow X \cup Y \in \sharp$$

and *conflict heredity*:

$$X \cup \{e'\} \in \sharp \wedge e' < e \Rightarrow X \cup \{e\} \in \sharp;$$

- $l: E \to A$ is a *labeling function*.

A labeled prime event structure represents a concurrent system in the following way: action names $a \in A$ represent actions the system may perform, an event $e \in E$ labeled with a represents an occurrence of a during a possible run of the system, $e' < e$ means that e' is a prerequisite for e, and $X \in \sharp$ means that the events from X cannot happen together in the same run.

A prime event structure is said to have *binary conflict* if every set $X \in \sharp$ has a subset $Y \in \sharp$ containing only two events. In this case the conflict relation is completely determined by the binary relation $\#$ given by $e \# e' \Leftrightarrow \{e, e'\} \in \sharp$.

From here onwards we leave out the adjectives 'labeled' and 'prime' of our event structures. One usually writes $e' \le e$ for $e' < e \vee e' = e$, $>$ for $<^{-1}$ and $\ge$ for $\le^{-1}$. $X \in \sharp$ is also denoted as $\sharp X$, and in the special case that $X = \{e, e'\}$ also as $e \sharp e'$. The components of an event structure $\mathbf{E}$ will be denoted by respectively $E_{\mathbf{E}}$, $<_{\mathbf{E}}$, $\sharp_{\mathbf{E}}$ and $l_{\mathbf{E}}$. The derived relations will be denoted $\le_{\mathbf{E}}$, $>_{\mathbf{E}}$ and $\ge_{\mathbf{E}}$.

Two event structures $\mathbf{E}$ and $\mathbf{F}$ are *isomorphic*, $\mathbf{E} \cong \mathbf{F}$, iff there exists a bijection between their sets of events preserving $\le$, $\sharp$ and labeling. Generally, we will not distinguish between isomorphic event structures.

In graphical representations of event structures, following [23], the conflict relation is denoted by means of dotted lines between pairs of conflicting events — we only picture event structures with binary conflict — and the causality relation by arrows. We omit causal links derivable by transitivity and conflicts derivable by conflict heredity and extension. Instead of events only their labels are displayed; hence these pictures determine event structures merely up to isomorphism.

4.2. DEFINITION. Let $\mathbf{E}$ be an event structure over some alphabet A. A sequence $e_1 \cdots e_n \in (E_{\mathbf{E}})^*$ is an *event sequence* of $\mathbf{E}$ if all events in the sequence are different, $\{e_1, \ldots, e_n\} \notin \sharp_{\mathbf{E}}$, and whenever e occurs in the sequence and $e' <_{\mathbf{E}} e$ then the occurrence of e is preceded by an occurrence of e'. In that case $\{e_1, \ldots, e_n\}$ is called a *configuration* of $\mathbf{E}$.

An event sequence $e_1 \cdots e_n$ of an event structure $\mathbf{E}$ is *interleaving equivalent* with an event sequence $f_1 \cdots f_m$ of an event structure $\mathbf{F}$ if $n = m$ and for $1 \le i \le n$: $l_{\mathbf{E}}(e_i) = l_{\mathbf{F}}(f_i)$. They are *pomset equivalent* if moreover, for $1 \le i, j \le n$: $e_i <_{\mathbf{E}} e_j \Leftrightarrow f_i <_{\mathbf{F}} f_j$.

Two event structures $\mathbf{E}$ and $\mathbf{F}$ are *interleaving trace equivalent* (*pomset trace equivalent*), notation $\mathbf{E} \approx_{it} \mathbf{F}$ ($\mathbf{E} \approx_{pt} \mathbf{F}$), if for any event sequence of $\mathbf{E}$ there is an interleaving equivalent (pomset equivalent) event sequence of $\mathbf{F}$, and vice versa.

Two event structures E and F are *interleaving bisimulation equivalent* (*history preserving bisimulation equivalent*), notation $E \approx_{ib} F$ ($E \approx_h F$), if there exists a binary relation between the event sequences of E and F (a *bisimulation*), only relating interleaving equivalent (pomset equivalent) event sequences, such that the empty event sequences of E and F are related and if two event sequences are related then each extension of one of them must be related to an extension of the other.

4.3. The equivalences of Definition 4.2 are four extremes in a 2-dimensional classification of semantic equivalences. By definition $\approx_{it} \supset \approx_{ib} \supset \approx_h$ and $\approx_{it} \supset \approx_{pt} \supset \approx_h$. The two bisimulation equivalences are known as *branching time* equivalences. They distinguishing between systems such as the first two event structures of Figure 4, that only differ in the branching point between two different courses of action. The trace equivalences, that do not make such distinctions, are known as *linear time* equivalences. In between there are several *decorated trace equivalences* [9], where part of the branching structure is taken into account.

Pomset trace equivalence and history preserving bisimulation equivalence are *partial order* equivalences. They distinguish between systems such as the last two event structures of Figure 4, that have the same traces but differ in the causal relationships between action occurrences in the corresponding runs. These distinctions are not made by the interleaving equivalences. Again there are various equivalences in between, such as the *ST-equivalences* [8, 12] and the split equivalences that are the subject of this paper.

$$
\begin{array}{c}
a \\
\swarrow \quad \searrow \\
b \cdots\cdots c
\end{array}
\qquad
\begin{array}{c}
\approx_{pt} \\
\not\approx_{ib}
\end{array}
\qquad
\begin{array}{cc}
a\cdots\cdots a \\
\downarrow \qquad \downarrow \\
b \qquad c
\end{array}
\qquad\qquad
a \quad b
\qquad
\begin{array}{c}
\approx_{ib} \\
\not\approx_{pt}
\end{array}
\qquad
\begin{array}{cc}
a\cdots\cdots b \\
\downarrow \qquad \downarrow \\
b \qquad a
\end{array}
$$

$$
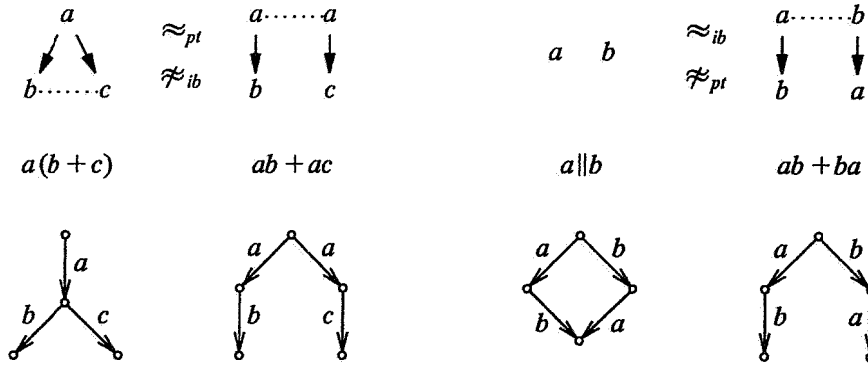a(b+c) \qquad\qquad ab+ac \qquad\qquad a\|b \qquad\qquad ab+ba
$$

FIGURE 4: *Linear time versus branching time and interleaving versus partial orders (standard examples)*

Directly below each of the four event structures in Figure 4 we have given their representations as process expressions. As the relation between process expressions and event structures is not treated in this paper, these serve merely as illustration. Besides, they form a compact notation to refer to these event structures. At the bottom of the figure the four event structures are represented as process graphs. As such representations are useful in understanding our forthcoming examples we will define the translation to process graphs formally.

4.4. DEFINITION. A *process graph* (over an alphabet A) is a triple $g=(S,T,I)$ where
- S is a set of *states* or *nodes*,
- $T\subseteq S\times A\times S$ is a set of *transitions* or *edges*,
- $I\in S$ is the *initial state* or *root*.

The process graph $g(E)$ *associated* to an event structure E is $(C_E, T_E, \emptyset)$ where C_E is the set of configurations of E and $(X,a,Y)\in T_E$ iff $X\subset Y$ and a is the label of the unique event in $Y-X$.

Two process graphs are *isomorphic* iff there exists a bijection between their sets of states preserving transitions and the initial state. Again, we will not distinguish between isomorphic process graphs.

Process graphs are mostly used in the context of interleaving semantics, where the rightmost two processes of Figure 4 are identified. However, as pointed out in [7], there is no need for this restriction. Process graphs have enough structure to express causality and all other features expressible by event structures. This is illustrated by the last two process graphs of Figure 4: a square, as in $a\|b$, denotes concurrency, whereas branching, as in $ab + ba$, denotes conflict. That this feature makes process graphs at least as expressive as event structures follows from the following theorem.

4.5. THEOREM. *The mapping ϑ from event structures to process graphs is injective.*
PROOF. In order to show injectivity we construct a map $\mathcal{S}$ from process graphs to event structures, such that every event structure $\mathbf{E}$ is isomorphic to $\mathcal{S}(\vartheta(\mathbf{E}))$.

Let g be a process graph. If (p,a,q) is a transition in g we write $p \xrightarrow{a} q$. Let $\sim$ be the smallest equivalence relation on the transitions of g satisfying

$$p \xrightarrow{a} q \xrightarrow{b} s, \; p \xrightarrow{b} r \xrightarrow{a} s, \; q \neq r \Rightarrow (p,a,q) \sim (r,a,s).$$

Note that the transitions (p,a,q) and (r,a,s) above are 'opposites in a square'. As squares denote concurrency, these edges must be understood to represent the same action occurrence. Therefore we call the equivalence classes w.r.t. $\sim$ *events*. A transition in such an equivalence class is said to *represent* that event and $[p,a,q]$ denotes the event represented by transition (p,a,q). A *path* in g is a connected sequence of transitions $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} p_n$ starting in the root of g. We define $\mathcal{S}(g)$ to be the event structure $(E, <, \sharp, l)$ where E is the set of events of g, l is the function associating to each event the (unique) label of its representatives, $e' < e$ for $e, e' \in E$ iff $e \neq e'$ and all paths that contain a representative of e also contain a representative of e', and for X a finite, nonempty, nonsingleton subset of E, $\sharp X$ iff g does not have a path containing representatives of all events in X.

Now let $\mathbf{E}$ be an event structure. We construct an isomorphism between $\mathbf{F} = \mathcal{S}(\vartheta(\mathbf{E}))$ and $\mathbf{E}$. Note that if $(X,a,Y) \sim (X',a,Y')$ then $Y - X = Y' - X'$ and the unique event in $Y - X$ has label a. Let $f : E_\mathbf{F} \to E_\mathbf{E}$ be the function that associates to $[X,a,Y]$ the unique event in $Y - X$. In order to establish that f is an isomorphism we need the following lemma.

DEFINITION. Let X be a finite set of events of an event structure $\mathbf{E}$. The *left-closure* of X is the set $\downarrow X = \{e' \in E_\mathbf{E} \mid e' \leqslant_\mathbf{E} e \in X\}$. For $e \in E_\mathbf{E}$ we abbreviate $\downarrow\{e\}$ by $\downarrow e$. X is *left-closed* if $\downarrow X = X$. X is *conflict-free* if $X \notin \sharp_\mathbf{E}$.

LEMMA. Let $\mathbf{E}$ be an event structure and let Y be a finite subset of $E_\mathbf{E}$. Then
1. If $Y \notin \sharp_\mathbf{E}$, then $\downarrow Y$ is a configuration of $\mathbf{E}$.
2. Every event in $E_\mathbf{E}$ occurs in an event sequence of $\mathbf{E}$.
PROOF. Suppose $Y \notin \sharp_\mathbf{E}$. Note that a finite set of events is a configuration iff it is left-closed and conflict-free. As $\downarrow Y$ is left-closed by definition, we only have to check conflict-freeness. Suppose that $\downarrow Y \in \sharp_\mathbf{E}$. Take $e \in \downarrow Y - Y$. As Y contains an event e' with $e <_\mathbf{E} e'$ the principle of conflict heredity (with $X = \downarrow Y - \{e\}$) implies $\downarrow Y - \{e\} \in \sharp_\mathbf{E}$. Repeating this argument yields $Y \in \sharp_\mathbf{E}$, contradicting the assumptions. The second statement follows from the first because singleton sets cannot be in conflict. $\square$

We now check that f is an isomorphism:
- f is surjective by part 2 of the lemma.
- Suppose $f(e) = f(e')$ for $e, e' \in E_\mathbf{F}$. Let $e = [X,a,Y]$ and $e' = [X',a,Y']$. Note that $X = Y - \{f(e)\}$. Suppose that Y contains an event $d \notin \downarrow f(e)$ that is maximal in Y w.r.t. $<_\mathbf{E}$. Then $\mathbf{E}$ has configurations $X - \{d\}$ and $Y - \{d\}$ and $(X - \{d\}, a, Y - \{d\}) \sim (X,a,Y)$. Repeating this argument yields

$$(X,a,Y) \sim (\downarrow f(e) - \{f(e)\}, a, \downarrow f(e)) = (\downarrow f(e') - \{f(e')\}, a, \downarrow f(e')) \sim (X',a,Y').$$

This implies that f is injective.

- By construction we have $l_E(f(e))=l_F(e)$, $f(e')<_E f(e)\Rightarrow e'<_F e$ and $\#_E(f(X))\Rightarrow\#_F(X)$.
- $f(X)\notin\#_E$ implies $X\notin\#_F$ by part 1 of the lemma.
- Suppose that not $f(e')<_E f(e)$. Then $\downarrow f(e)\in C_E$ by the lemma, contradicting $e'<_F e$. $\qquad\square$

Modeling concurrency by means of squares (or cubes, hypercubes etc. in case of three or more concurrent actions) is particularly useful when actions are thought to have a duration or structure. A concurrent execution of a and b in the process $a\|b$ can now be thought of as a continuous path through the surface of the square, starting at the top and terminating at the bottom node, while being nondecreasing when projected on any edge. This makes splitting of actions an easy to visualize operation on the graph representations of event structures, as will be illustrated below.

4.6. DEFINITION. Let E be an event structure over alphabet A and let $n\geqslant 1$. The event structure $F = split_n(E)$ over alphabet $A_n = \{a_i\,|\,a\in A$ and $1\leqslant i\leqslant n\}$ is defined by

$$E_F = \{e_i\,|\,e\in E_E \text{ and } 1\leqslant i\leqslant n\},$$

$$e_i <_F f_j \quad\text{iff}\quad e <_E f \text{ or } (e = f \text{ and } i < j),$$

$$X \in \#_F \quad\text{iff}\quad origin(X) \in \#_E, \text{ where } origin(e_i) = e \text{ and } origin(X) = \{origin(f)\,|\,f\in X\},$$

$$l_F(e_i) = (l_E(e))_i.$$

As before, we say that an event e of E has *started* in an event sequence or configuration X of $split_n(E)$ if e_1 occurs in X. The event e has *finished* if e_n occurs in X. If e has started but not finished, then we say that e is *active* in X.

Two event structures E and F are *split-n trace equivalent*, notation $E \approx^n_{it} F$, if $split_n(E) \approx_{it} split_n(F)$. They are *split-n bisimulation equivalent*, $E \approx^n_{ib} F$, if $split_n(E) \approx_{ib} split_n(F)$. *Split-$\omega$ trace (resp. bisimulation) equivalence* is again defined to be the intersection of $\approx^n_{it}$ (resp. $\approx^n_{ib}$) for all n.

In the same way one could define *split-n pomset trace equivalence* and *split-n history preserving bisimulation equivalence*, but as pomset trace and history preserving bisimulation equivalence are preserved under refinement of actions [10] these notions would coincide with $\approx_{pt}$ and $\approx_h$, respectively.

4.7. EXAMPLE. Below the operation $split_2$ is applied to an event structure as well as to its associated process graph. We do not formally define splitting on graphs; the graph obtained is just the process graph associated to the split event structure. However, it helps thinking of the operation as filling in squares with quadrants, etc.
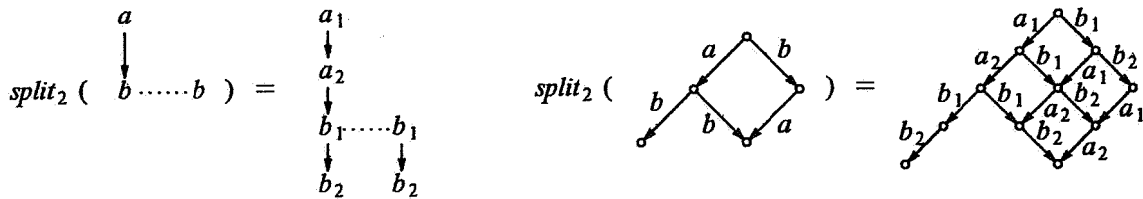

FIGURE 5: *Splitting an event structure and its corresponding process graph*

4.8. DEFINITION. Let E be an event structure over some alphabet A. A sequence $e_1 \cdots e_n \in (E_E)^*$ is an *ST-sequence* of E if each event occurs at most twice in the sequence, $\{e_1, \ldots, e_n\}\notin\#$, and if e occurs in the sequence and $e'<_E e$ then the first occurrence of e is preceded by two occurrences of e'.

A sequence $e_1 \cdots e_n$ of events from an event structure E is *ST-equivalent* with a sequence $f_1 \cdots f_m$ of events from an event structure F if $n=m$, for $1\leqslant i\leqslant n$: $l_E(e_i)=l_F(f_i)$ and for $1\leqslant i,j\leqslant n$:

$$e_i = e_j \Leftrightarrow f_i = f_j.$$

Two event structures E and F are *ST-trace equivalent*, notation E $\approx_{STt}$ F, if for any ST-sequence of E there is an ST-equivalent ST-sequence of F, and vice versa.

Two event structures E and F are *ST-bisimulation equivalent*, notation E $\approx_{STb}$ F, if there exists a binary relation (an *ST-bisimulation*) between the ST-sequences of E and F, only relating ST-equivalent ST-sequences, such that the empty ST-sequences of E and F are related and if two ST-sequences are related then each extension of one of them must be related to an extension of the other.

It is left as an easy exercise for the reader to verify that our Definitions 4.2 and 4.8 agree with the more configuration oriented ones in [10] and [8].

4.9. Event structures classify as a *branching time* model of concurrency, because systems like $a(b+c)$ and $ab+ac$ in Figure 4 have different representations. This gives us the freedom to divide out either a linear time or a branching time equivalence. Pomset processes on the other hand constitute a *linear time* model of concurrency: systems like $a(b+c)$ and $ab+ac$ have the same representation, and consequently there is no option to consider branching time equivalences. The relation between event structures and pomset processes is formalized by the canonical translation *POM* from event structures to pomset processes, defined by

$$POM(\mathbf{E}) = \{[p] \mid E_p \text{ is a configuration of } \mathbf{E}, \ <_p = <_E \restriction E_p, \ l_p = l_E \restriction E_p\}.$$

Note that $POM(a(b+c)) = POM(ab+ac)$ whereas $POM(a\|b)$ and $POM(ab+ba)$ are the two pomset processes of Figure 1. Under this translation the linear time equivalences on event structures correspond exactly with those on pomset processes:

4.10. PROPOSITION. *Two event structures E and F are interleaving trace, split-n trace, split-ω trace, respectively ST-trace equivalent, iff the associated pomset processes POM(E) and POM(F) are. They are pomset trace equivalent iff POM(E)=POM(F).*
PROOF. Fairly straightforward. As this result is not crucial for the main contributions of this paper we leave the elaboration to the reader. □

5. CLASSIFICATION OF BRANCHING TIME EQUIVALENCES
In this section the equivalences introduced in Section 4 will be ordered by inclusion. The results are summarized in Figure 6:

$$
\begin{array}{ccccccccccc}
\approx_{ib} & = & \approx_{ib}^{1} & \supset & \approx_{ib}^{2} & \supset & \cdots & \supset & \approx_{ib}^{\omega} & \supseteq & \approx_{STb} & \supset & \approx_{h} \\
\cap & & \cap & & \cap & & & & \cap & & \cap & & \cap \\
\approx_{it} & = & \approx_{it}^{1} & \supset & \approx_{it}^{2} & \supset & \cdots & \supset & \approx_{it}^{\omega} & \supset & \approx_{STt} & \supset & \approx_{pt}
\end{array}
$$

FIGURE 6: *Semantic equivalences*

Note that the bottom row of this figure is a restatement of Theorem 3.12, thanks to Proposition 4.10. We will prove these results again however, since, with one exception, they are easy corollaries from the lemma's we need to establish for the classification of branching time equivalences in the top row of Figure 6. At the end of this section we will discuss the relationships between the split-ω equivalences and the ST-equivalences.

5.1. THEOREM. *For $n \geq 1$, $\approx_{it}^{n+1} \subseteq \approx_{it}^{n}$ and $\approx_{ib}^{n+1} \subseteq \approx_{ib}^{n}$.*
PROOF. Let $n \geq 1$. In this proof, for any event structure E, let π_E be the function on sequences of events of $split_n(\mathbf{E})$, replacing each event e_n by the sequence $e_n e_{n+1}$. Before we come to the main argument of this proof we establish three lemmas.

EDGE LEMMA. σ is an event sequence of $split_n(E)$ iff $\pi_E(\sigma)$ is an event sequence of $split_{n+1}(E)$.
PROOF. Straightforward. Another edge lemma will be proved in Section 5.2. □

SQUARE LEMMA. Let E and F be event structures, let α and β be interleaving equivalent event sequences of $split_{n+1}(E)$ and $split_{n+1}(F)$ and let σ be an event sequence of $split_n(E)$ with $\pi_E(\sigma) = \alpha$. Then there exists exactly one event sequence ρ of $split_n(F)$ which is interleaving equivalent with σ and for which $\pi_F(\rho) = \beta$.

PROOF. Let $\alpha = e^1 \cdots e^k$ and $\beta = f^1 \cdots f^k$. Obtain ρ as $\phi(\sigma)$, where ϕ is the function on sequences of events that replaces each occurrence of e^h by f^h $(h = 1, \ldots, k)$. Since in α, and hence in β, each event with label a_n is immediately followed by an event with label a_{n+1}, it follows (by induction on the length of β) that in β each event e_n is immediately followed by e_{n+1} (with the same e). Therefore, $\pi_F(\phi(\sigma)) = \phi(\pi_E(\sigma))$. As a consequence $\pi_F(\rho) = \beta$ and hence, by the edge lemma, ρ is an event sequence of $split_n(F)$. By construction σ and ρ are interleaving equivalent.

Since, for any ρ', $\pi_F(\rho') = \beta$ implies that ρ' can be obtained from β by leaving out all events of the form f_{n+1}, ρ is the only sequence ρ' with $\pi_F(\rho') = \beta$. □
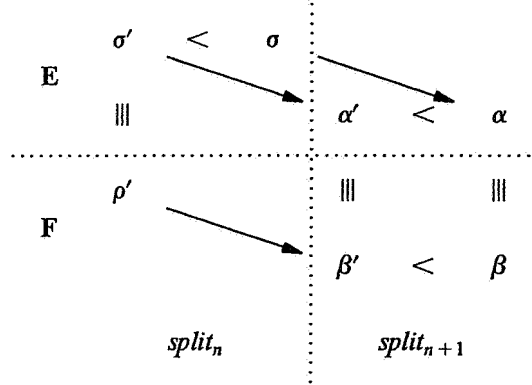

FIGURE 7: *The cube of the cube lemma*

CUBE LEMMA. Let E and F be event structures, let α and α' be event sequences of $split_{n+1}(E)$, β and β' be event sequences of $split_{n+1}(F)$, σ and σ' be event sequences of $split_n(E)$ and ρ' be an event sequence of $split_n(F)$. Furthermore let α' be a prefix of α, β' of β and σ' of σ; let α be interleaving equivalent with β, α' with β' and σ' with ρ'; and let $\pi_E(\sigma) = \alpha$, $\pi_E(\sigma') = \alpha'$ and $\pi_F(\rho') = \beta'$. Then there is an event sequence ρ of $split_n(F)$ which is interleaving equivalent with σ, an extension of ρ', and for which $\pi_F(\rho) = \beta$.

PROOF. Note that the aim of this proof is to complete the cube of Figure 7. By the square lemma there is an event sequence ρ of $split_n(F)$ which is interleaving equivalent with σ and for which $\pi_F(\rho) = \beta$. Let ϕ be as in the proof of the square lemma, then $\rho = \phi(\sigma)$ is an extension of $\phi(\sigma')$. By construction $\phi(\sigma')$ is an event sequence of $split_n(F)$ which is interleaving equivalent with σ'. Moreover, just as in the proof of the square lemma, $\pi_F(\phi(\sigma')) = \phi(\pi_E(\sigma')) = \beta'$. Another application of the square lemma gives $\phi(\sigma') = \rho'$. □

For the proof of $\approx_{it}^{n+1} \subseteq \approx_{it}^n$, let E and F be event structures with $E \approx_{it}^{n+1} F$, and let σ be an event sequence in $split_n(E)$. Then $\alpha = \pi_E(\sigma)$ is an event sequence of $split_{n+1}(E)$, and hence there must be an interleaving equivalent event sequence β in $split_{n+1}(F)$. Now the square lemma yields an event sequence ρ of $split_n(F)$ which is interleaving equivalent with σ. The 'vice versa' follows by symmetry and therefore $E \approx_{it}^n F$. □

For the proof of $\approx_{ib}^{n+1} \subseteq \approx_{ib}^{n}$, let E and F be event structures with $E \approx_{ib}^{n+1} F$, and let R be a bisimulation between the event sequences of $split_{n+1}(E)$ and $split_{n+1}(F)$. Define the relation R' between the event sequences of $split_n(E)$ and $split_n(F)$ by

$$\sigma R' \rho \quad \Leftrightarrow \quad \sigma \text{ and } \rho \text{ are interleaving equivalent and } \pi_E(\sigma) R \pi_F(\rho).$$

The empty event sequences are related by R' since they are related by R. Now suppose that $\sigma' R' \rho'$ and σ is an extension of σ'. Define $\alpha' = \pi_E(\sigma')$, $\beta' = \pi_F(\rho')$ and $\alpha = \pi_E(\sigma)$. Then $\alpha' R \beta'$ and α is an extension of α'. Since R is a bisimulation, there must be an extension β of β' with $\alpha R \beta$. According to the cube lemma there must be an extension ρ of ρ' that is interleaving equivalent with σ and for which $\pi_F(\rho) = \beta$. By definition $\sigma R' \rho$. The 'vice versa' follows by symmetry and therefore R' is a bisimulation and $E \approx_{ib}^{n} F$. $\square$

By definition we have, for $n \geq 1$, $\approx_{it}^{\omega} \subseteq \approx_{it}^{n}$ and $\approx_{ib}^{\omega} \subseteq \approx_{ib}^{n}$.

5.2. THEOREM. *For $n \geq 2$, $\approx_{STi} \subseteq \approx_{it}^{n}$ and $\approx_{STb} \subseteq \approx_{ib}^{n}$.*
PROOF. Let $n \geq 2$. A sequence of events in an event structure $split_n(E)$ is *well-formed* if no event occurs twice in the sequence and each occurrence of an event of the form e_{i+1} for certain $e \in E_E$ and $1 \leq i \leq n-1$ is preceded by an occurrence of e_i. Note that each event sequence of $split_n(E)$ is a well-formed sequence of events. Let in this proof, for any event structure E, π_E be the function on well-formed sequences of events of $split_n(E)$ that leaves out all events of the form e_i for $e \in E_E$ and $2 \leq i \leq n-1$ and replaces each event of the form e_1 or e_n by e. We again prove three lemmas.

EDGE LEMMA. *A well-formed σ is an event sequence of $split_n(E)$ iff $\pi_E(\sigma)$ is an ST-sequence of E.*
PROOF. '$\Rightarrow$': Suppose σ is an event sequence of $split_n(E)$. Since all events in σ are different, no event occurs more than twice in $\pi_E(\sigma)$. An event e occurs in $\pi_E(\sigma)$ only if e_1 occurs in σ. Hence the set of events occurring in $\pi_E(\sigma)$ must be conflict-free. Now let e occur in $\pi_E(\sigma)$ and $e' <_E e$. The first occurrence of e in $\pi_E(\sigma)$ must originate from an occurrence of e_1 in σ. Since $e'_i <_{split_n(E)} e_1$, this occurrence of e_1 is preceded by an occurrence of e'_1 as well as an occurrence of e'_n. Hence in $\pi_E(\sigma)$ the first occurrence of e is preceded by two occurrences of e'.

'$\Leftarrow$': Suppose $\pi_E(\sigma)$ is an ST-sequence of E and assume σ is a well-formed sequence of events of $split_n(E)$. Then no event occurs twice in σ, and an event e_i occurs in σ only if e occurs in $\pi_E(\sigma)$. Hence the set of events occurring in σ must be conflict-free. Now let e_j occur in σ and $e'_i <_{split_n(E)} e_j$. Then $e' <_E e$ or ($e' = e$ and $i < j$). In the latter case e_i occurs in σ by the requirement of well-formedness. In the former case e_1 occurs in σ by well-formedness, so e occurs in $\pi_E(\sigma)$, e' occurs twice in $\pi_E(\sigma)$, e'_n occurs in σ and by well-formedness e'_i occurs in σ. $\square$

DEFINITION. Let E and F be event structures. An event sequence $e^1 \cdots e^k$ of $split_n(E)$ is *ST-split-equivalent* with an event sequence $f^1 \cdots f^l$ of $split_n(F)$ if $k = l$, for $1 \leq h \leq k$: $l_{split_n(E)}(e^h) = l_{split_n(F)}(f^h)$ and for $1 \leq g, h \leq k$: $origin(e^g) = origin(e^h) \Leftrightarrow origin(f^g) = origin(f^h)$.

SQUARE LEMMA. *Let E and F be event structures, let α and β be ST-equivalent ST-sequences of E and F and let σ be an event sequence of $split_n(E)$ with $\pi_E(\sigma) = \alpha$. Then there exists exactly one event sequence ρ of $split_n(F)$ which is ST-split-equivalent with σ and for which $\pi_F(\rho) = \beta$.*
PROOF. Let $\alpha = e^1 \cdots e^k$ and $\beta = f^1 \cdots f^k$. Obtain ρ as $\phi(\sigma)$, where ϕ is the function on sequences of events that replaces each occurrence of e^h by f^h and of e_i^h by f_i^h ($h = 1, ..., k$; $i = 1, ..., n$). Then $\pi_F(\phi(\sigma)) = \phi(\pi_E(\sigma))$. As a consequence $\pi_F(\rho) = \beta$ and hence, by the edge lemma, ρ is an event sequence of $split_n(F)$. By construction σ and ρ are ST-split-equivalent.

Suppose ρ' is another event sequence of $split_n(F)$ which is ST-split-equivalent with σ and for which $\pi_F(\rho') = \beta$. Let d be the first event in ρ that differs from the corresponding event d' in ρ'. If either d or d' is of the form f_i with $i > 1$, a contradiction is obtained with the observation that ρ and ρ' are ST-split-equivalent. If both are of the form f_1 a contradiction is obtained with $\pi_F(\rho) = \pi_F(\rho')$. $\square$

CUBE LEMMA. *Let* E *and* F *be event structures, let* α *and* α' *be ST-sequences of* E, β *and* β' *be ST-sequences of* F, σ *and* σ' *be event sequences of* $split_n(E)$ *and* ρ' *be an event sequence of* $split_n(F)$. *Furthermore let* α' *be a prefix of* α, β' *of* β *and* σ' *of* σ; *let* α *be ST-equivalent with* β, α' *with* β' *and* σ' *with* ρ'; *and let* $\pi_E(\sigma) = \alpha$, $\pi_E(\sigma') = \alpha'$ *and* $\pi_F(\rho') = \beta'$. *Then there is an event sequence* ρ *of* $split_n(F)$ *which is ST-split-equivalent with* σ, *an extension of* ρ', *and for which* $\pi_F(\rho) = \beta$.
PROOF. Exactly as the one of the previous cube lemma. □

The remainder of the proof of Theorem 5.2 proceeds exactly as the proof of Theorem 5.1. □

COROLLARY. $\approx_{STt} \subseteq \approx_{it}^{\omega}$ *and* $\approx_{STb} \subseteq \approx_{ib}^{\omega}$. □

5.3. THEOREM. $\approx_{pt} \subseteq \approx_{STt}$ *and* $\approx_h \subseteq \approx_{STb}$.
PROOF. Let for any event structure E, π_E be the function on sequences of events, that omits for every event its second occurrence. Apart from the edge lemma, the proof goes along the same lines as the previous ones.

WEAK EDGE LEMMA. *Let* σ *and* ρ *be ST-equivalent sequences of events of event structures* E *and* F *respectively, such that* $\pi_E(\sigma)$ *and* $\pi_F(\rho)$ *are pomset equivalent event sequences. Then* σ *is an ST-sequence of* E *iff* ρ *is an ST-sequence of* E.
PROOF. Let $\sigma = e^1 \cdots e^n$ and $\rho = f^1 \cdots f^n$. Clearly each event occurs at most twice in σ and in ρ and $\{e^1, \ldots, e^n\} \notin \#_E$, $\{f^1, \ldots, f^n\} \notin \#_F$. As σ and ρ are ST-equivalent, second occurrences of events in the one occur at the same position as second occurrences of events in the other. Hence, two events $e \in E_E$ and $f \in E_F$ occur in the same position in σ and ρ iff they occur in the same position in $\pi_E(\sigma)$ and $\pi_F(\rho)$. Thus also for σ and ρ we have that for $1 \leqslant i,j \leqslant n$: $e_i <_E e_j \Leftrightarrow f_i <_F f_j$. Now suppose σ is an ST-sequence of E. Let f occur in ρ, say its first occurrence is in position k, and $f' <_F f$. Then f occurs in $\pi_F(\rho)$, and since $\pi_E(\rho)$ is an event sequence also f' occurs in it, prior to f. Hence f' occurs in ρ, say in position $i < k$. It follows that $e^i <_E e^k$ and hence e^k is preceded by two occurrences of e^i, say in positions $i,j < k$. As σ and ρ are ST-equivalent, f^i and f^j are the two occurrences of f' that precede the first occurrence of f. □

SQUARE LEMMA. *Let* E *and* F *be event structures, let* α *and* β *be pomset equivalent event sequences of* E *and* F *and let* σ *be an ST-sequence of* E *with* $\pi_E(\sigma) = \alpha$. *Then there exists exactly one ST-sequence* ρ *of* F *which is ST-equivalent with* σ *and for which* $\pi_F(\rho) = \beta$.
PROOF. Let $\alpha = e^1 \cdots e^k$ and $\beta = f^1 \cdots f^k$. Obtain ρ as $\phi(\sigma)$, where ϕ is the function on sequences of events that replaces each occurrence of e^h by f^h ($h = 1, \cdots, k$). By construction σ and ρ are ST-equivalent and $\pi_F(\phi(\sigma)) = \phi(\pi_E(\sigma))$. As a consequence $\pi_F(\rho) = \beta$ and hence, by the weak edge lemma, ρ is an ST-sequence of F.

Suppose ρ' is another event sequence of F which is ST-equivalent with σ and for which $\pi_F(\rho') = \beta$. Let d be the first event occurrence in ρ that differs from the corresponding event occurrence d' in ρ'. If either d or d' is a second occurrence of an event, a contradiction is obtained with the observation that ρ and ρ' are ST-equivalent. If both are first occurrences, a contradiction is obtained with $\pi_F(\rho) = \pi_F(\rho')$. □

The cube lemma and the remainder of the proof of Theorem 5.3 go exactly as in the previous two cases. □

5.4. Thus we established all horizontal inclusions of Figure 6. The vertical ones follow immediately from the definitions. It remains to show that all inclusions, with the possible exception of $\approx_{STb} \subseteq \approx_{ib}^{\omega}$, are strict, and that there are no further inclusions. In the vertical direction this follows from the processes $a(b+c)$ and $ab+ac$ of Figure 4, that separate the trace equivalences from the bisimulation equivalences. Furthermore, the processes $a\|b$ and $ab+ba$ of Figure 4 are split-2 trace

distinguishable, and therefore separate the interleaving ($=$ split-1) equivalences from the split-n equivalences (for $n \geqslant 2$). The process $a \| b$ is ST-bisimulation equivalent to

$$a$$
$$\downarrow$$
$$b \cdots\cdots b$$

but the two are not pomset trace equivalence. This example separates the partial order equivalences from the ST-equivalences. Note that applying the operator *POM* of Section 4.9 to these processes yields the two pomset processes of Section 3.11. We now come to the main contribution of this paper, namely a parametrized example separating the split-n equivalences from the split-$n + 1$ equivalences, for $n \geqslant 2$.

In Figure 8 two event structures **E** and **F** can be found that are split-2 bisimulation equivalent, but not split-3 trace equivalent (and therefore also not split-3 bisimulation equivalent).
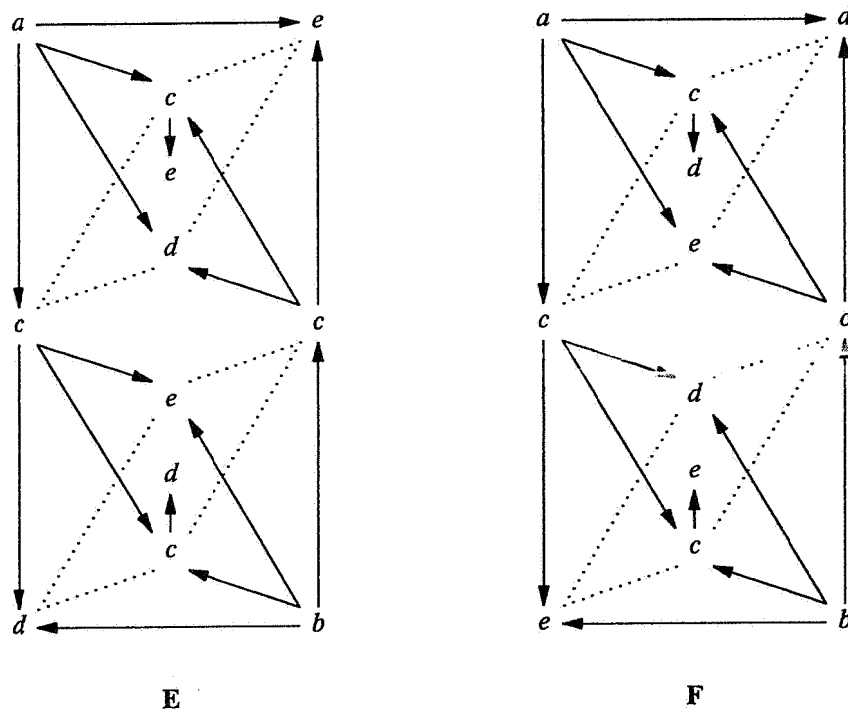

E   F

FIGURE 8: *Event structure representation of the owl example*

Although these event structures are esthetically pleasing, they are not as easy to analyze as their associated process graphs, displayed in Figure 9. These show the example in the shape we found it. As in Proposition 3.4, after splitting in 3, only the first process has a trace $a \, c_1 \, c_2 \, b \, c_1 \, c_3 \, d \, c_2 \, c_3 \, e$ and allocating 1 time unit to actions a, d and e, 2 units to action b and 3 units to action c shows a difference in their real-time behavior.

In Figure 10 one sees what happens to these processes when they are split in 2. Any step of the one process can be matched in a bisimulation by an identical step of the other, until both processes pass through one of the black nodes in the middle. Suppose both processes arrive at their leftmost black node. From there on their futures look entirely identical. Any move made in the body of one owl is mimicked by a move in the wing of the other, and vice versa. The same holds for the black nodes on the right. Finally, if both processes arrive at the black node in the middle, their futures are identical
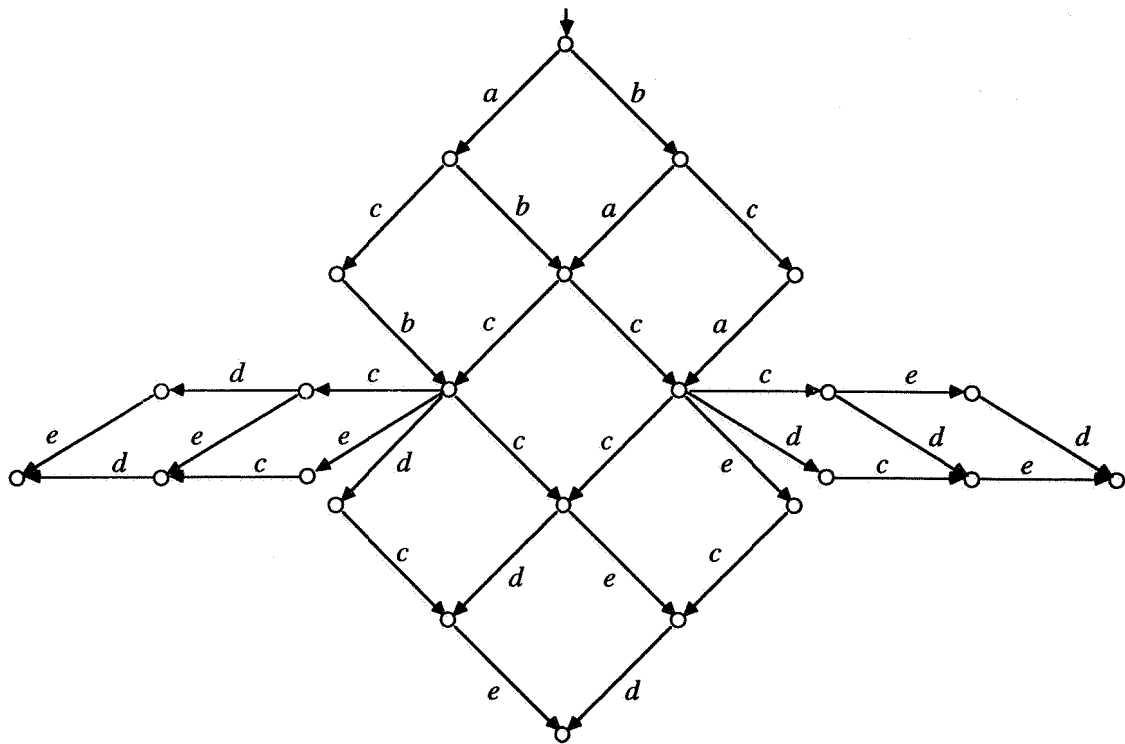
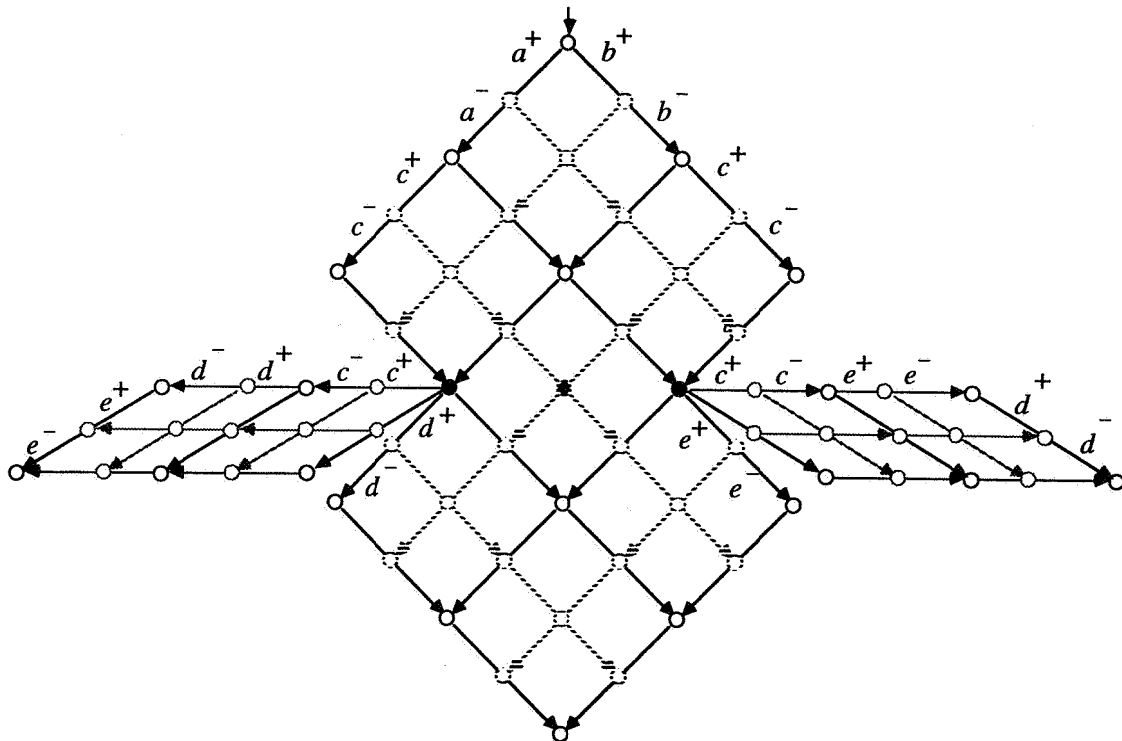FIGURE 9: *Owl example; process graph of* **E**

FIGURE 10: *Split owls*

too. Any move from the one process is now matched by a move from the other in mirror image. It follows that the split owls are bisimulation equivalent, and hence the original ones are split-2 bisimilar.

Unfortunately, this example does not generalize straightforwardly to higher dimensions. However, a variant in which both owls have 4 wings — two at each side, so that both processes have identical wings — does.

5.5 Description of the event structure $\mathbf{E}_n^\pi$ and its behavior.

The event structure $\mathbf{E}_n^\pi$ ($n \geq 2$, $\pi \in \{even,\ odd\}$) consists of n sequential components C_i ($i = 1,...,n$). For $i = 1,...,n$, component C_i is as depicted in Figure 11, where the causality relation is represented by arrows and unordered events are understood to be in pairwise conflict.
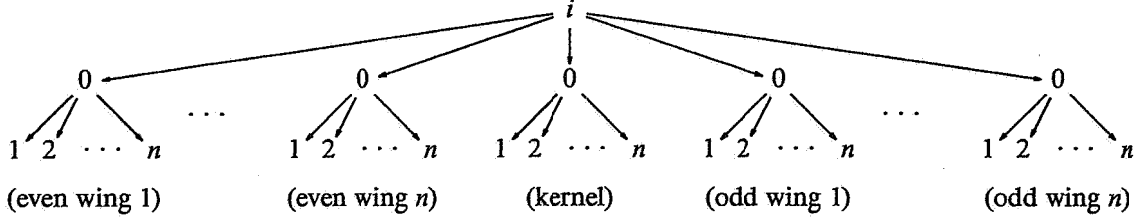

FIGURE 11: *Component C_i*

The component C_i consists of one initial event, $n - 1$ 'even wings', numbered from 1 to n but skipping i, one kernel, and $n - 1$ 'odd wings', also numbered from 1 to $i - 1$ and $i + 1$ to n. The kernel and each of the $2n - 2$ wings have $n + 1$ events. Component C_i first performs an action i, announcing its serial number; then the action 0 (for which it can choose from $2n - 1$ events) and finally one action s_i taken from the set $\{1,...,n\}$.

In the event structure $\mathbf{E}_n^\pi$ the components C_i are in principle independent. Only the events in the odd as well as the even wing j of component i are dependent on the 0-event in the kernel of component j (for $j \neq i$). Furthermore, all wings of different components are pairwise in conflict. Thus at most one component can execute a wing instead of its kernel and this can only happen if some other component completed the 0-event in its kernel first. Finally, equal actions s_i and s_j of different components i and j are in pairwise conflict, and there is an n-ary conflict between $s_1, \ldots, s_n$ unless

- $(s_1, \ldots, s_n)$ is an even permutation and one s_i is in an even wing,
- $(s_1, \ldots, s_n)$ is an odd permutation and one s_i is in an odd wing, or
- $(s_1, \ldots, s_n)$ is a π permutation and no s_i is in a wing.

Thus each component C_i should perform a different action s_i, and the last ($=3n^{th}$) action of $\mathbf{E}_n^\pi$ can only be performed in the three cases listed above.

DEFINITION. For $n \geq 2$ and $\pi \in \{even,odd\}$, $\mathbf{E}_n^\pi$ is the event structure $(E, <, \#, l)$ given by:

$E = \bigcup_{1 \leq i \leq n} C_i$ with $C_i = \{(i, 0)\} \cup \{(h,i) | 0 \leq h \leq n\} \cup \{(h,i,j,p) | 0 \leq h \leq n;\ 1 \leq j \leq n,\ j \neq i;\ p \in \{even,odd\}\}$

$l(h,i) = h$	$(0 \leq i, h \leq n,\ (i,h) \neq (0,0))$
$l(h,i,j,p) = h$	$(0 \leq h \leq n;\ 1 \leq i,j \leq n,\ j \neq i;\ p \in \{even,odd\})$
$(i, 0) < (0,i) < (h,i)$	$(h,i = 1,...,n)$
$(i, 0) < (0,i,j,p) < (h,i,j,p)$	$(h,i,j = 1,...,n,\ i \neq j;\ p = even,odd)$
$(0,j) < (0,i,j,p)$	$(i,j = 1,...,n,\ i \neq j;\ p = even,odd)$

$(h,i)\#(g,i,j,p)$		$(g,h = 0,...,n;\ i,j = 1,...,n,\ i \neq j;\ p = even,odd)$
$(h,i,j,p)\#(g,k,l,q)$	if $(i,j,p) \neq (k,l,q)$	$(g,h = 0,...,n;\ i,j,k,l = 1,...,n,\ i \neq j,\ k \neq l;\ p,q = even,odd)$
$(h,i,j,p)\#(g,i,j,p)$	if $g \neq h$	$(g,h,i,j = 1,...,n,\ i \neq j;\ p = even,odd)$

$(h,i) \# (g,i)$ if $g \neq h$ $(g,h,i = 1,...,n)$

$(h,i) \# (h,k)$ if $i \neq k$ $(h,i,k = 1,...,n)$

$(h,i) \# (h,k,j,p)$ if $i \neq k$ $(h,i,j,k = 1,...,n,\ k \neq j\ ; p = even,odd)$

$\#\{(1,i_1),...,(m-1,i_{m-1}),(m,i_m,j,p),(m+1,i_{m+1}),...,(n,i_n)\}$ if $(i_1...i_n)$ is a non-p permutation of $(12...n)$

$\#\{(1,i_1),(2,i_2),...,(n,i_n)\}$ if $(i_1...i_n)$ is a non-π permutation of $(12...n)$.

For fixed i, the events in C_i constitute a *component*; $(i,0)$ is the *initial event* of C_i, $\{(h,i)|0 \leqslant h \leqslant n\}$ is the *kernel* of C_i and, for fixed j,p, $\{(h,i,j,p)|0 \leqslant h \leqslant n\}$ is a *wing* of C_i, where j is the *wing number* and p the *parity* of the wing.

Note that $POM(\mathbf{E}_n^\pi) = Y_n^\pi$ for $n \geqslant 2$ and $\pi \in \{even,odd\}$ (cf. Section 3.6).

We find it convenient to distinguish three kinds of configurations in $split_m(\mathbf{E}_n^\pi)$. An *upper configuration* is one in which at least one event $(0,i)$ has not started (i.e. for certain $1 \leqslant i \leqslant n$ the configuration does not contain the event $(0,i)_1$), a *middle configuration* is one in which all n events $(0,i)$ are active, and an *under configuration* is one in which all events $(0,i)$ have started and at least one of them has terminated (i.e. for certain $1 \leqslant i \leqslant n$ the configuration contains the event $(0,i)_m$). Note that all configurations that contain any event from a wing count as upper configurations.

5.6. PROPOSITION. *For all $n \geqslant 2$, the event structures $\mathbf{E}_n^{even}$ and $\mathbf{E}_n^{odd}$ are not split-n $+1$ trace equivalent.*
PROOF. Since $(1,\ldots,n)$ is an even permutation, the following sequence is a trace of $split_{n+1}(\mathbf{E}_n^{even})$.

$$1\ 0_1\ ...\ 0_n\ 2\ 0_1\ ...\ 0_{n-1}\ 3\ 0_1\ ...\ 0_{n-2}\ \cdots\ n\ 0_1\ 0_{n+1}\ 1\ 0_n\ 0_{n+1}\ 2\ 0_{n-1}\ 0_n\ 0_{n+1}3\ \cdots\ 0_2\ ...\ 0_{n+1}\ n.$$

Just before the first 0_{n+1} action the computation passes through a middle configuration. This implies that all n kernels are executed. Moreover, the second occurrence of the action i $(i = 1,...,n)$ must stem from the ith component. It follows that the sequence is not a trace of $split_{n+1}(\mathbf{E}_n^{odd})$. $\square$

5.7. It remains to be shown that $\mathbf{E}_n^{even}$ and $\mathbf{E}_n^{odd}$ are split-n bisimulation equivalent. For this purpose we need the following notions.

DEFINITION. If $\mathbf{E}$ is an event structure with configuration X, then $\mathbf{F} = \mathbf{E}$ *after* X is the event structure given by
- $E_\mathbf{F} = \{e \in E_\mathbf{E} | e \notin X \wedge \{e\} \cup X \notin \#_\mathbf{E}\}$,
- $d <_\mathbf{F} e$ iff $d <_\mathbf{E} e$,
- $Y \in \#_\mathbf{F}$ iff $Y \cup X \in \#_\mathbf{E}$,
- $l_\mathbf{F}(e) = l_\mathbf{E}(e)$.

If X and Y are two configurations of $\mathbf{E}$ with $Y - X = \{e\}$ and $l_\mathbf{E}(e) = a$ we write $X \overset{a}{\to} Y$.

An *abbreviated bisimulation* between two event structures $\mathbf{E}$ and $\mathbf{F}$ is a binary relation R between the configurations of $\mathbf{E}$ and $\mathbf{F}$ such that
- $\varnothing R \varnothing$,
- if XRY and $X \overset{a}{\to} X'$ then $Y \overset{a}{\to} Y'$ for a Y' with either $X'RY'$ or $\mathbf{E}$ *after* $X' \cong \mathbf{F}$ *after* Y',
- if XRY and $Y \overset{a}{\to} Y'$ then $X \overset{a}{\to} X'$ for an X' with either $X'RY'$ or $\mathbf{E}$ *after* $X' \cong \mathbf{F}$ *after* Y'.

PROPOSITION. *If there exists an abbreviated bisimulation between two event structures, then they are interleaving bisimulation equivalent.*
PROOF. Trivial. $\square$

THEOREM. For all $n \geqslant 2$, the event structures $\mathbf{E}_n^{even}$ and $\mathbf{E}_n^{odd}$ are split-n bisimulation equivalent.

PROOF. Note that the upper configurations of $split_n(\mathbf{E}_n^{even})$ are the same as the the upper configurations of $split_n(\mathbf{E}_n^{odd})$. The same observation can be made for middle configurations, but not for under configurations. We claim that the identity relation between their upper configurations is an

abbreviated bisimulation between $split_n(E_n^{even})$ and $split_n(E_n^{odd})$.

- The empty configuration is an upper configuration of both event structures.

- Suppose $(X,Y) \in R$, i.e. $X = Y$ and X is an upper configuration, and $X \xrightarrow{a} X'$. There are three possibilities:

1. X' is an upper configuration of $split_n(E_n^{even})$, and hence of $split_n(E_n^{odd})$. In this case the requirement is trivially satisfied.

2. X' is a middle configuration of $split_n(E_n^{even})$, and hence of $split_n(E_n^{odd})$. Choose $Y' = X'$. Now it suffices to show that $split_n(E_n^{even})$ *after* $X' \cong split_n(E_n^{odd})$ *after* X'. Note that for each $i \in \{1,...,n\}$ there is an $u_i \in \{1,...,n-1\}$ such that $(0,i)_{u_i} \in X'$ but $(0,i)_{u_i+1} \notin X'$. By the pigeonhole principle, there must be $i, j \in \{1,...,n\}$, $i \neq j$, with $u_i = u_j$. Now the mapping $f : E_{split_n(E_n^{even})}$ *after* $X' \to E_{split_n(E_n^{odd})}$ *after* X', defined by

$$f((h,i)_u) = (h,j)_u$$

$$f((h,j)_u) = (h,i)_u$$

$$f((h,k)_u) = (h,k)_u \text{ if } k \neq i,j$$

is an isomorphism. The only non-trivial requirement is the preservation of n-ary conflict. Here the argument is that an even permutation $(i_1 \cdots i_n)$ changes in an odd one (and vice versa) when i and j are exchanged.

3. X' is an under configuration of $split_n(E_n^{even})$. Since X was an upper configuration, it must be the case that $X' - X = \{(0,i)_1\}$ for certain $i \in \{1,...,n\}$ and X' contains an event $(0,j)_n$ for certain $j \neq i$. Choose $Y' = Y \cup \{(0,i,j,even)_1\}$. Since all prerequisites of the event $(0,i,j,even)_1$ are already in Y, and since Y contains no events (h,i) and no wing events (these are the only events that can be in conflict with $(0,i,j,even)_1$), we have $Y \xrightarrow{a} Y'$. Furthermore $split_n(E_n^{even})$ *after* $X' \cong split_n(E_n^{odd})$ *after* Y' through the isomorphism f, defined by:

$$f((h,i)_u) = (h,i,j,even)_u$$

$$f((h,k)_u) = (h,k)_u \text{ if } k \neq i.$$

- The remaining requirement follows by symmetry. $\square$

5.8 The difference between sequence refinements and conflict refinements. Figures 12 and 13 provide a branching time version of the example of Definition 3.8, showing that split-ω trace equivalence differs from ST-trace equivalence.
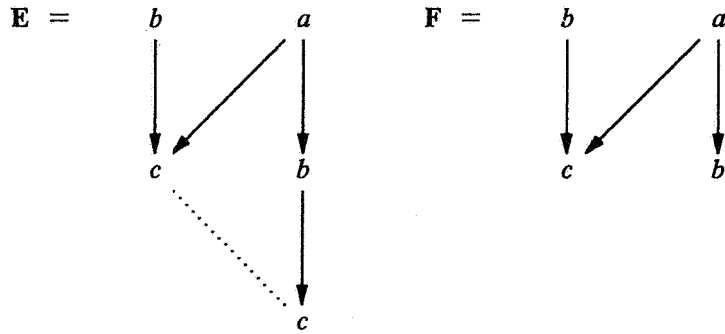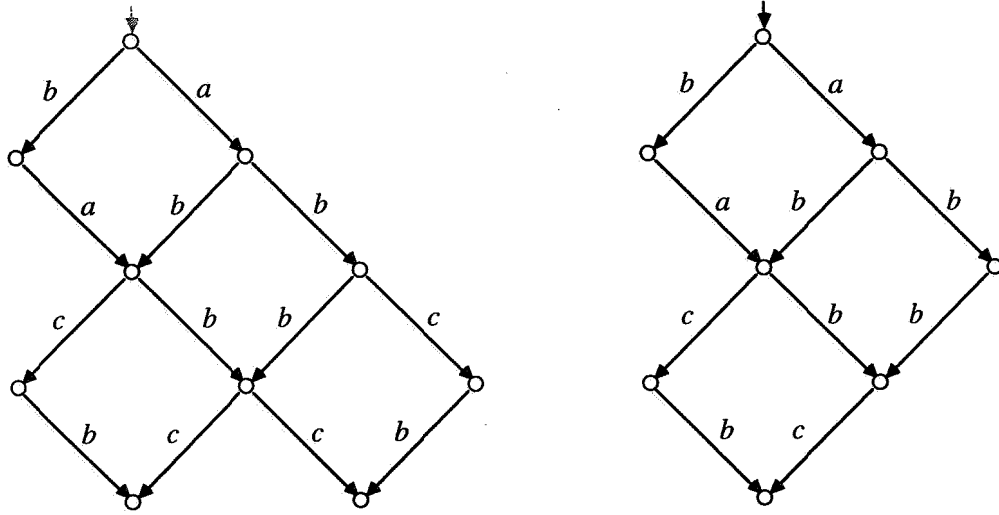

FIGURE 12: *The difference between splitting and conflict refinements*

We claim that for any $n \geq 2$, $E \approx_{it}^n F$. In order to see that this is true, first observe that one can obtain the process graph of $split_n(E)$ (resp. $split_n(F)$) by placing an $n \times n$ grid in each square of Figure 13. As

FIGURE 13: *Graph representations of* E *and* F

an example we have depicted in Figure 14 the process graphs of $split_2(E)$ and $split_2(F)$. This reveals that any trace of say $split_n(E)$ can be depicted as a zig-zag line through the graph of E.
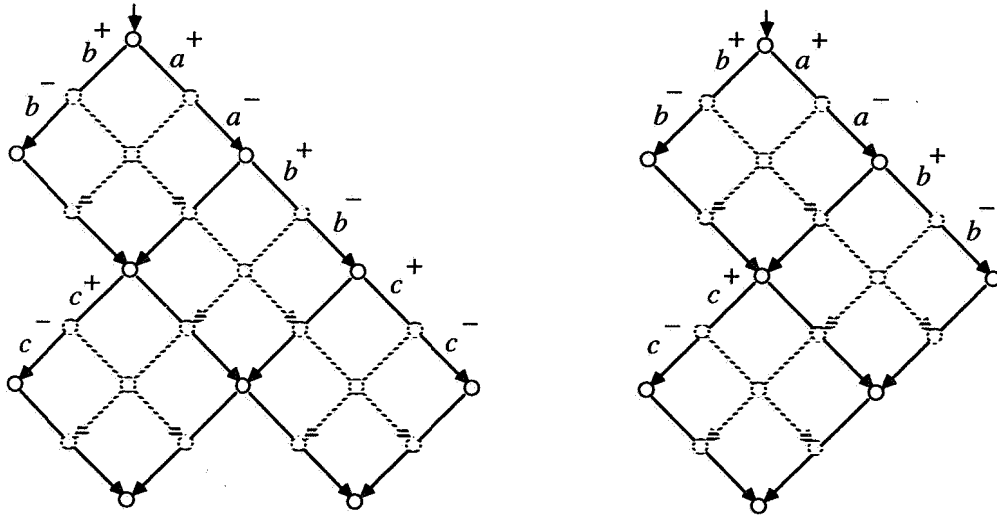
FIGURE 14: *Graph representations of* $split_2(E)$ *and* $split_2(F)$

Any zig-zag of F clearly corresponds to a zig-zag of E. The only zig-zag's of E for which it is non-trivial that there is a corresponding zig-zag in F (i.e. one determining the same action sequence), are the ones which enter the rightmost square of the graph of E. To deal with these zig-zag's we consider the line in the graph of E that corresponds to the moments at which the first a has finished and both b's are at the same stage of their execution (see Figure 15). We observe that for any zig-zag in the process graph of E, one can construct a corresponding zig-zag for F by mirroring anything on the right side of the symmetry line in E and placing the resulting zig-zag in the process graph of F. This argument does not carry over to ST-traces, as the end of one b may not be matched by the end of the other.
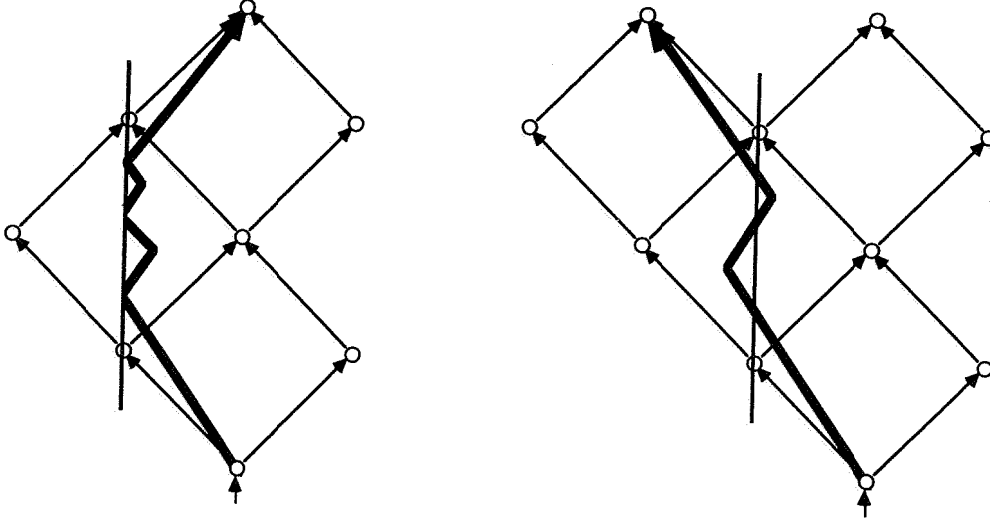
FIGURE 15: *Mirroring executions*

It is interesting to note that E and F can be distinguished if we consider refinements which introduce conflicts. We will not bother here to state a formal definition of these refinements,[1] because we think it is possible to make this point without it: If one refines b in E and F by the event structure $b_1 \cdot b_2 + b_3 \cdot b_4$, then the refined version of E has a trace $b_1 a b_3 b_4 c$, which the refined version of F does not have.

5.9. The observation above may be taken as an incentive to propose extensions of the splitting doctrine beyond the level of sequence refinement. Could it be that durational and structural aspects of actions can be captured in interleaving semantics after refining actions a into $a_1 \cdot a_2 + a_3 \cdot a_4$, or other finite processes? The example E_n^π of Section 5.5 tells that this is not possible. Let an *internal state* be any state in the graph representation of a process that is not initial or final (for this purpose take the final states to be the ones without outgoing transitions). The process $a_1 \cdot a_2 \cdot \cdots \cdot a_n$ that is substituted for a in split-n semantics for instance has $n - 1$ internal states, whereas $a_1 \cdot a_2 + a_3 \cdot a_4$ has two. Now a straightforward generalization of the proof of Theorem 5.7 shows that after splitting each action into a process with less than n internal states, the systems E_n^{even} and E_n^{odd} are interleaving bisimulation equivalent. Still, these systems have different real-time execution sequences and can be distinguished by splitting actions in $n + 1$ parts.

5.10. The example of Section 5.8 does not show that $\approx_{ib}^\omega$ differs from $\approx_{STb}$, as the two process graphs of Figure 13 are not even interleaving bisimulation equivalent. Namely only F can execute ab in such a way that afterwards no c is possible. In fact, GORRIERI & LANEVE [13] show that at least for guarded CCS processes these equivalences coincide, as conjectured in [12]. This is the only point in our classification where the picture for branching time equivalences looks different from the one for the linear time equivalences. It is therefore natural to ask, at which point in the *linear time - branching time spectrum* [9] the limit of the split equivalences collapses with ST-equivalence. The example above can easily be generalized to one that distinguishes *split-ω ready trace equivalence* from *ST-ready trace equivalence*, namely by comparing F with E+F, where the + is implemented on event structures by taking the disjoint union and putting all events on the left in conflict with all those on the right. On acyclic process graphs this amounts to identifying their roots (and keeping the rest disjoint). F and

---

1. Actually, such a definition is nontrivial on the domain of event structures we consider in this paper because one has to duplicate events. A much more natural definition has been given in [11] using the *flow event structures* of [4].

E+F are still ST-tra e different, as only the latter one has an ST-trace as mentioned in Proposition 3.9. However, every event sequence of $split_n(E+F)$ can be matched by one of $split_n(F)$, in such a way that in every pair of corresponding configurations the set of enabled actions is the same. This makes these systems ready trace equivalent [9]. E and F are not ready trace equivalent, as the latter can execute ab in such a way that afterwards no c is possible. Moreover, F and E+F are not *simulation equivalent* [9], and we know no example showing a difference between split-ω simulation equivalence and ST-simulation equivalence.

5.11. Note that omitting the causal link between a and c in both event structures of Figure 12 (yielding an extra square in Figure 13) gives us an equally satisfactory example. Yet another one (that does not generalize to ready trace semantics or even failure semantics) is given by the process expressions $(b\|abc) + (bc\|ab)$ and $(bc\|ab)$. This example stems from K.S. LARSEN [17]. It is especially interesting because it fits in the subset of CCS studied by ACETO & HENNESSY [1]. On this language split-2 bisimulation equivalence coincides with $\approx_{STb}$ and even with history preserving bisimulation equivalence (as observed in [8]). The example above shows that on this language split-2 trace equivalence does not coincide with ST-bisimulation however.

REFERENCES

[1] L. ACETO & M. HENNESSY (1993): *Towards action-refinement in process algebras.* Information and Computation 103(2), pp. 204-269.

[2] L. ACETO & M. HENNESSY (1994): *Adding action refinement to a finite process algebra.* Information and Computation 115(2), pp. 179-247.

[3] B. BLOOM (1988): *Constructing two-writer atomic registers.* IEEE Transactions on Computers 37(12), pp. 1506-1514.

[4] G. BOUDOL (1990): *Flow event structures and flow nets.* In: Semantics of Systems of Concurrent Processes (I. Guessarian, ed.), Proceedings of the LITP Spring School on Theoretical Computer Science, La Roche Posay, France 1990, LNCS 469, Springer-Verlag, pp. 62-95.

[5] L. CASTELLANO, G. DE MICHELIS & L. POMELLO (1987): *Concurrency vs Interleaving: an instructive example.* Bulletin of the EATCS 31, pp. 12-15.

[6] J.L. GISCHER (1984): *Partial orders and the axiomatic theory of shuffle.* Ph.D. Thesis, Report No. STAN-CS-84-1033, Stanford University.

[7] R.J. VAN GLABBEEK (1988): *An operational non-interleaved process graph semantics of CCSP* (abstract). In: Combining compositionality and concurrency, summary of a GMD-workshop, Königswinter, March 1988 (E.-R. Olderog, U. Goltz & R.J. van Glabbeek, eds.), Arbeitspapiere der GMD 320, Sankt Augustin, pp. 18-19.

[8] R.J. VAN GLABBEEK (1990): *The refinement theorem for ST-bisimulation semantics.* In: Programming Concepts and Methods (M. Broy & C.B. Jones, eds.), Proceedings IFIP Working Group 2.2/2.3 Working Conference, Sea of Galilee, Israel 1990, Elsevier Science Publishers B.V. (North Holland), pp. 27-52.

[9] R.J. VAN GLABBEEK (1990): *The linear time - branching time spectrum.* In: Proc. CONCUR 90, Amsterdam (J.C.M. Baeten & J.W. Klop, eds.), LNCS 458, Springer-Verlag, pp. 278-297.

[10] R.J. VAN GLABBEEK & U. GOLTZ (1989): *Equivalence notions for concurrent systems and refinement of actions.* Arbeitspapiere der GMD 366, Sankt Augustin, extended abstract appeared in: Proceedings 14th Symposium on Mathematical Foundations of Computer Science (MFCS), Porąbka-Kozubnik, Poland, August/September 1989 (A. Kreczmar & G. Mirkowska, eds.), LNCS 379, Springer-Verlag, pp. 237-248.

[11] R.J. VAN GLABBEEK & U. GOLTZ (1990): *Refinement of actions in causality based models.* In: Proceedings of the REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness (J.W. de Bakker, W.-P. de Roever & G. Rozenberg, eds.), Mook, The Netherlands 1989, LNCS 430, Springer-Verlag, pp. 267-300.

[12] R.J. VAN GLABBEEK & F.W. VAANDRAGER (1987): *Petri net models for algebraic theories of concurrency*. In: Proceedings PARLE conference, Eindhoven, Vol. II (Parallel Languages) (J.W. de Bakker, A.J. Nijman & P.C. Treleaven, eds.), LNCS 259, Springer-Verlag, pp. 224-242.

[13] R. GORRIERI & C. LANEVE (1995): *Split and ST Bisimulation Semantics*. Information and Computation, to appear.

[14] M. HENNESSY (1988): *Axiomatising finite concurrent processes*. SIAM Journal on Computing 17(5), pp. 997-1017.

[15] C.A.R. HOARE (1985): *Communicating sequential processes*, Prentice-Hall International.

[16] L. LAMPORT (1986): *On interprocess communication*. Distributed Computing 1, pp. 77-101.

[17] K.S. LARSEN (1988): *A fully abstract model for a process algebra with refinements*. Master Thesis, Aarhus University, Denmark.

[18] M. NIELSEN, U. ENGBERG & K.S. LARSEN (1989): *Fully abstract models for a process language with refinement*. In: Proceedings REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Noordwijkerhout (J.W. de Bakker, W.-P. de Roever & G. Rozenberg, eds.), LNCS 354, Springer-Verlag, pp. 523-548.

[19] M. NIELSEN, G.D. PLOTKIN & G. WINSKEL (1981): *Petri nets, event structures and domains, part I*. Theoretical Computer Science 13(1), pp. 85-108.

[20] E.-R. OLDEROG (1991): *Nets, terms and formulas*, Cambridge University Press.

[21] V.R. PRATT (1986): *Modelling concurrency with partial orders*. International Journal of Parallel Programming 15(1), pp. 33-71.

[22] V.R. PRATT (13 Nov 1990): *DO the great debate CONTINUE*, message to concurrency @theory.lcs.mit.edu, available at http://theory.stanford.edu/people/rvg/continue.html#Pratt.

[23] F.W. VAANDRAGER (1991): *Determinism → (event structure isomorphism = step sequence equivalence)*. Theoretical Computer Science 79, pp. 275-294.

[24] W. VOGLER (1993): *The limit of $split_n$-language equivalence*. Report Nr 288, Institut für Mathematik, Universität Augsburg, Germany.

[25] G. WINSKEL (1987): *Event structures*. In: Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II; Proceedings of an Advanced Course, Bad Honnef, September 1986 (W. Brauer, W. Reisig & G. Rozenberg, eds.), LNCS 255, Springer-Verlag, pp. 325-392.