Refinement of Actions and Equivalence Notions for Concurrent Systems

Rob van Glabbeek
Computer Science Department, Stanford University
Stanford, CA 94305-9045, USA.
rvg@cs.stanford.edu

Ursula Goltz
Institut für Informatik, Universität Hildesheim
Postfach 10 13 63, D-31113 Hildesheim, Germany.
goltz@informatik.uni-hildesheim.de

We study an operator for refinement of actions to be used in the design of concurrent systems. Actions on a given level of abstraction are replaced by more complicated processes on a lower level. This is done in such a way that the behaviour of the refined system may be inferred compositionally from the behaviour of the original system and from the behaviour of the processes substituted for actions. We recall that interleaving models of concurrent systems are not suited for defining such an operator in its general form. Instead, we define this operator on several causality based, event oriented models, taking into account the distinction between deadlock and successful termination. Then we investigate the interplay of action refinement with abstraction in terms of equivalence notions for concurrent systems, considering both linear time and branching time approaches. We show that besides the interleaving equivalences, also the equivalences based on steps are not preserved under refinement of actions. We prove that linear time partial order semantics are invariant under refinement. Finally we consider various bisimulation equivalences based on partial orders and show that the finest two of them are preserved under refinement whereas the others are not. Termination sensitive versions of these equivalences are even congruences for action refinement.

Keywords Concurrency, action refinement, event structures, semantic equivalences, interleaving vs. partial orders, bisimulation, termination, deadlock.

Notes This paper combines and extends the material of [GG-a/c/d/e], except for the part in [GG-c] on refinement of transitions in Petri nets and the discussion of TCSP-like parallel composition in [GG-e]. An informal presentation of some basic ingredients of this paper appeared as [GG-b]. Among others, the treatment of action refinement in stable and non-stable event structures is new.
The research reported here was supported by Esprit project 432 (METEOR), Esprit Basic Research Action 3148 (DEMON), Sonderforschungsbereich 342 of the TU München, ONR grant N00014-92-J-1974 and the Human Capital and Mobility Cooperation Network EXPRESS (Expressiveness of Languages for Concurrency).

Contents

1 Introduction

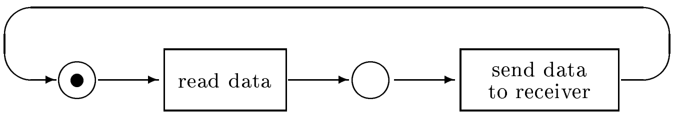
Our aim is to investigate methods to design concurrent systems in a modular way, in order to achieve higher reliability with respect to functional correctness. In the context of this research, we introduce an operator for refinement of actions. This section serves as a motivation for our choices. In Part I of this paper the refinement operator will be defined on various causality based, event oriented models. In Part II we investigate the interplay between refinement and abstraction in terms of equivalence notions for concurrent systems. Related work is discussed in the concluding section.

1.1 Refinement of actions

We consider the design of concurrent systems in the framework of approaches where the basic building blocks are the actions which may occur in a system. By an action we understand here any activity which is considered as a conceptual entity on a chosen level of abstraction. This allows the representation of systems in a hierarchical way, changing the level of abstraction by interpreting actions on a higher level by more complicated processes on a lower level. We refer to such a change in the level of abstraction as *refinement of actions*. This approach is in the line of *stepwise refinement*, a methodology in program development first named and described in [Wirth]: a "program is gradually developed in a sequence of *refinement steps*. In each step, one or several instructions of the given program are decomposed into more detailed instructions." More recently the keyword "refinement" has been used to indicate any transformation of a program or system that can be justified because the transformed program is related to the original through an implementation relation. Suitable implementation relations are preorders such as *trace inclusion*, various kinds of *simulation* or any of the semantic equivalences mentioned in this paper. Many papers in this tradition can be found in [BRR]. In order to distinguish our notion of refinement, which is more in the spirit of [Wirth], from these latter approaches, we adopt the terminology *action* refinement.

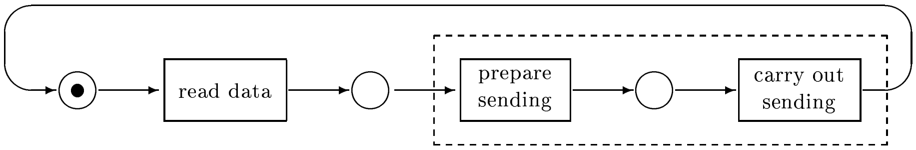
Example 1.1

Consider the design of a sender, repeatedly reading data and sending them to a certain receiver. A first description of this system is given by the Petri net shown below.[1]



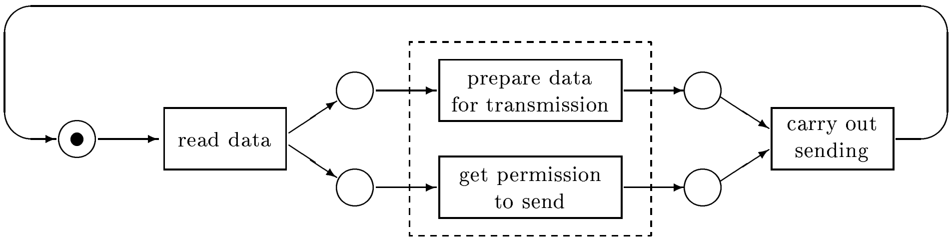
[1] An introduction to Petri nets and the way they model concurrent systems can be found in [Reisig]; the refinement mechanism used in this example has been treated formally in [GG-c].

On a slightly less abstract description level the action "send data to receiver" might turn out to consist of two parts "prepare sending" and "carry out sending", to be executed sequentially. This corresponds to the following refined Petri net.



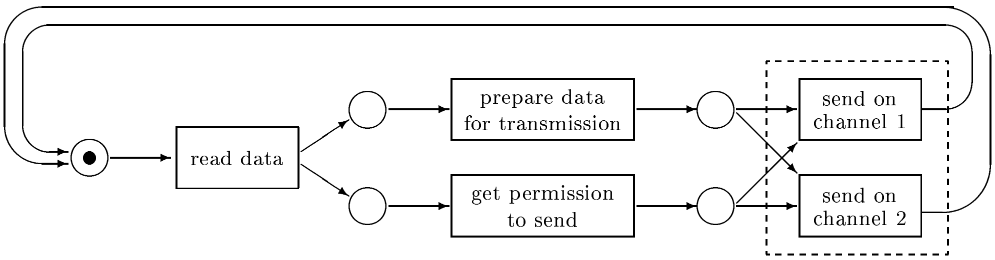
Refinement by a sequential process

Then the action "prepare sending" may be decomposed in two independent activities "prepare data for transmission" and "get permission to send", to be executed on different processors.



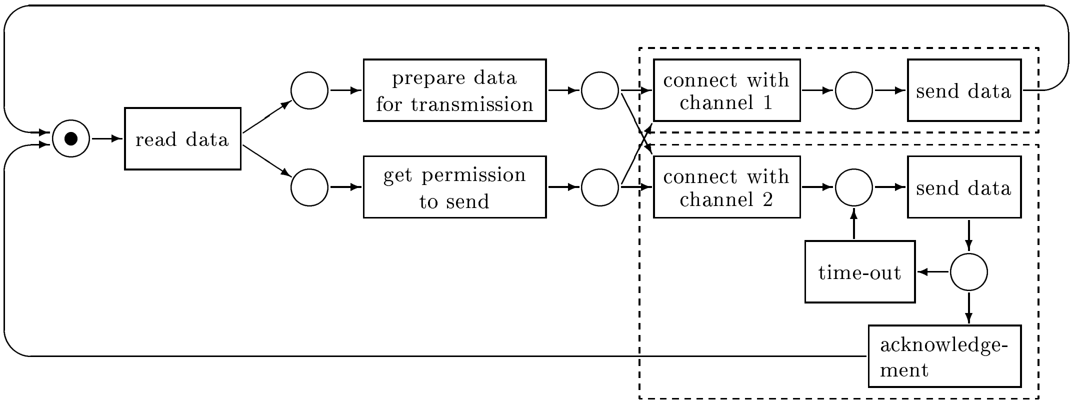
Refinement by a parallel process

Furthermore it may turn out that there are two alternative channels for sending messages. Each time the sender should choose one of them to send a message, perhaps depending on which one is available at the moment.



Refinement by alternative actions

On an even more concrete level of abstraction, channel 2 may happen to be rather unreliable, and getting a message at the other end requires the use of a communication protocol. On the other hand, channel 1 may be found to be reliable, and does not need such a precaution.



Refinement by an infinite process

Here we see that it may happen that the process we have substituted for the action "send on channel 2" does not terminate. It may happen that the attempt of sending data always fails and this prevents the system of reaching its initial state again.

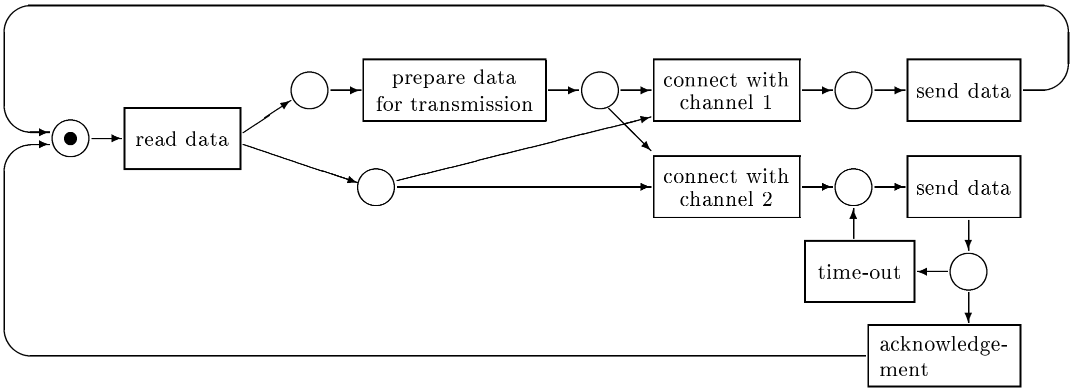
In this paper, action refinement will be modelled as an operator, taking as arguments a system description on a given level of abstraction and an interpretation of (some of) the actions on this level by more complicated processes on a lower level, and yielding a system description on the lower level. This will be done in such a way that the behaviour of the refined system may be inferred compositionally from the behaviour of the original system and from the behaviour of the processes substituted for actions. It should be noted that in our framework (as is common in many action oriented approaches) actions are semantically treated as uninterpreted activities. Thus the "behaviour of the original system" does not contain any information about the nature of actions that are to be refined. Such information is added during action refinement. As a consequence there is no such concept as the "correctness" of a refinement. This may indicate a difference with other approaches (cf. Conclusion).

As illustrated above, we want to allow to substitute rather general kinds of behaviours for actions. We even allow the refinement of an action by an infinite behaviour. This contradicts a common assumption that an action takes only a finite amount of time. It means that when regarding a sequential composition $a;b$ we can not be sure that b occurs under all circumstances; it can only occur if the action a really terminates successfully.

The only type of refinement that we will not allow is *forgetful refinement*, "forgetting" actions by replacing them with the empty process.

Example 1.2

Continuing Example 1.1 we could imagine that getting permission to send turns out to be unnecessary and can be skipped. Hence we replace the corresponding action by the empty behaviour, thus obtaining



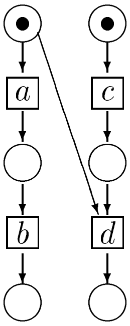
Forgetful refinement

Even though this operation seems natural when applied as in the above example, it may cause drastic changes in the possible behaviours of a system. It may happen that executing a certain action a prevents another action from happening. This property should be preserved under refinement of a. However, if a is completely removed, it cannot prevent anything any more, which can remove a deadlock possibility from the system.

Example 1.3

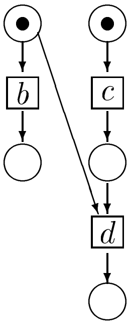
Consider the Petri net

$N =$



and the net obtained when refining a by the empty behaviour:

$N' =$



In the first net it is possible to execute a and c, and by this reach a state where d is not possible. If we try to deduce the behaviour after refinement from the behaviour of N, we would expect that the refined system may reach a state, by executing just c, where d is not possible. However, this is not the case for N'. After c, but before b, it is always possible to execute d in N'.

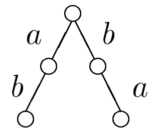
Thus "forgetful" refinements can not be explained by a change in the level of abstraction at which systems are regarded. For this reason they will not be considered here. Other methods to avoid this problem will be reviewed in the conclusion.

1.2 Modelling concurrent systems

In order to define a suitable refinement operator, one first has to select a model for the description of concurrent systems. The models of concurrency found in the literature can roughly be distinguished in two kinds: those in which the independent execution of two processes is modelled by specifying the possible interleavings of their (atomic) actions, and those in which the causal relations between the actions of a system are represented explicitly. The interleaving based models were devised to describe systems built from actions that are assumed to be instantaneous or indivisible. Nevertheless, one might be tempted to use them also for the description of systems built from actions that may have a duration or structure. However, the following example shows that it is not possible to define the desired compositional refinement operator on such models of concurrency without imposing some restrictions (as already observed in [Pratt-a] and [CDP]).

Example 1.4

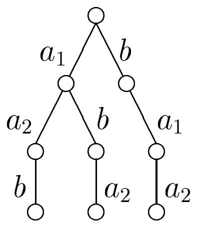
The systems $P = a|b$, executing the actions a and b independently, and $Q = a;b+b;a$, executing either the sequence ab or the sequence ba, cannot be distinguished in interleaving models; they are represented, for example, by the same tree in the model of synchronisation trees [Milner-a].

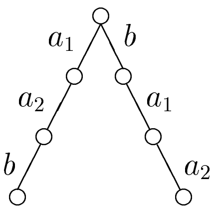
tree $(P) =$ *tree* $(Q) =$ 

After refining a into the sequential composition of a_1 and a_2, thereby obtaining the systems

$$P' = (a_1;a_2) \mid b \quad \text{and} \quad Q' = (a_1;a_2);b + b;(a_1;a_2)\,,$$

their tree representations are different:

tree $(P') =$ 

, *tree* $(Q') =$ 

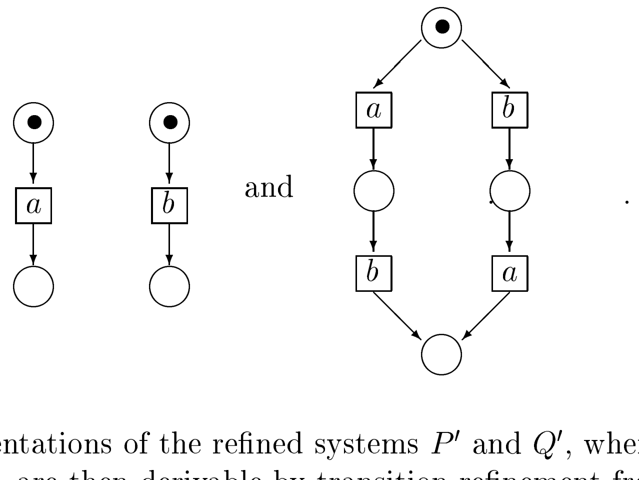
.

The two systems are even non-equivalent, according to any reasonable semantic equivalence, since only P' can perform the sequence of actions $a_1 b a_2$. Hence, in the model of synchronisation trees the semantic representation of the refined systems is not derivable from the semantic representation of the original systems. The same holds for other interleaving models.

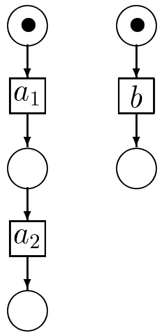
There are still ways left to define a compositional refinement operator on interleaving based models. First of all one could restrict the kind of refinements that are allowed in such a way that situations as in Example 1.4 cannot occur. Of course this would exclude the possibility of refining a in $a_1;a_2$ in either P or Q (or both). This idea is investigated in [CGG]. Secondly, one could refrain from dealing with both P and Q by restricting the class of systems under consideration. Thirdly, one could consider a different notion of action refinement such that either $\mathit{ref}(P) \neq P'$ or $\mathit{ref}(Q) \neq Q'$. Finally, one could find reasons to consider P' and Q' equivalent after all. Approaches based on these possibilities will be discussed in the conclusion. In this paper we chose to allow general systems, including P and Q, and rather general refinements, including at least the one of Example 1.4. Furthermore, our understanding of how action refinement should work out is exactly captured by Example 1.4, and we have in general no reasons to identify P' and Q'. Hence interleaving based models are unsuited for our approach. On the other hand we will show that the desired compositional refinement operator *can* be defined on causality based models of concurrency without imposing such restrictions.

Example 1.5

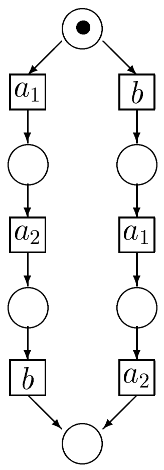
The systems $P = a|b$ and $Q = a;b + b;a$ from Example 1.4 may be represented by the Petri nets



The Petri net representations of the refined systems P' and Q', where a is replaced by the sequence $a_1 a_2$, are then derivable by transition refinement from the nets for the original systems. We obtain



and



.

Since action refinement entails general sequential composition (namely $P;Q$ is the result of refining a into P and b into Q in $a;b$), we do not escape from the problems usually associated with the latter. There are basically two approaches to defining a sequential composition-like operator (and as a consequence also two approaches in defining a refinement operator). One idea is that the second component starts when the first component cannot do any further actions. This approach is for instance taken in [Moller]. The other idea is that systems end either in a state of deadlock (this arises for instance if all components in a parallel composition are waiting for an opportunity to synchronise with another component) or by terminating successfully. Now the second component only starts when the first component terminates successfully. This approach is taken for instance in the system description language ACP [BK, BW]. We refer to an operator based on the first idea as a *sequencing* operator, and to one based on the second idea as a *sequential composition* operator. The difference between the two operators is illustrated by the following example.

Example 1.6

We consider a variant of the classical example of the dining philosophers who are seated around a round table with one fork between every two of them. In our case, the philosophers do not think, and eat only once. Each philosopher takes first one of the two forks next to his plate, then the other one, and subsequently eats his spaghetti with the two forks. If any of the forks is missing (because it was taken first by his neighbour) he just waits until it becomes available again. After the meal, both forks are returned where they were found. The system P can be described as the parallel composition of the philosophers and the forks with an appropriate synchronization mechanism. Let Q be the system consisting of a waiter, who cleans up the table afterwards. In the sequential composition of P and Q, the waiter starts his work only when all philosophers finished eating. The situation were all philosophers have taken their right-hand fork leads to a state of deadlock, in which the waiter remains passive. When P and Q are sequenced however, the waiter must be equipped with a mechanism for deadlock detection, and starts his work when he is sure that no philosopher will eat any more.

Here we have chosen the sequential composition approach, and define the refinement operator accordingly. Thus our semantic models must be able to distinguish between deadlock and successful termination.

In this paper we have chosen *event structures* as our model of concurrency. They feature explicit representations of causal links between action occurrences, thereby also representing independence of activities. Additionally, they represent the choice structure of systems; they show where decisions between alternative courses of action are taken. We will consider several types of event structures, as will be discussed in the beginning of Part I. Traditionally, in event structures deadlock and successful termination are not distinguished. We extend the event structure models accordingly where necessary. Additionally, we consider *configuration structures*, which can be regarded as more abstract representations of event structures.

Examples 1.1 and 1.5 may have suggested Petri nets as an appropriate model of concurrency for the definition of a refinement operator. They have appealing graphical representations and, unlike for event structures, many infinite behaviours may be represented as finite net structures together with the "token game". However, the formal definition of a general refinement operator [GG-c, JM, BDE] is rather complicated and gives rise to even more complicated proofs. In part this is due to the bookkeeping of places, which play a secondary rôle in an action-oriented approach. Therefore in the present paper we have chosen event structures instead. Although Petri nets are less suitable than event structures for the formal definition and analysis of the refinement operator, they may be preferred for representing examples of systems before and after refinement (cf. the final net of Example 1.1).

Example 1.4 may suggest *term models* for the definition of a refinement operator. Concurrent systems are then represented by process expressions built from action constants by means of operators for alternative, sequential and parallel composition, etc., and refinement takes place by replacement of actions by compound terms. Such notions of *syntactic action refinement* are pursued in [Aceto-a, AH-a/b, NEL, GGR]. In this context, we consider it important to work with a parallel composition operator with synchronisation, which involves the study of the interaction between action refinement and the chosen synchronisation mechanism [Aceto-a, AH-b, GGR]. Since we consider this to be outside the scope of the present paper, we prefer to deal with *semantic* action refinement. Another reason to do so is that some of the equivalence notions that we will study are defined on term models only through an interpretation of terms in a domain of e.g. Petri nets or event structures. So for our study of equivalence notions it is then a more direct approach to define action refinement in such a semantic domain as well.

Approaches to action refinement in Petri nets, term models and other models of concurrency will be discussed in the conclusion.

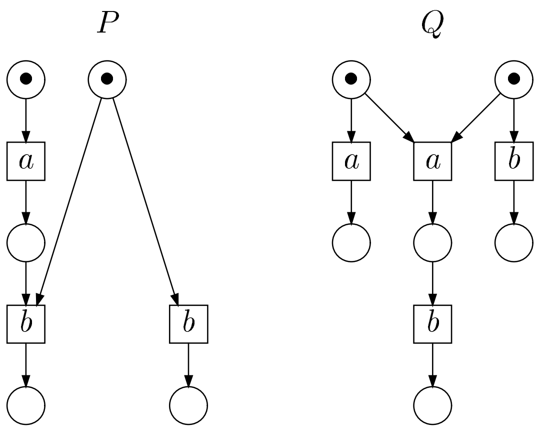
Even though we do not employ process algebraic terms formally in this paper, we will use them in examples, as in Example 1.4. We use ; for sequential composition, + for choice, $|$ for independent parallel composition and $\|_A$ for parallel composition with synchronisation on a set of actions A (hence $| = \|_\emptyset$). The operator + binds weaker than ;.

1.3 Equivalence notions

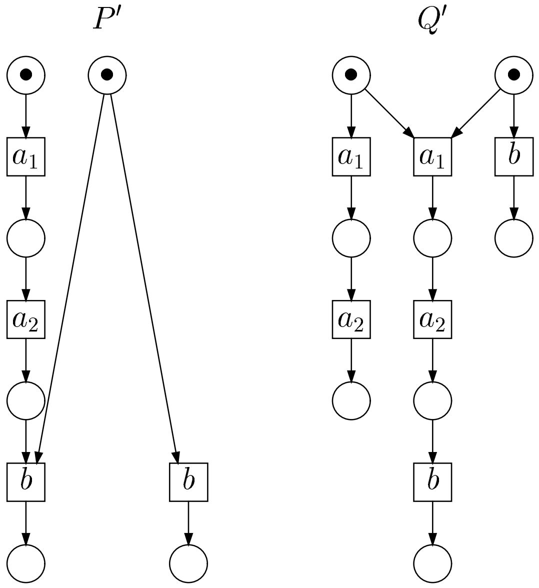
It can be argued that a concurrent system should not be represented just by an event structure (or a Petri net or a process expression), but rather by an equivalence class of such objects, since these models give a representation which is not abstract enough. For this purpose many equivalence notions have been proposed in the literature.

Often equivalence relations are used to establish the correctness of implementations with respect to specifications of concurrent systems. If P represents a specification and Q an implementation of a concurrent system, then the equivalence $P \approx Q$ states that the implementation is correct. If the representations contain occurrences of uninterpreted action symbols $a, b, \ldots$, this statement is independent of the interpretation of these symbols. Taking into account that action refinement can be understood to correspond to a change in the level of abstraction at which systems are described, without changing these systems in any way, it is reasonable to expect that if P and Q are equivalent according to a particular equivalence notion, this must still be the case after refining both system representations using the same interpretation of action symbols, unless it has been explicitly assumed that action symbols refer to atomic or instantaneous activities only.

Example 1.7 [vG-b]

> Consider the following specification of a concurrent system: "The actions a and b should in principle be performed independently on different processors, but if one of the processors happens to be ready with a before the other starts with b, b may also be executed on this processor instead of the other one." This system description is represented by the Petri net P below.
>
> 
>
>
> Suppose that someone comes up with an implementation in which first it is determined whether the actions a and b will happen sequentially or independently, and subsequently one of these alternatives will take place, as represented by the Petri net Q. Although this implementation does not seem very convincing, it will be considered "correct" by many equivalences occurring in the literature.

Let the next step in the design process consist of refining the action a in the sequential composition of two actions a_1 and a_2. From Q one thereby obtains the net Q' on the right.



If Q' is going to be placed in an environment where a_2 becomes causally dependent on b — it may be the case that b is an output action, a_2 is an input action, and the environment needs data from b in order to compute the data that are requested by a_2 — then deadlock can occur. However, if the refinement step splitting a in a_1 and a_2 is carried out on P already, the resulting system P' is deadlock free in the environment sketched above.

Thus the possibility of refining a somehow invalidates the correctness of the design step from P to Q.

In other words, a semantic equivalence that is not *preserved under action refinement* can only be useful in verifications if the actions involved in its definition are assumed to be atomic and unrefinable. Such equivalences are said to be strictly based on the assumption of *action atomicity* [CDP].

At this point a reader may argue that adding more information about the precise implementation of systems makes it more likely that systems turn out to be different that were indistinguishable on a higher level of abstraction. However, in our understanding a statement $P \approx Q$ says that based on the information present in the representations P and Q, the equivalence of the represented systems can be concluded. Thus adding more information to two system representations could make them equivalent if they were not so before, but cannot invalidate equivalence. Crucial for this argument is that, in the original system representations P and Q, it is already decided that different occurrences of the same action will be implemented in the same way, so that any difference between P and Q that could arise after refinement is already visible in the abstract representations, namely through the use of different action names for corresponding events.

Many different equivalence notions have been proposed in the literature, driven by the consideration which aspects of system behaviour are crucial in the chosen context, and which one wants to abstract from. Roughly speaking, there are two important aspects of equivalences used in their classification: the preserved level of detail in runs of systems and the preserved level of detail of the choice structure between these runs. Concerning the first aspect, two opposite approaches are *interleaving semantics* and *causal semantics*. In *interleaving semantics* runs are represented by sequences of actions, thereby abstracting from the causal links between these actions. In *causal semantics* all causal dependencies between action occurrences in runs of processes are preserved. By *partial order semantics* we denote causal semantics in which causal dependencies can be represented as partial orders. Concerning the second aspect, the choice structure of systems, the simplest notion in the spectrum of equivalences yields *trace semantics* ("linear time"), in which a system is fully determined by the set of its possible runs, thereby completely neglecting the choice structure of concurrent systems. On the other end, *bisimulation semantics* ("branching time") additionally preserve the information where two different runs diverge (although choices between similar runs are still neglected). In between there are several *decorated trace semantics*, where part of the choice structure is taken into account. Mostly these are motivated by the observable behaviour of concurrent systems, according to some testing scenario. By combining both parameters in this classification, a two-dimensional spectrum of equivalence notions emerges. A third aspect of equivalences used in their classification is the treatment of internal or invisible actions (*strong* versus *weak* equivalences). In this paper, we only consider strong equivalences, that do not distinguish internal from external actions. Already early in the development of this spectrum of semantics for concurrent systems it has been observed that the issue of atomicity or refinability of actions is crucial in this context.

The limitations of models of concurrency in which actions are taken to be atomic and unrefinable are addressed in [Lamport] and [Pratt-a]: "These models are not appropriate for studying such fundamental questions as what it means to implement an atomic operation, in which the nonatomicity of operations must be directly addressed" [Lamport] and "we would like a theory of processes to be just as usable for events having a duration or structure, where a single event can be atomic from one point of view and compound from another" [Pratt-a]. [Pratt-a] uses this criterion as a reason for preferring partial orders over interleaving semantics: "A serious difficulty with the interleaving model is that exactly what is interleaved depends on which events of a process one takes to be atomic" and "In the partial-order model what it means for two events to be concurrent does not depend on the granularity of atomicity". In the same spirit, [CDP] shows by means of a simple example (essentially Example 1.4) that interleaving equivalences are not invariant under refinement of actions, and claims that "on the other hand, the approaches based on partial order are not constrained to the assumption of atomicity". Of course none of this came as a surprise for the "interleavers"; the assumption of action atomicity is explicitly made in almost all papers about interleaving semantics.

In Part II of this paper, we consider various equivalence notions from the spectrum explained above. We study equivalences based on interleaving of single actions or *steps* of concurrently executable actions, and on representing causality in runs by partial orders. For each of these notions of system runs, we study linear time equivalences as well as equivalences taking additionally the choice structure into account. It turns out that all interleaving and step equivalences are not preserved under refinement, regardless to which extent the choice structure is taken into account. We show that linear time partial order semantics, namely *pomset trace equivalence*, is indeed preserved under refinement of actions.

For branching time partial order semantics, considering bisimulation equivalences based on partial orders, the situation is more intricate (see the introduction to Part II). We will show that the two originally suggested versions of such equivalences are not preserved under action refinement (or only for restricted situations). Subsequently we show that a finer equivalence notion, proposed in [RT] under the name BS-bisimulation, is indeed preserved under refinement. We call this notion *history preserving bisimulation equivalence.* A strengthening of this notion due to [Bednarczyk] is preserved under refinement as well.

Since we treat action refinement as a binary operator, the property of an equivalence of being preserved under action refinement does not imply that it is a *congruence* for the refinement operator. The latter is the case if it is not only preserved when refining actions in the two systems by identical structures, but also when refining them by equivalent structures. Action refinement is only well-defined on a quotient domain induced by a semantic equivalence if this equivalence is a congruence for refinement. A semantic equivalence can only be congruence for sequential composition, and hence for action refinement, if it distinguishes between deadlock and termination. One can define a termination sensitive and a termination insensitive variant of each of the equivalence notions mentioned before. We will proof that the termination sensitive variants of pomset trace equivalence and history preserving bisimulation equivalence are congruences for action refinement. A counterexample will show that termination detection is indeed necessary for this.

Part I

Models and refinement

We consider systems that are capable of performing actions from a given set *Act* of action names. We will not distinguish external and internal actions here; we do not consider abstraction by hiding of actions.

The basic entities of our system models are called *events*; they model action occurrences and are labelled with the corresponding action names. Concurrent systems will be modelled as sets of labelled events together with some additional structure, specifying the occurrences of events in system runs.

We will consider four such event oriented models: *prime event structures with binary conflict* [NPW], *flow event structures* [BC-b], *(stable) event structures* [Winskel] and *configuration structures.*

Prime event structures with binary conflict are the original form of event structures and have simple and clear notions of causality and conflict between events. This allows for a particularly natural and simple definition of action refinement when refining actions by finite and conflict-free processes. However, this definition does not generalise to refinement by alternative actions or by infinite processes (cf. Example 1.1). In addition, prime event structures do not allow straightforward definitions of sequential and parallel composition (with synchronisation), and for that reason more general models have been considered.

Flow event structures were proposed in [BC-b] as a form of event structures which are particularly suited for giving semantics to languages like CCS. Here we show that they allow for a straightforward generalisation of the definition of action refinement on prime event structures to refinement by arbitrary non-empty processes. We establish that the behaviour of a refined event structure may be deduced compositionally from the behaviours of the original event structure and of the refinements of actions. However, the dynamic behaviour of flow event structures is defined in a rather indirect way, and it is often cumbersome to prove properties about it. Moreover, as shown in [CZ], the canonical definition of parallel composition on flow event structures is meaningful only on a subclass obeying a complicated structural property.

Stable and *non-stable event structures* were introduced in [Winskel]. They also allow for convenient definitions of CCS-like operators, including parallel composition, and do not suffer from the disadvantages of flow event structures mentioned above. Again we define a compositional general action refinement operator. However, the definition is somewhat more complicated than for prime and flow event structures, due to the less direct notion of causal dependence.

In the three models discussed so far, usually no distinction between *deadlock* and *successful termination* is made. They have been used as semantic models for CCS-like languages where action prefixing is the only form of sequential composition; in this context, the distinction is not necessary. As explained in the introduction, dealing with action refinement incorporates sequential composition, and the distinction becomes relevant. In [BV] this distinction is achieved, in the context of prime event structures, by introducing a special event-label $\surd$. The occurrence of a $\surd$-event models successful termination. A system run deadlocks iff it does not contain a $\surd$-event and cannot be prolonged. Another way of establishing this distinction is by upgrading the notion of an event structure with an explicit predicate indicating which runs of the system do successfully terminate. This is the approach we have chosen in our treatment of stable and non-stable event structures. For flow event structures on the other hand, we show that such a predicate can be derived from their structural properties. A notion of sequential composition on flow event structures will be introduced showing that this predicate has the intended semantics. In our treatment of prime event structures, which is included mainly to introduce our concept of action refinement, we avoid the entire issue by implicitly restricting attention to deadlock-free systems.[2]

Finally, we introduce a more abstract model of concurrency, called *configuration structures*, in which a concurrent system is described by exactly the relevant features that are used to derive the behaviour of event structures of each of the forms above. We define a refinement operator for them, and show that the more "syntactic" constructions in the previous sections are consistent with this general notion.

[2]When modelling operators like restriction or parallel composition, which can introduce deadlocks, this limitation is no longer acceptable and leads to contra-intuitive results when considering sequential composition or refinement as defined here (see Section 2.3).

2 Refinement of actions in prime event structures

In this section, we show how to refine actions in the most simple form of event structures, prime event structures with a binary conflict relation [NPW]. Furthermore, we motivate our move to more general structures in the next sections by discussing the limitations of this approach.

2.1 Prime event structures

Definition 2.1

A *(labelled) prime event structure (over an alphabet Act)* is a 4-tuple $\mathcal{E} = (E, <, \#, l)$ where

- E is a set of *events*,
- $< \subseteq E \times E$ is an (irreflexive) partial order (the *causality relation*) satisfying the *principle of finite causes*: $\forall e \in E : \{d \in E \mid d < e\}$ is finite,
- $\# \subseteq E \times E$ is an irreflexive, symmetric relation (the *conflict relation*) satisfying the *principle of conflict heredity*: $\forall d, e, f \in E : d < e \wedge d \# f \Rightarrow e \# f$,
- $l : E \to Act$ is the *labelling function*.

A prime event structure represents a concurrent system in the following way: action names $a \in Act$ represent actions the system might perform, an event $e \in E$ labelled with a represents an occurrence of a during a possible run of the system, $d < e$ means that d is a prerequisite for e and $d \# e$ means that d and e cannot happen both in the same run.

Causal independence (*concurrency*) of events is expressed by the derived relation $co \subseteq E \times E : d \; co \; e$ iff $\neg(d < e \vee e < d \vee d \# e)$. By definition, $<, >, \#$ and co form a partition of $E \times E$.

Throughout the paper, we assume a fixed set Act of action names as labelling set. Let $\mathbb{E}_{prime}$ denote the domain of prime event structures labelled over Act. The components of a prime event structure $\mathcal{E} \in \mathbb{E}_{prime}$ will be denoted by $E_{\mathcal{E}}, <_{\mathcal{E}}, \#_{\mathcal{E}}$ and $l_{\mathcal{E}}$ — a convention that will also apply to other structures given as tuples. If clear from the context, the index $\mathcal{E}$ will be omitted. A prime event structure $\mathcal{E}$ is *empty* iff $E_{\mathcal{E}}$ is empty, *finite* iff $E_{\mathcal{E}}$ is finite and *conflict-free* iff $\#_{\mathcal{E}} = \emptyset$.

Two prime event structures $\mathcal{E}$ and $\mathcal{F}$ are *isomorphic* ($\mathcal{E} \cong \mathcal{F}$) iff there exists a bijection between their sets of events preserving $<, \#$ and labelling. Generally, we will not distinguish isomorphic event structures.

The behaviour of a prime event structure is described by explaining which subsets of events constitute possible (partial) runs of the represented system (thus formalising the interpretation of the conflict and the causality relation). These subsets are called *configurations*. Configurations have to be conflict-free. Furthermore they must be closed with respect to causal predecessors; all prerequisites for any event occurring in the run must also occur. Configurations may also be considered as possible states of the system. They determine the remaining behaviour of the system as being the set of all events which have not yet occurred and are not excluded because of conflicts.

Definition 2.2 Let $\mathcal{E} \in \mathbb{E}_{prime}$.

(i) A subset $X \subseteq E_{\mathcal{E}}$ of events in $\mathcal{E}$ is *conflict-free in* $\mathcal{E}$ iff $\#_{\mathcal{E}} \cap (X \times X) = \emptyset$.
X is *left-closed in* $\mathcal{E}$ iff, for all $d, e \in E$, $e \in X \wedge d <_{\mathcal{E}} e \Rightarrow d \in X$.

(ii) A subset $X \subseteq E_{\mathcal{E}}$ is a *(finite) configuration* of $\mathcal{E}$ iff X is finite, left-closed and conflict-free in $\mathcal{E}$. $Conf(\mathcal{E})$ denotes the set of all configurations of $\mathcal{E}$. We call a configuration $X \in Conf(\mathcal{E})$ *(successfully) terminated* iff $\forall d \in E_{\mathcal{E}} : d \notin X \Rightarrow \exists e \in X$ with $d \# e$.

Note that a configuration X is terminated iff it is maximal, i.e. $X \subseteq Y \in Conf(\mathcal{E})$ implies $X = Y$.

Unlike [Winskel], we only consider finite configurations here. As usual, we assume that in a finite period of time only finitely many actions may be performed. Now the restriction says that we only consider runs which are executable in a finite period of time. This does not cause a loss of expressiveness, since Winskel's infinite configurations are completely determined by the finite ones. We will further comment on this point in Section 4.

In order to represent causal dependencies in a run of an event structure $\mathcal{E}$, we can take the corresponding configuration X of $\mathcal{E}$ together with the causality relation, $<_X$, inherited from $\mathcal{E}$. A more abstract view of a run is obtained by replacing events by their labels. Formally, this is done below by taking the isomorphism class of $(X, <_X, l \restriction X)$ (which can be seen as the restriction of $\mathcal{E}$ to X), yielding a *partially ordered multiset* (*pomset*). Pomsets as representations of runs of systems have been studied, among others, in [Pratt-a].

Definition 2.3 Let $\mathcal{E} \in \mathbb{E}_{prime}$.

(i) Let $\mathcal{X} = (X, <_X, l_X)$ and $\mathcal{Y} = (Y, <_Y, l_Y)$ be partial orders which are labelled over *Act*. $\mathcal{X}$ and $\mathcal{Y}$ are *isomorphic* ($\mathcal{X} \cong \mathcal{Y}$) iff there exists a bijection between X and Y respecting the ordering and the labelling. The isomorphism class of a partial order labelled over *Act* is called a *pomset over Act*.

(ii) For a configuration $X \in Conf(\mathcal{E})$, $<_X$, the *causality relation on* X, is defined as $<_X := <_{\mathcal{E}} \cap (X \times X)$. The *pomset of* X is given by $pomset(X) := [(X, <_X, l \restriction X)]_{\cong}$.

In graphical representations of prime event structures, only immediate conflicts — not the inherited conflicts — are indicated. The $<$-relation is represented by arcs, omitting those derivable by transitivity. Furthermore, instead of events, only their labels are displayed; if a label occurs twice it represents two different events. Thus these pictures determine event structures only up to isomorphism.

Example 2.1

The system $(a|b) + (a;b;c)$, executing either a and b independently or a, b and c sequentially, may be represented by the prime event structure with events e_1, e_2, e_3, e_4, e_5 with $l(e_1) = l(e_3) = a$, $l(e_2) = l(e_4) = b$ and $l(e_5) = c$, where e_1 *co* e_2, $e_3 < e_4$, $e_4 < e_5$, $e_3 < e_5$ and each of e_1, e_2 is in conflict with each of e_3, e_4, e_5. Following the conventions set out above, this event structure is represented as

$$\begin{array}{l} a \\ \# \\ a \rightarrow b \rightarrow c \quad . \\ \# \\ b \end{array}$$

Its configurations are

$$\emptyset, \{e_1\}, \{e_2\}, \{e_1, e_2\}, \{e_3\}, \{e_3, e_4\}, \{e_3, e_4, e_5\},$$

corresponding to the pomsets

$$\emptyset,\ a,\ b,\ \begin{array}{c} a \\ b \end{array}, a \longrightarrow b \text{ and } a \longrightarrow b \longrightarrow c\,.$$

$\begin{array}{c} a \\ b \end{array}$ and $a \longrightarrow b \longrightarrow c$ correspond to terminated configurations.

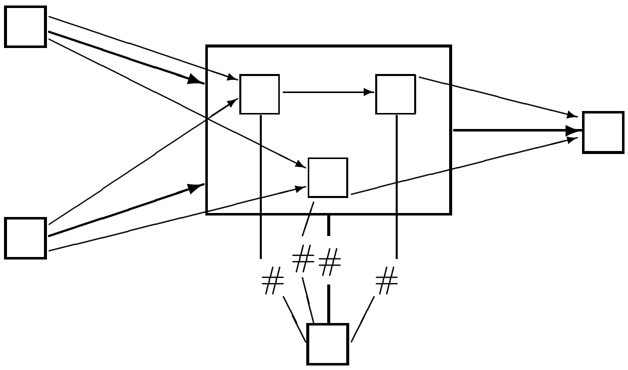
2.2 Refinement of actions

We will now define a refinement operator substituting actions by finite, conflict-free, non-empty event structures. As discussed in the introduction, we will not allow forgetful refinements, replacing actions by the empty event structure. In Section 2.3 we will explain why we have to restrict to finite and conflict-free refinements of actions.

A refinement function will be a function *ref* specifying, for each action a, an event structure *ref* (a) which is to be substituted for a. Interesting refinements (and also the refinements in our examples) will mostly refine only certain actions, hence replace most actions by themselves. However, for uniformity (and for simplicity in proofs) we consider all actions to be refined.

Given an event structure $\mathcal{E}$ and a refinement function *ref*, we construct the refined event structure *ref* $(\mathcal{E})$ as follows. Each event e labelled by a is replaced by a disjoint copy, $\mathcal{E}_e$, of *ref* (a). The causality and conflict structure is inherited from $\mathcal{E}$: every event which was causally before e will be causally before all events of $\mathcal{E}_e$, all events which causally followed e will causally follow all the events of $\mathcal{E}_e$, and all events in conflict with e will be in conflict with all the events of $\mathcal{E}_e$.

Graphically, the idea may be sketched as follows.



Definition 2.4

(i) A function $ref : Act \to \mathbb{E}_{prime}$ is called a *refinement function (for prime event structures)* iff $\forall a \in Act : ref\,(a)$ is non-empty, finite and conflict-free.

(ii) Let $\mathcal{E} \in \mathbb{E}_{prime}$ and let *ref* be a refinement function.
Then $ref\,(\mathcal{E})$ is the prime event structure defined by
– $E_{ref(\mathcal{E})} := \{(e,e')|e \in E_{\mathcal{E}}, e' \in E_{ref(l_{\mathcal{E}}(e))}\}$,
– $(d,d') <_{ref(\mathcal{E})} (e,e')$ iff $d <_{\mathcal{E}} e$ or $(d = e \wedge d' <_{ref(l_{\mathcal{E}}(d))} e')$,
– $(d,d') \#_{ref(\mathcal{E})} (e,e')$ iff $d \#_{\mathcal{E}} e$,
– $l_{ref(\mathcal{E})}(e,e') := l_{ref(l_{\mathcal{E}}(e))}(e')$.

The following proposition states that refinement is a well-defined operation on prime event structures, even when isomorphic prime event structures are identified.

Proposition 2.1

(i) If $\mathcal{E} \in \mathbb{E}_{prime}$ and *ref* is a refinement function then $ref(\mathcal{E})$ is a prime event structure indeed.

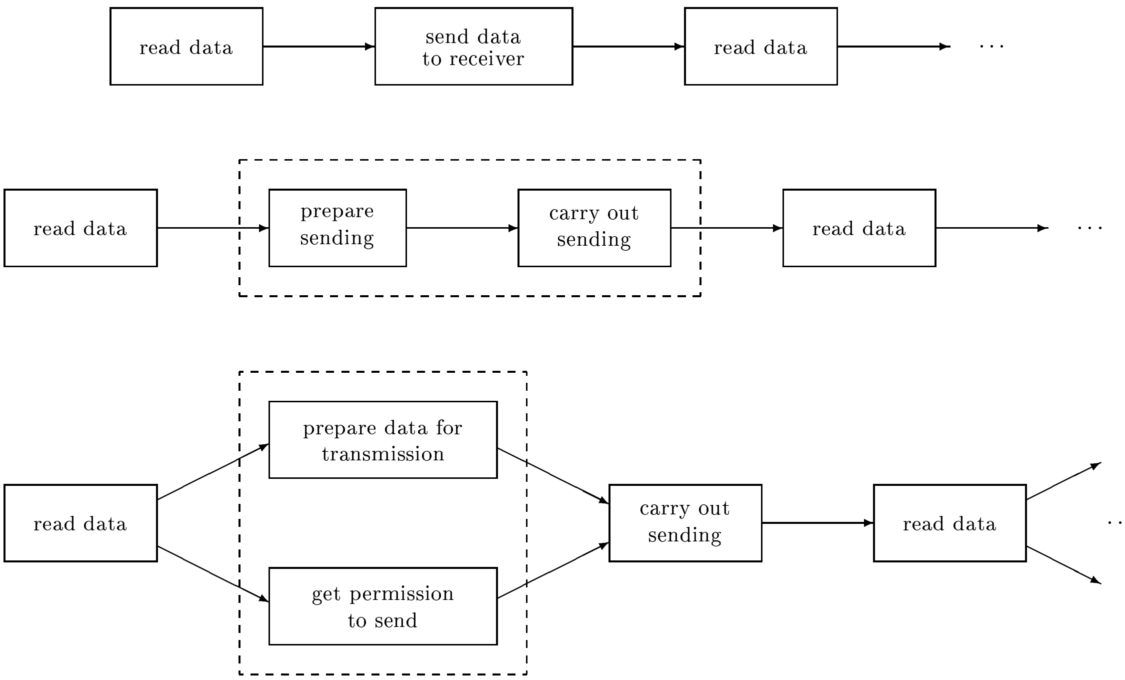
(ii) If $\mathcal{E} \in \mathbb{E}_{prime}$ and *ref*, *ref'* are refinement functions with $ref(a) \cong ref'(a)$ for all $a \in Act$ then $ref(\mathcal{E}) \cong ref'(\mathcal{E})$.

(iii) If $\mathcal{E}, \mathcal{F} \in \mathbb{E}_{prime}$, $\mathcal{E} \cong \mathcal{F}$, and *ref* is a refinement function then $ref(\mathcal{E}) \cong ref(\mathcal{F})$.

Proof Straightforward. ■

Example 2.2

Consider the sender (Example 1.1) from the introduction. As every action occurrence is represented by a separate event, the representation of this system as an event structure is infinite. We may carry out the first two steps of the design in terms of prime event structures as follows.



Refinement by a sequential and a parallel process in prime event structures

The next refinement step, introducing alternative channels for sending, would require a refinement of an action by conflicting behaviours. This is not possible in our framework up to now.

The following proposition shows how the behaviour of the refined event structure $ref(\mathcal{E})$ is determined by the behaviour of $\mathcal{E}$ and by the behaviour of the event structures which are substituted for actions.

Proposition 2.2 Let $\mathcal{E} \in \mathbb{E}_{prime}$, let *ref* be a refinement function.

We call $\widetilde{X}$ a *refinement of configuration* $X \in Conf(\mathcal{E})$ *by ref* iff

- $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ where $\forall e \in X : X_e \in Conf(ref(l_{\mathcal{E}}(e))) - \{\emptyset\}$,
- $e \in busy(\widetilde{X}) \Rightarrow e$ maximal in X with respect to $<_{\mathcal{E}}$ where $busy(\widetilde{X}) := \{e \in X \mid X_e \text{ not terminated}\}$.

Then $Conf\,(ref\,(\mathcal{E})) = \{\widetilde{X} \mid \widetilde{X}$ is a refinement of a configuration $X \in Conf(\mathcal{E})\}$.

Proof As a special case of Proposition 3.3. ■

Hence the configurations of $ref\,(\mathcal{E})$ are exactly those configurations which are refinements of configurations of $\mathcal{E}$. A refinement of a configuration X of $\mathcal{E}$ is obtained by replacing each event e in X by a non-empty configuration X_e of $ref\,(l_{\mathcal{E}}(e))$. Events which are causally necessary for other events in X may only be replaced by terminated configurations.

2.3 The limitations of prime event structures

The reason that so far we only refine actions by conflict-free event structures is the principle of conflict heredity and the notion of configuration in prime event structures. They imply that any event will always occur with a unique history in terms of its causal predecessors. This makes it particularly easy to analyse prime event structures. On the other hand, it makes it is rather difficult to define certain composition operations on them, such as a general sequential composition operator ";" (not just action prefixing).

Example 2.3

Consider $\begin{matrix} c \\ \# \\ d \end{matrix}\ ;\ b$. In this case b can be caused either by c or by d. The simplest prime event structure representation of the resulting behaviour is

$$\begin{matrix} c \longrightarrow b \\ \# \qquad \\ d \longrightarrow b \end{matrix} \quad .$$

It is not possible to represent this behaviour by a prime event structure with only one b-event.

Exactly the same problem occurs when refining a by $c\#d$ in $a \longrightarrow b$. A definition of sequential composition (cf. [BV]), or of a refinement operator on prime event structures allowing conflicts in refinements, would require duplications of events, as illustrated above. This would lead to undesirable complications when using these constructions. We will consider more general forms of event structures that do not require duplication of events in Sections 3 and 4.

The restriction to refinement of actions by finite event structures is necessary to ensure that the resulting event structure will obey the principle of finite causes. Again the same problem occurs already in a simple sequential composition.

Example 2.4

Consider the sequential composition of an infinite process $a_1 \longrightarrow a_2 \longrightarrow \cdots$ with a process b. Since, following e.g. [Winskel], we made the assumption that in a finite time only finitely many actions can happen, the only consistent interpretation of the resulting system is the one in which b will never occur. As in a prime event structure there are no "dead" events (every event occurs in some configuration), the only way to represent this system is by erasing the b-event.

A definition of sequential composition, or of a refinement operator on prime event structures allowing infinite refinements, would hence again lead to undesirable complications. In the more general models we will consider later, the principle of finite causes will be abandoned, and this will allow also refinements by infinite behaviours as discussed in the introduction.

Also when considering a parallel composition operator, it may be necessary to duplicate or erase events when using prime event structures.

Example 2.5

Consider the parallel composition operator $\|_A$, where $A \subseteq Act$, from TCSP [OH]. In $P \|_A Q$, P and Q are executed in parallel; however any action in A may only be executed by both P and Q together as one joint action.

Let $P := a; b; c$ and $Q := a'; b; c'$, representable by the prime event structures

$$a \longrightarrow b \longrightarrow c \quad \text{and} \quad a' \longrightarrow b \longrightarrow c' .$$

A prime event structure for $P \|_{\{b\}} Q$ is obtained by taking the union of the event structures for P and Q and identifying the b-events:

$$\begin{tikzcd} a \arrow[dr] & & a' \\ & b \arrow[ur] \arrow[dr] & \\ c \arrow[ur] & & c' \end{tikzcd} .$$

However, in general the situation is not so simple. Consider e.g.

$$\begin{array}{l} a \\ \# \\ c \longrightarrow a \end{array} \qquad \|_{\{a\}} \quad a \longrightarrow b \quad .$$

Here, we will have two possibilities for b to occur: Either immediately after an a-communication (choosing the upper a-event on the left side) or after c followed by an a-communication. Because of the unique history of events, this may again only be represented by duplicating the b-event:

$$\begin{array}{l} a \longrightarrow b \\ \# \\ c \longrightarrow a \longrightarrow b \end{array} .$$

Example 2.6

Consider

$$a \longrightarrow b \quad ||_{\{a,b\}} \quad b \longrightarrow a \quad .$$

In the resulting system, no event can occur because of the cyclic causal dependency between the a- and the b-communication. Since in prime event structures there are no dead events, the only prime event structure to represent this is the empty one, which is not obtainable by a construction as suggested in Example 2.5.

This shows that the parallel composition for prime event structures may not simply be defined by taking the cartesian product of the events that have to synchronise and the union of the other events. One has to determine which events can actually take place and to duplicate events which get different histories. This leads to a rather complicated definition [LG, DDM-b, Vaandrager-a].

Thus, it is possible, though somewhat complicated, to define parallel composition as well as action refinement (or just sequential composition) on the prime event structures considered here. However, having both operations around at the same time is problematic. In the definition of action refinement, we implicitly assume that all maximal configurations represent successful termination (cf. Definition 2.2). This implies that we can not model systems that may end in deadlock. A parallel composition operator as in Example 2.5 on the other hand (or the *restriction operator* of CCS) can introduce deadlocks; defining it on the prime event structures considered here implicitly assumes that one represents systems that may have deadlocks, but that deadlock and successful termination are not distinguished. Although each of these two implicit assumptions is consistent, their combination is not, as illustrated by the following example.

Example 2.7

The system $P := (a\,;b\,;c)||_{\{a,b,c\}}(a\,;c\,;b)$ constitutes a typical example of a deadlock situation. After the occurrence of the action a, each component in the parallel composition is waiting in vain for an opportunity to synchronise with the other component; yet the system will never perform any further action. In view of the approach to sequential composition outlined in Section 1.2, this implies that in the system $P\,;d$ the action d will never occur.

However, in the model of prime event structures as considered here, P can only be represented by the event structure consisting of a single event labelled a (regardless of the precise definition of $||_A$). Hence $P\,;d$ will denote the event structure $a \longrightarrow d$, in which the action d *can* occur.

The same inconsistency occurs when refining e into P in the structure $e \longrightarrow d$.

This problem can be avoided by distinguishing deadlock from successful termination. We will do so in the following sections, simultaneously considering more general kinds of event structures in order to avoid the erasure and duplication of events encountered in the preceding examples.

3 Flow event structures and action refinement

In the previous section, we have indicated that for refining actions by event structures with conflicts, or by infinite event structures, as well as for defining sequential and parallel composition, more general models than prime event structures are appropriate. In [BC-b] a form of event structures, called *flow event structures*, has been proposed which is particularly suited for giving semantics to languages like CCS.

In this section, we introduce flow event structures following closely [Boudol-b]. We show that they allow for a straightforward generalisation of Definition 2.4 to refinement by arbitrary non-empty processes, and that deadlocks can be expressed appropriately. Moreover, we establish that the behaviour of a refined event structure may be deduced compositionally from the behaviours of the original event structure and of the refinements of actions.

3.1 Flow event structures

Flow event structures have a similar representation as prime event structures, but use a (not necessarily transitive) relation which represents "possible direct" causes instead of the transitive causality relation $<$. They even allow for "syntactic" cycles in this so-called *flow relation*. This is convenient for modelling parallel composition. The axiom of conflict heredity is dropped, as well as the axiom of finite causes. The conflict relation may be reflexive, such that "inconsistent" events are possible which will never occur. These are convenient to model restriction in CCS.

Definition 3.1

A *(labelled) flow event structure (over an alphabet Act)* is a 4-tuple $\mathcal{E} = (E, \prec, \#, l)$ where

- E is a set of *events*,
- $\prec \subseteq E \times E$ is an irreflexive relation (the *flow relation*),
- $\# \subseteq E \times E$ is a symmetric relation (the *conflict relation*),
- $l : E \to Act$ is the *labelling function*.

Let $\mathbb{E}_{\mathit{flow}}$ denote the domain of flow event structures labelled over Act. O denotes the empty flow event structure $(\emptyset, \emptyset, \emptyset, \emptyset)$.

Two flow event structures $\mathcal{E}$ and $\mathcal{F}$ are *isomorphic* ($\mathcal{E} \cong \mathcal{F}$) iff there exists a bijection between their sets of events preserving $\prec, \#$ and labelling.

The interpretation of the conflict and the flow relation is formalised by defining configurations of flow event structures. Configurations have to be conflict-free; in particular, self-conflicting events will never occur in any configuration. $d \prec e$ will mean that d is a *possible immediate cause* for e. For an event to occur it is necessary that a *complete* non-conflicting set of its causes has occurred. Here a set of causes is complete iff for any cause which is not contained there is a conflicting event which is contained. Finally, no cycles with respect to causal dependence may occur.

Definition 3.2 Let $\mathcal{E} \in \mathbb{E}_{flow}$.

(i) $X \subseteq E$ is *conflict-free in* $\mathcal{E}$ iff $\# \cap (X \times X) = \emptyset$.
$X \subseteq E$ is *left-closed in* $\mathcal{E}$ *up to conflicts* iff $\forall d, e \in E$:
if $e \in X, d \prec e$ and $d \notin X$ then there exists an $f \in X$ with $f \prec e$ and $d \# f$.

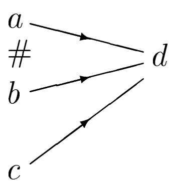
(ii) $X \subseteq E$ is a *(finite) configuration* of $\mathcal{E}$ iff X is finite, left-closed up to conflicts and conflict-free and does not contain a causality cycle, i.e. $<_X := (\prec \cap (X \times X))^+$ (where $^+$ denotes transitive closure) is irreflexive.
Conf $(\mathcal{E})$ denotes the set of all configurations of $\mathcal{E}$.
A configuration X is called *maximal* iff $X \subseteq Y \in$ *Conf* $(\mathcal{E})$ implies $X = Y$.
A configuration X is called *(successfully) terminated* iff $\forall d \in E : d \notin X \Rightarrow \exists e \in X$ with $d \# e$.

The causal dependence between action occurrences in a configuration may again, as for prime event structures, be represented by a pomset; for $X \in$ *Conf* $(\mathcal{E})$, we take the isomorphism class of $(X, <_X, l_{\mathcal{E}} \restriction X)$.

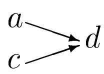
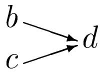
In graphical representations we omit names of events and represent $\prec$ by arcs of the form $\longrightarrow$. A self-conflicting event labelled d is denoted $\boxed{d}$.

Example 3.1

The system $((a + b) \mid c); d$ may be represented by the flow event structure



.

The pomsets  and  correspond to terminated configurations.

Note that prime event structures can be regarded as special flow event structures by defining $d \prec e$ iff $d < e$. The requirement of left-closedness up to conflicts for configurations then reduces to left-closedness with respect to $<$; hence the notions of a configuration according to Definitions 2.2 and 3.2 coincide. Furthermore, in each configuration X, the relation $<_X$ of Definition 2.3 coincides with the one of Definition 3.2.

3.2 Deadlocks in flow event structures

We have introduced the set of configurations to formalise the intuitive meaning of flow event structures. When considering flow event structures as a model for CCS, this is sufficient. However, when considering sequential composition, we have to distinguish successful termination and deadlock. We will show here that this distinction is possible in flow event structures, but is lost when describing a flow event structure merely by its set of configurations. We will look at configurations more closely in order to distinguish configurations corresponding to successful termination from those corresponding to deadlock states.

We start by defining a sequential composition operation. For flow event structures, a simple and natural definition may be given by putting all events in the first component in $\prec$-relation with all events of the second component.

Definition 3.3 Let $\mathcal{E}, \mathcal{F} \in \mathbb{E}_{\mathit{flow}}$, w.l.o.g. $E_{\mathcal{E}} \cap E_{\mathcal{F}} = \emptyset$.

$$\text{Then } \mathcal{E}; \mathcal{F} := \quad (E_{\mathcal{E}} \cup E_{\mathcal{F}}, \prec_{\mathcal{E}} \cup \prec_{\mathcal{F}} \cup (E_{\mathcal{E}} \times E_{\mathcal{F}}), \#_{\mathcal{E}} \cup \#_{\mathcal{F}}, l_{\mathcal{E}} \cup l_{\mathcal{F}}).$$

We are now able to distinguish deadlocking from successfully terminating behaviours by considering them as first components in a sequential composition.

Example 3.2

Consider the event structure O (the empty process) and an event structure $D := \boxed{d}$ containing just one inconsistent event. We have $\mathit{Conf}(O) = \{\emptyset\} = \mathit{Conf}(D)$. However, in $O; a$ the action a will occur, whereas in $D; a$ it will not. Hence O represents a *skip*-process which terminates successfully without executing any action, but D deadlocks.

Behaviours that cannot be prolonged are modelled by maximal configurations. In prime event structures, every maximal configuration X terminates successfully: every event which is not contained in X is in conflict with some event in X. However, this is not true for flow event structures: there may be configurations which are maximal but not terminated, and these configurations may be interpreted as deadlock situations. We will show that this captures exactly deadlocking behaviour with respect to our notion of sequential composition.

Definition 3.4 Let $\mathcal{E} \in \mathbb{E}_{\mathit{flow}}$.

$X \in \mathit{Conf}(\mathcal{E})$ is called a *deadlock* iff X is maximal but not terminated.
$\mathcal{E}$ is called *deadlock-free* iff $\mathcal{E}$ has no deadlock configurations.

Example 3.2 (continued)

As expected, O is deadlock-free whereas D deadlocks; the only configuration $\emptyset$ of D is not terminated.

Deadlocks may not only be caused by inconsistent events. Obviously causality cycles may also cause deadlocks, as in $a\,;b \parallel_{\{a,b\}} b\,;a$ (cf. Example 2.6). A third cause of deadlocks originates from the fact that in flow event structures "syntactic" and "semantic" conflict do not necessarily coincide. Two events are in *semantic* conflict iff there is no configuration containing them both. Flow event structures where syntactic and semantic conflict coincide are called *faithful* in [Boudol-b].

Example 3.3

Let $\mathcal{E} := \begin{array}{ccc} a & \longrightarrow & c \\ \# & & \\ b & & \end{array}$. In $\mathcal{E}$ we reach a deadlock by executing b, since c is disabled without being in syntactic conflict with b. In $\mathcal{E}; d$ it may happen that d cannot occur. However, the faithful event structure corresponding to $\mathcal{E}$, $\begin{tikzcd}[row sep=small, column sep=small] a \arrow[r] & c \\ \# & \# \\ b \arrow[ur, dotted, no head] & \end{tikzcd}$, is deadlock-free.

The next example shows that deadlocks may occur also in faithful event structures without self-conflicting events and without causality cycles.

Example 3.4 [Schreiber]

$$\text{Let } \mathcal{E} := \begin{tikzcd}[row sep=tiny] a_1 & \\ \# & \\ b_1 \arrow[dr] & \\ \# & c \\ b_2 \arrow[ur] & \\ \# & \\ a_2 & \end{tikzcd}$$

The configuration containing a_1 and a_2 is maximal but not terminated.

We will now show that our notion of termination indeed captures exactly the behaviours which, when occuring in the first component of a sequential composition, allow action occurrences in the second component. Hence deadlock configurations are precisely those maximal configurations that prevent action occurrences in the second component. Note however that this depends on our definition of sequential composition and on the notion of configuration for flow event structures.

Proposition 3.1 Let $\mathcal{E}, \mathcal{F} \in \mathbb{E}_{\mathit{flow}}$.

Then $\mathit{Conf}(\mathcal{E};\mathcal{F}) = \mathit{Conf}(\mathcal{E}) \cup \{X \cup Y \mid X \in \mathit{Conf}(\mathcal{E}),\ X \text{ terminated},\ Y \in \mathit{Conf}(\mathcal{F})\}$.
Moreover, $Z \in \mathit{Conf}(\mathcal{E};\mathcal{F})$ is terminated iff $Z \cap E_{\mathcal{E}}$ and $Z \cap E_{\mathcal{F}}$ are terminated.

Proof Let $\mathcal{E}, \mathcal{F} \in \mathbb{E}_{\mathit{flow}}$.
"$\supseteq$" and "if": Suppose $X \in \mathit{Conf}(\mathcal{E})$ .Then X is finite. As X is left-closed up to conflicts, conflict-free and without causality cycles in $\mathcal{E}$, it must be left-closed up to conflicts, conflict-free and without causality cycles in $\mathcal{E};\mathcal{F}$. Hence $X \in \mathit{Conf}(\mathcal{E};\mathcal{F})$.

Now suppose $X \in Conf(\mathcal{E})$ is terminated and $Y \in Conf(\mathcal{E})$. We show that $X \cup Y \in Conf(\mathcal{E};\mathcal{F})$. Clearly, $X \cup Y$ is finite and conflict-free, and $<_{X\cup Y}$ contains no causality cycles. So we show that $X \cup Y$ is left-closed up to conflicts.

Let $d, e \in E_{\mathcal{E};\mathcal{F}}$, $d \notin X \cup Y$ and $d \prec_{\mathcal{E};\mathcal{F}} e \in X \cup Y$. If $d, e \in E_{\mathcal{E}}$ or $d, e \in E_{\mathcal{F}}$ we may apply left-closedness up to conflicts of X or Y, respectively. Hence we only need to consider the case $d \in E_{\mathcal{E}}$, $e \in Y$. As $d \notin X$, there is an $f \in X$ with $d \#_{\mathcal{E}} f$ (since X is terminated). Since $f \in E_{\mathcal{E}}$ and $e \in E_{\mathcal{F}}$, we have $d \#_{\mathcal{E};\mathcal{F}} f \prec_{\mathcal{E};\mathcal{F}} e$.

In case X and Y are both terminated it follows immediately that also $X \cup Y$ is terminated.

"$\subseteq$" and "only if": Let $Z \in Conf(\mathcal{E};\mathcal{F})$ and let $X := Z \cap E_{\mathcal{E}}$ and $Y := Z \cap E_{\mathcal{F}}$. As Z is finite, left-closed up to conflicts, conflict-free and without causality cycles, so are X and Y. In case $Y = \emptyset$ we have $Z = X \in Conf(\mathcal{E})$. Otherwise, it only remains to be shown that X is terminated. Take $e \in Y$. Let $d \in E_{\mathcal{E}}$, $d \notin X$. Then $d \prec_{\mathcal{E};\mathcal{F}} e$, and since Z is left-closed up conflicts there is an $f \in Z$ such that $d \#_{\mathcal{E};\mathcal{F}} f \prec_{\mathcal{E};\mathcal{F}} e$. As $d \in E_{\mathcal{E}}$ and $d \#_{\mathcal{E};\mathcal{F}} f$ it must be that $f \in E_{\mathcal{E}}$ and $d \#_{\mathcal{E}} f$, and since $f \in Z$ we have $f \in X$. Hence X is terminated.

In case Z is terminated it follows immediately that both X and Y are terminated. ∎

We conclude this section by discussing our notion of deadlock with respect to infinite behaviours. In this case, sequential composition does not provide an obvious criterion.

Example 3.5

Consider a process executing an infinite sequence of a's. It may be represented by the event structure

$$\mathcal{E} := a \longrightarrow a \longrightarrow a \longrightarrow \cdots \quad .$$

In $a\,;b$, b will not occur in any configuration of $\mathcal{E}$, even when considering infinite configurations as in [Boudol-b] (since the axiom of finite causes is kept for configurations). Nevertheless, $\mathcal{E}$ is clearly deadlock-free.

Now consider the event structure

$$\mathcal{F} := \begin{array}{l} a \longrightarrow a \longrightarrow a \longrightarrow \cdots \\ \\ \boxed{d} \end{array}$$

where $\mathcal{E}$ is put in parallel with a deadlocking process. $\mathcal{F}$ is deadlock-free according to our definition. One could argue that a parallel composition should deadlock whenever one of its components deadlocks and then $\mathcal{F}$ should not be regarded as deadlock-free. Alternatively, one can say that a parallel composition deadlocks when some components deadlock, and all others have terminated successfully. This is captured by our definition.

3.3 Refinement of actions

Refinement of actions in flow event structures may now be defined as follows. We assume a refinement function $\mathit{ref}: \mathit{Act} \to \mathbb{E}_{\mathit{flow}} - \{O\}$ mapping actions to non-empty flow event structures, and replace each event labelled by a by a disjoint copy of $\mathit{ref}(a)$. The conflict and causality structure will just be inherited.

Hence, we may replace actions also by behaviours with conflicts and by infinite behaviours.

Definition 3.5

(i) A function $\mathit{ref}: \mathit{Act} \to \mathbb{E}_{\mathit{flow}} - \{O\}$ is called a *refinement function (for flow event structures)*.

(ii) Let $\mathcal{E} \in \mathbb{E}_{\mathit{flow}}$ and let ref be a refinement function.
Then the *refinement of* $\mathcal{E}$ *by ref*, $\mathit{ref}(\mathcal{E})$, is the flow event structure defined by
– $E_{\mathit{ref}(\mathcal{E})} := \{(e, e') | e \in E_{\mathcal{E}}, e' \in E_{\mathit{ref}(l_{\mathcal{E}}(e))}\}$,
– $(d, d') \prec_{\mathit{ref}(\mathcal{E})} (e, e')$ iff $d \prec_{\mathcal{E}} e$ or $(d = e \wedge d' \prec_{\mathit{ref}(l_{\mathcal{E}}(d))} e')$,
– $(d, d') \#_{\mathit{ref}(\mathcal{E})} (e, e')$ iff $d \#_{\mathcal{E}} e$ or $(d = e \wedge d' \#_{\mathit{ref}(l_{\mathcal{E}}(d))} e')$,
– $l_{\mathit{ref}(\mathcal{E})}(e, e') := l_{\mathit{ref}(l_{\mathcal{E}}(e))}(e')$.

As for prime event structures, we verify that $\mathit{ref}(\mathcal{E})$ is well-defined, even when isomorphic flow event structures are identified.

Proposition 3.2

(i) If $\mathcal{E} \in \mathbb{E}_{\mathit{flow}}$ and ref is a refinement function then $\mathit{ref}(\mathcal{E})$ is a flow event structure.

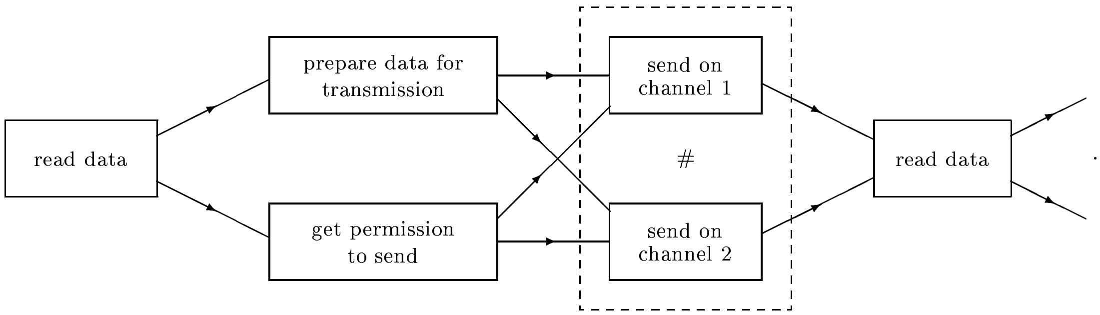
(ii) If $\mathcal{E} \in \mathbb{E}_{\mathit{flow}}$ and ref, ref' are refinement functions with $\mathit{ref}(a) \cong \mathit{ref}'(a)$ for all $a \in \mathit{Act}$ then $\mathit{ref}(\mathcal{E}) \cong \mathit{ref}'(\mathcal{E})$.

(iii) If $\mathcal{E}, \mathcal{F} \in \mathbb{E}_{\mathit{flow}}$, $\mathcal{E} \cong \mathcal{F}$, and ref is a refinement function then $\mathit{ref}(\mathcal{E}) \cong \mathit{ref}(\mathcal{F})$.

Proof Straightforward. ■

Example 3.6

Continuing Example 2.2, the third step in the design of the sender from the introduction yields the following flow event structure, which is not prime.



Refinement by alternative actions in flow event structures

Next, we show that, analogously to prime event structures, the behaviour of a refined flow event structure $\mathit{ref}\,(\mathcal{E})$ may be deduced compositionally from the behaviour of $\mathcal{E}$ and the behaviour of the refinements of actions, where the behaviour of a flow event structure is given by its configurations and their termination status.

Proposition 3.3

Let $\mathcal{E} \in \mathbb{E}_{\mathit{flow}}$, let ref be a refinement function for flow event structures.

We call $\widetilde{X}$ a *refinement of configuration* $X \in \mathit{Conf}\,(\mathcal{E})$ *by ref* iff

- $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ where $\forall e \in X : X_e \in \mathit{Conf}\,(\mathit{ref}\,(l_{\mathcal{E}}(e))) - \{\emptyset\}$,
- $e \in \mathit{busy}(\widetilde{X}) \Rightarrow e$ maximal in X with respect to $<_X$ where $\mathit{busy}\,(\widetilde{X}) := \{e \in X \mid X_e \text{ not terminated}\}$.

Then $\mathit{Conf}\,(\mathit{ref}\,(\mathcal{E})) = \{\widetilde{X} \mid \widetilde{X}$ is a refinement of a configuration $X \in \mathit{Conf}\,(\mathcal{E})\}$. Moreover, $\widetilde{X}$ is terminated iff X is terminated and $\forall e \in X : X_e$ is terminated.

Proof For any set $\widetilde{X} \subseteq E_{\mathit{ref}(\mathcal{E})}$ of events in $\mathit{ref}\,(\mathcal{E})$, define the projections

$$\pi_1(\widetilde{X}) := \{e \mid \exists f : (e,f) \in \widetilde{X}\} \qquad \text{and} \qquad \pi_2^e(\widetilde{X}) := \{f \mid (e,f) \in \widetilde{X}\}.$$

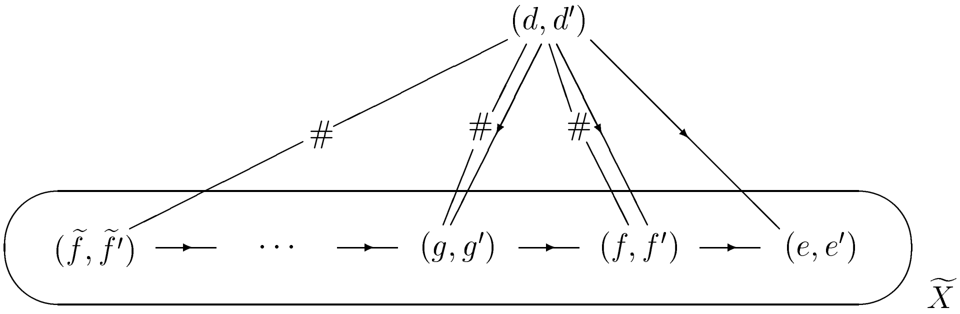
Now $\widetilde{X}$ can be written as $\widetilde{X} = \bigcup_{e \in \pi_1(\widetilde{X})} \{e\} \times \pi_2^e(\widetilde{X})$.

"$\subseteq$" Let $\widetilde{X} \in \mathit{Conf}\,(\mathit{ref}\,(\mathcal{E}))$.

First we show that $X := \pi_1(\widetilde{X}) \in \mathit{Conf}\,(\mathcal{E})$.

X is finite since $\widetilde{X}$ is finite.

X is left-closed in $\mathcal{E}$ up to conflicts:
Let $e \in X,\ d \in E_{\mathcal{E}}$ with $d \prec_{\mathcal{E}} e$ and $d \notin X$.
We have to show that there exists an $f \in X$ with $f \prec_{\mathcal{E}} e$ and $f \#_{\mathcal{E}} d$.
Since $e \in X$ there must be some $(e,e') \in \widetilde{X}$.
There exists $(d,d') \in E_{\mathit{ref}(\mathcal{E})},\ (d,d') \notin \widetilde{X}$ since $\mathit{ref}\,(d) \neq O$ and $d \notin X$.
Furthermore $(d,d') \prec_{\mathit{ref}(\mathcal{E})} (e,e')$ since $d \prec_{\mathcal{E}} e$.
So there exists $(f,f') \in \widetilde{X}$ with $(f,f') \prec_{\mathit{ref}(\mathcal{E})} (e,e')$ and $(f,f') \#_{\mathit{ref}(\mathcal{E})} (d,d')$.
$f \neq d$ since $f \in X, d \notin X$; hence $f \#_{\mathcal{E}} d$.
If $f \neq e$ we have $f \prec_{\mathcal{E}} e$ and we are done.
Assume $f = e$, then $(d,d') \prec_{\mathit{ref}(\mathcal{E})} (f,f')$.
So there exists $(g,g') \in \widetilde{X}$ with $(g,g') \prec_{\mathit{ref}(\mathcal{E})} (f,f') = (e,f')$ and $(g,g') \#_{\mathit{ref}(\mathcal{E})} (d,d')$.
As $\widetilde{X}$ has no causality cycles, $(g,g') \neq (e,e')$. $g \in X$, so $g \neq d$, and hence $g \#_{\mathcal{E}} d$.
If $g \neq f = e$ then $g \prec_{\mathcal{E}} e$ and we are done.
Since $\widetilde{X}$ is finite, we will find (by repeating this), after finitely many steps, $(\widetilde{f}, \widetilde{f}') \in \widetilde{X}$ with $\widetilde{f} \#_{\mathcal{E}} d$ and $\widetilde{f} \prec_{\mathcal{E}} e$. Hence X is left-closed up to conflicts.



X is conflict-free:
Assume $d, e \in X$ with $d \#_{\mathcal{E}} e$. Then there exist $(d, d'), (e, e') \in \widetilde{X}$ with $(d, d') \#_{\mathit{ref}(\mathcal{E})} (e, e')$. This is a contradiction since $\widetilde{X}$ is conflict-free.

Finally we have to show that X *does not contain a causality cycle.*
Assume $d, e \in X,\ d \neq e,\ d <_X e$ and $e <_X d$ (where $<_X := (\prec_{\mathcal{E}} \cap (X \times X))^+$).
It is straightforward to verify that this implies that there are $(d, d'), (e, e') \in \widetilde{X}$ with $(d, d') \neq (e, e'),\ (d, d') <_{\widetilde{X}} (e, e')$ and $(e, e') <_{\widetilde{X}} (d, d')$.
This is in contradiction with the cyclefreeness of $\widetilde{X}$.

Hence $X \in \mathit{Conf}\,(\mathcal{E})$.

Next we will show that $\widetilde{X}$ is a refinement of X.

Let $e \in X$. Let $\mathcal{E}_e := \mathit{ref}\,(l_{\mathcal{E}}(e))$. We want to show that $\pi_2^e(\widetilde{X}) \in \mathit{Conf}\,(\mathcal{E}_e) - \{\emptyset\}$.
By construction $\pi_2^e(\widetilde{X}) \neq \emptyset$ and obviously $\pi_2^e(\widetilde{X}) \subseteq E_{\mathcal{E}_e}$.
$\pi_2^e(\widetilde{X})$ is finite, conflict-free and cycle-free since $\widetilde{X}$ is finite, conflict-free and cycle-free.
So it only remains to be shown that $\pi_2^e(\widetilde{X})$ is left-closed up to conflicts.

Let $d' \in \mathcal{E}_e,\ d' \prec_{\mathcal{E}_e} e' \in X_e,\ d' \notin X_e$.
Then $(e, d') \in E_{\mathit{ref}(\mathcal{E})},\ (e, d') \prec_{\mathit{ref}(\mathcal{E})} (e, e') \in \widetilde{X}$ and $(e, d') \notin \widetilde{X}$.
So there exists $(f, f') \in \widetilde{X}$ with $(f, f') \prec_{\mathit{ref}(\mathcal{E})} (e, e')$ and $(f, f') \#_{\mathit{ref}(\mathcal{E})} (e, d')$.
As $f, e \in X$ we have $\neg(f \#_{\mathcal{E}} e)$, so $f = e \wedge f' \#_{\mathcal{E}_e} d'$. Thus $f' \in X_e$ and $f' \prec_{\mathcal{E}_e} e'$.
Hence $\pi_2^e(\widetilde{X}) \in \mathit{Conf}\,(\mathcal{E}_e)$.

From what we have shown by now it follows that $\widetilde{X} = \bigcup\limits_{e \in X} \{e\} \times X_e$ with $X \in \mathit{Conf}\,(\mathcal{E})$ and, for all $e \in X$, $X_e \in \mathit{Conf}\,(\mathit{ref}\,(l_{\mathcal{E}}(e))) - \{\emptyset\}$.

Now let $e \in \mathit{busy}\,(\widetilde{X})$. We have to show that e is maximal in X with respect to $<_X$.

Suppose e is not maximal in X.
Then there exists $f \in X$ with $e \prec_{\mathcal{E}} f$, and there is an $(f, f') \in \widetilde{X}$.
Since $\pi_2^e(\widetilde{X})$ is not terminated there exists $d' \in E_{\mathcal{E}_e} - \pi_2^e(\widetilde{X})$ with

$$(*) \qquad \forall e' \in \pi_2^e(\widetilde{X}) : \neg(d' \#_{\mathcal{E}_e} e').$$

We have $(e, d') \prec_{\mathit{ref}\,(\mathcal{E})} (f, f')$ and $(e, d') \notin \widetilde{X}$.
Since $\widetilde{X}$ is a configuration, there then exists $(g, g') \in \widetilde{X}$ with $(g, g') \prec_{\mathit{ref}\,(\mathcal{E})} (f, f')$ and

$(g, g')\#_{\mathit{ref}\,(\mathcal{E})}(e, d')$. As $g, e \in X$, we have $\neg(g\#_{\mathcal{E}}e)$. Hence $g = e$, $g' \in \pi_2^e(\widetilde{X})$ and $g'\#_{\mathcal{E}_e}d'$. However this contradicts $(*)$.

"$\supseteq$" Let $\widetilde{X}$ be a refinement of $X \in \mathit{Conf}\,(\mathcal{E})$. We show that $\widetilde{X} \in \mathit{Conf}\,(\mathit{ref}\,(\mathcal{E}))$.

It follows in a straightforward manner from the corresponding properties of X and the X_e's that $\widetilde{X}$ is finite and conflict-free and contains no causality cycles.
Hence it suffices to show that $\widetilde{X}$ is left-closed up to conflicts.

So let $(e, e') \in \widetilde{X}$, let $(d, d') \in E_{\mathit{ref}(\mathcal{E})} - \widetilde{X}$ with $(d, d') \prec_{\mathit{ref}(\mathcal{E})} (e, e')$. We have to show that there exists $(f, f') \in \widetilde{X}$ with $(f, f') \prec_{\mathit{ref}(\mathcal{E})} (e, e')$ and $(f, f')\#_{\mathit{ref}(\mathcal{E})}(d, d')$.
In case $d = e$ this follows immediately from the corresponding property of X_e.
So let $d \neq e$.
If $d \notin X$ then the requirement follows from the corresponding property of X.
So we now consider the remaining case that $d \neq e$ and $d \in X$. Then $d' \notin X_d$.
Since $d \neq e$ we have $d \prec_{\mathcal{E}} e$, hence d is not maximal in X.
Thus X_d must be terminated.
So $d' \notin X_d$ implies $\exists f' \in X_d$ with $f'\#_{\mathit{ref}(l_{\mathcal{E}}(d))}d'$.
Hence $(d, f') \in \widetilde{X}$, $(d, f') \prec_{\mathit{ref}(\mathcal{E})} (e, e')$ and $(d, f')\#_{\mathit{ref}(\mathcal{E})}(d, d')$.

This completes the proof of the first statement of the proposition. We now turn to the second.

"$\Rightarrow$" Let $\widetilde{X} \in \mathit{Conf}(\mathit{ref}(\mathcal{E}))$ be terminated.
First we show that $X = \pi_1(\widetilde{X})$ is terminated.
Let $d \in E_{\mathcal{E}} - X$. We have to show: $\exists e \in X$ with $d\#_{\mathcal{E}}e$.
Since $\mathcal{E}_d \neq O$, there exists $(d, d') \in E_{ref(\mathcal{E})}$ with $(d, d') \notin \widetilde{X}$.
Since $\widetilde{X}$ is terminated, there exists $(e, e') \in \widetilde{X}$ with $(d, d')\#_{ref(\mathcal{E})}(e, e')$, $e \in X$.
$d = e$ would imply $d'\#_{\mathcal{E}_e}e'$, however $d', e' \in \pi_2^e(\widetilde{X})$, which is conflict-free.
Hence $d \neq e$ and thus $d\#_{\mathcal{E}}e$.

Next we show, for $e \in X$, that $\pi_2^e(\widetilde{X})$ is terminated.
Let $e \in X$. Let $d' \in E_{\mathcal{E}_e} - \pi_2^e(\widetilde{X})$. We have to show: $\exists e' \in \pi_2^e(\widetilde{X})$ with $d'\#_{\mathcal{E}_e}e'$.
We have $(e, d') \notin \widetilde{X}$.
Since $\widetilde{X}$ is terminated, there exists $(f, e') \in \widetilde{X}$ with $(f, e')\#_{ref(\mathcal{E})}(e, d')$.
However, $f = e$ since otherwise $e\#_{\mathcal{E}}f$, but $e, f \in X$.
Thus $e' \in \pi_2^e(\widetilde{X})$ and $d'\#_{\mathcal{E}_e}e'$.

"$\Leftarrow$" Assume X is terminated and, for all $e \in X$, X_e is terminated.
We show that $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ is terminated.
Let $(d, d') \in E_{ref(\mathcal{E})} - \widetilde{X}$.
First consider the case that $d \notin X$.
Then $\exists e \in X$ with $d\#_{\mathcal{E}}e$. Since $X_e \neq \emptyset$, $\exists e' \in E_{\mathcal{E}_e}$ with $(e, e') \in \widetilde{X}$ and $(d, d')\#_{ref(\mathcal{E})}(e, e')$.
Now let $d \in X$.
As $(d, d') \notin \widetilde{X}$ we have $d' \notin X_d$, so $\exists e' \in X_d$ with $d'\#_{\mathcal{E}_e}e'$ (since X_d is terminated).
Thus $(d, e') \in \widetilde{X}$ and $(d, d')\#_{ref(\mathcal{E})}(d, e')$. ■

The following proposition shows that sequential composition for non-empty flow event structures can be regarded as a special case of action refinement.

Proposition 3.4 Let $\mathcal{E}$ and $\mathcal{F}$ be non-empty flow event structures, and $a, b \in \mathit{Act}$.

Let *ref* be a refinement function with $\mathit{ref}(a) = \mathcal{E}$ and $\mathit{ref}(b) = \mathcal{F}$. Then

$$\mathcal{E};\mathcal{F} \cong \mathit{ref}(a \rightarrow b).$$

Proof Let a and b be the two events in the event structure $a \rightarrow b$ (where a is the event labelled a, and b the one labelled b). Let $\mathcal{G}$ be the event structure $\mathit{ref}(a \rightarrow b)$. In Definition 3.3 of the sequential composition operator ; it is assumed that $E_{\mathcal{E}} \cap E_{\mathcal{F}} = \emptyset$. Now the function $i : E_{\mathcal{E};\mathcal{F}} \to E_{\mathcal{G}}$ given by $i(e) := (a, e)$ if $e \in E_{\mathcal{E}}$ and $i(e) := (b, e)$ if $e \in E_{\mathcal{F}}$ is clearly an isomorphism. ■

The following adaptation of Proposition 3.1 can now be obtained as a corollary of Proposition 3.3.

Corollary 3.1 Let $\mathcal{E}$, $\mathcal{F}$ and *ref* be as in Proposition 3.4.

Then $\mathit{Conf}(\mathit{ref}(a \rightarrow b)) = \{(\{a\} \times X_a) \cup (\{b\} \times X_b) \mid X_a \in \mathit{Conf}(\mathcal{E}), X_b \in \mathit{Conf}(\mathcal{F})$ and X_a is terminated or $X_b = \emptyset\}$. Moreover, $(\{a\} \times X_a) \cup (\{b\} \times X_b)$ is terminated iff both X_a and X_b are terminated. ■

In this paper, apart from action refinement and sequential composition, we do not treat process algebraic operations formally; in particular we do not consider an operator for parallel composition. When considering such an operator, the following problem occurs. Flow event structures allow for an easy definition of CCS-like parallel composition [BC-b], which however does not always give the desired result. In [CZ] it is shown that the subclass of flow event structures satisfying the so-called Δ-axiom is well-behaved under parallel composition, and is closed under all operators of CCS as defined in [BC-b].

Axiom Δ: $\quad a\#b \prec c \wedge a \not\sim c \Rightarrow \exists d : b\#d \prec c \wedge \forall e\#d : (e \neq b \Rightarrow b\#e \sim c).$

Here $e \sim e'$ abbreviates $e\#e' \vee e \prec e' \vee e' \prec e$. So $\not\sim$ is defined on flow event structures exactly as *co* is on prime event structures. However, this class is not closed under our refinement operator.

Example 3.7

Refining b into $b_1|b_2$ in $\begin{tikzcd} a \arrow[r, phantom, "\#"] & b \arrow[r, phantom, "\#"] \arrow[d] & d \arrow[dl] \\ & c & \end{tikzcd}$ yields $\begin{tikzcd} a \arrow[r, phantom, "\#"] \arrow[d, phantom, "\#"] & b_1 \arrow[dl, phantom, "\#"] \arrow[ddl] & \\ b_2 \arrow[rr, phantom, "\#"] \arrow[d] & & d \arrow[dll] \\ c & & \end{tikzcd}$. The former structure satisfies Δ, by lack of events $e \neq b$ with $e\#d$. However, the latter does not: take $b := b_2$ and $e := b_1$; then $b_2\#b_1$fails to hold.[3]

[3] After renaming d into $\overline{a}$ and b into τ, the original event structure can be denoted by the CCS-expression $a\,|\,\overline{a}.c.\mathsf{nil}$. Hence, excluding this event structure from consideration by strengthening the Δ-axiom is not an option.

We claim that this problem can be solved by regarding a larger subclass of flow event structures, closed under action refinement and the CCS operators as defined in [BC-b], for which parallel composition is still well-behaved. This will be discussed in a forthcoming paper.

We end this section with a lemma that will be useful later on.

Lemma 3.1 Let $\mathcal{E} \in \mathbb{E}_{\mathit{flow}}$, $X \in \mathit{Conf}\,(\mathcal{E})$ and $\mathit{busy} \subseteq X$.

Then $\forall e \in \mathit{busy} : e$ maximal in X with respect to $<_X$
$\Leftrightarrow \forall Y \subseteq \mathit{busy} : X - Y \in \mathit{Conf}\,(\mathcal{E})$.

Proof
"$\Rightarrow$" Let $\mathcal{E} \in \mathbb{E}_{\mathit{flow}}$, $X \in \mathit{Conf}\,(\mathcal{E})$, $Y \subseteq X$ and $\forall e \in Y : e$ maximal in X w.r.t. $<_X$.
It suffices to prove that $X - Y \in \mathit{Conf}\,(\mathcal{E})$.
$X - Y$ is finite and conflict-free and does not contain causality cycles since X has these properties. It remains to be shown that $X - Y$ is left-closed up to conflicts.
Suppose $e \in X - Y$, $d \prec_{\mathcal{E}} e$ and $d \notin X - Y$. If $d \in Y$ then d would be maximal in X w.r.t. $<_X$, contradicting $d \prec_{\mathcal{E}} e$. Thus $d \notin X$. Hence there is an $f \in X$ with $f \prec_{\mathcal{E}} e$ and $d \#_{\mathcal{E}} f$. Since $f \prec_{\mathcal{E}} e$, f is not maximal in X w.r.t. $<_X$, so $f \in X - Y$, which had to be proved.

"$\Leftarrow$" Let $\mathcal{E} \in \mathbb{E}_{\mathit{flow}}$, $X \in \mathit{Conf}\,(\mathcal{E})$, $d \in X$ and $X - \{d\} \in \mathit{Conf}(\mathcal{E})$.
It suffices to proof that d is maximal w.r.t. $<_X$.
Suppose it is not, then $\exists e \in X$ with $d \prec_{\mathcal{E}} e$. Since $X - \{d\} \in \mathit{Conf}\,(\mathcal{E})$, there exists an $f \in X - \{d\}$ with $f \prec_{\mathcal{E}} e$ and $d \#_{\mathcal{E}} f$, contradicting the conflict-freeness of X. ■

This means that, in Proposition 3.3, the condition "$e \in \mathit{busy}(\widetilde{X}) \Rightarrow e$ maximal in X w.r.t. $<_X$" can be replaced by "for all $Y \subseteq \mathit{busy}(\widetilde{X})$, $X - Y \in \mathit{Conf}\,(\mathcal{E})$". The underlying idea is that, as events which are causally necessary for other events in X may only be replaced by terminated configurations, it must be possible to take any subset of "non-terminated" or busy events out of X, again obtaining a configuration.

4 Refinement of actions in stable and non-stable event structures

Stable and non-stable event structures are introduced in the work of Winskel [Winskel]. In this section, we extend the model by a predicate for successful termination. We define a compositional operator for general action refinement on event structures and show that the subclass of stable event structures is closed under this operator.

4.1 Event structures

Definition 4.1

A *(labelled) event structure (over an alphabet Act)* is a 5-tuple $\mathcal{E} = (E, Con, \vdash, \surd, l)$ where

- E is a set of *events*,
- Con is a non-empty set of finite subsets of E (the *consistency predicate*) satisfying $Y \subseteq X \in Con \Rightarrow Y \in Con$,
- $\vdash \subseteq Con \times E$ is the *enabling relation*, satisfying $X \vdash e \wedge X \subseteq Y \in Con \Rightarrow Y \vdash e$,
- $\surd \subseteq Con$ is the *termination predicate*, satisfying $e \notin X \in \surd \Rightarrow X \cup \{e\} \notin Con$,
- $l : E \to Act$ is the *labelling function*.

Here, E and l play the same rôle as before. $X \in Con$ means that the events in X could happen in the same run — they are *consistent*. $\vdash$ indicates *possible causes*: an event e can occur whenever for some X with $X \vdash e$ all events in X have occurred before. Finally, $X \in \surd$ means that the represented system may terminate successfully after performing exactly the set of events X.

The termination predicate is added to distinguish between deadlock and successful termination, which was not done in [Winskel]. There are now for instance two event structures with an empty set of events: $\varepsilon = (\emptyset, \emptyset, \emptyset, \{\emptyset\}, \emptyset)$ represents successful termination, whereas $\delta = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$ represents deadlock (successful termination and deadlock are represented by ε and δ, respectively, in the context of ACP [BW, Vrancken]). The requirement on the termination predicate in Definition 4.1 says that a set of events can terminate only if adding any further events would make the set inconsistent. As we will show, dropping this requirement would give rise to notions of sequential composition and action refinement that are either more difficult to define and analyse or conceptually different from the notions considered so far.

Let $\mathbb{E}$ denote the domain of event structures labelled over Act.

Two event structures $\mathcal{E}$ and $\mathcal{F}$ are *isomorphic* ($\mathcal{E} \cong \mathcal{F}$) iff there exists a bijection between their sets of events preserving Con, $\vdash$, $\surd$ and labelling.

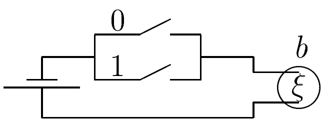
Definition 4.2 Let $\mathcal{E} \in \mathbb{E}$.

A subset $X \subseteq E$ is *secured* iff $\exists e_1, ..., e_n : X = \{e_1, ..., e_n\} \wedge \forall i < n : \{e_1, ..., e_i\} \vdash e_{i+1}$. X is a *(finite) configuration* of $\mathcal{E}$ iff X is secured and consistent. $Conf(\mathcal{E})$ denotes the set of all configurations of $\mathcal{E}$. A configuration X is called *terminated* iff $X \in \surd$.

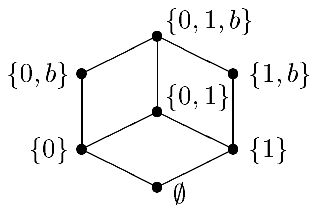
A collection of sets is called *directed* if for every two sets it contains, it also contains a superset of their union. In [Winskel] the family $\mathcal{F}(\mathcal{E})$ of (possibly infinite) configurations of an event structure $\mathcal{E}$ is defined. Under that definition (1.1.2 in [Winskel]), trivially, the infinite configurations are exactly the unions of directed collections of finite configurations. Moreover, our configurations coincide with the finite ones from [Winskel]. Hence the set $Conf(\mathcal{E})$ contains the same information as $\mathcal{F}(\mathcal{E})$. As pointed out in Section 2, the configurations in $Conf(\mathcal{E})$ represent the runs executable, and the states reachable, in a finite period of time.

Example 4.1 "Parallel switch" [Winskel].

We have two actions 0 and 1 interpreted as closing switch 0 and closing switch 1, respectively, in an electric circuit. As soon as at least one of the switches is closed, a bulb lights up; this is represented as an action b.



Assuming that each action occurs only once, this may be represented by an event structure with events 0, 1 and b, labelled with themselves, all sets of events consistent, $\vdash$ given by $\emptyset \vdash 0$, $\emptyset \vdash 1$, $\{0\} \vdash b$ and $\{1\} \vdash b$, and only $\{0, 1, b\}$ terminated[4]. The configurations are:



Note that the b-event may occur here without a unique "causal history"; in the configuration $\{0, 1, b\}$ it is not clear whether b is caused by 0 or by 1.

As observed above, the causal dependencies between the events in a configuration of an event structure can not always be faithfully represented by a partial order. Therefore, the subclass of *stable event structures* is defined in [Winskel]. In a configuration X of such a structure, causal dependence is given by a partial order $<_X$.

[4] One could argue that this process should be able to terminate successfully even if only one switch has been pressed. However, according to Definition 4.1, $\{0, b\}$ and $\{1, b\}$ cannot terminate; they are not maximal consistent sets of events.

Definition 4.3

An event structure $\mathcal{E}$ is *stable* iff $X \vdash e$ implies that there is a least subset Y of X with $Y \vdash e$.

Let $\mathcal{E} \in \mathbb{E}$ be stable and $X \in \mathit{Conf}(\mathcal{E})$. For $d, e \in X$, we write $d \prec_X e$ iff d is a member of the smallest subset Y of X with $Y \vdash e$. Now $<_X$, the *causality relation on* X, is defined as the transitive closure of $\prec_X$. We write $d \leq_X e$ for $d <_X e \vee d = e$.

We now observe how flow event structures (and therefore also prime event structures) can be seen as special cases of stable event structures.

Definition 4.4 Let $\mathcal{E}$ be a flow event structure.

Let $\mathit{Con}_{\mathcal{E}}$ be the class of conflict-free sets of events in $\mathcal{E}$ (cf. Definition 3.2) and $\surd_{\mathcal{E}}$ the class of terminated configurations of $\mathcal{E}$. We write $X \vdash_{\mathcal{E}} e$ iff X is consistent and contains a complete set of causes of e, i.e. iff $X \in \mathit{Con}_{\mathcal{E}}$ and $\forall d \in E_{\mathcal{E}}$: if $d \prec e$ and $d \notin X$ then there exists an $f \in X$ with $f \prec e$ and $d \# f$. Now $\mathcal{S}(\mathcal{E}) := (E_{\mathcal{E}}, \mathit{Con}_{\mathcal{E}}, \vdash_{\mathcal{E}}, \surd_{\mathcal{E}}, l_{\mathcal{E}})$ is the *stable event structure representation of* $\mathcal{E}$.

Proposition 4.1 Let $\mathcal{E}$ be a flow event structure.

Then $\mathcal{S}(\mathcal{E})$ is a stable event structure and $\mathit{Conf}(\mathcal{S}(\mathcal{E})) = \mathit{Conf}(\mathcal{E})$. Moreover, if X is a configuration of $\mathcal{E}$, then the causality relations $<_X$ on $\mathcal{E}$ and $\mathcal{S}(\mathcal{E})$ coincide, and X terminates in $\mathcal{E}$ iff it terminates in $\mathcal{S}(\mathcal{E})$.

Proof It follows immediately from the definitions that $\mathcal{S}(\mathcal{E})$ is a stable event structure.

"$\subseteq$": Let $X \in \mathit{Conf}(\mathcal{S}(\mathcal{E}))$. As X is secured, it must be finite and left-closed in $\mathcal{E}$ up to conflicts, and as it is consistent, it must be conflict-free in $\mathcal{E}$. It remains to show that X does not contain a causality cycle. Let $e_1, \ldots, e_n$ be a securing of X. It suffices to establish that $e_i \prec_{\mathcal{E}} e_j$ implies $i < j$. By contradiction, suppose $i \geq j$. As the set $\{e_1, \ldots, e_{j-1}\}$ enables e_j, it must contain a complete set of causes of e_j. Hence, there must be a $k < j$ with $e_i \# e_k$, contradicting the conflict-freeness of X.

"$\supseteq$": To prove $X \in \mathit{Conf}(\mathcal{E}) \Rightarrow X \in \mathit{Conf}(\mathcal{S}(\mathcal{E}))$ we use induction on $|X|$. So suppose $X \in \mathit{Conf}(\mathcal{E})$, and the implication has been established for smaller sets. X is conflict-free in $\mathcal{E}$ and hence consistent in $\mathcal{S}(\mathcal{E})$, so it remains to be shown that it is secured. As X is finite and without causality cycles, it must have an element e such that there is no $f \in X$ with $e \prec_{\mathcal{E}} f$. Now $X - \{e\} \in \mathit{Conf}(\mathcal{E})$, and by induction $X - \{e\} \in \mathit{Conf}(\mathcal{S}((\mathcal{E}))$. Let $e_1, \ldots, e_n$ be a securing of $X - \{e\}$. As X is left-closed up to conflicts, $X - \{e\}$ must contain a complete set of causes of e. Hence $e_1, \ldots, e_n, e$ is a securing of X.

The remaining statements are straightforward consequences of the definitions. ■

4.2 Refinement of actions

A refinement operator on event structures will be given by a function *ref* specifying for each action a an event structure $\mathit{ref}(a)$ which is to be substituted for a. Again we only consider non-forgetful refinements here, where forgetful refinements refine some action into an event structure with $\emptyset \in \surd$, or equivalently (by Definition 4.1) $\mathit{Con} = \surd = \{\emptyset\}$. Apart from this restriction, we may replace an action by any event structure. In particular, refinement into deadlocking processes is unproblematic.

Definition 4.5

(i) A function $\mathit{ref} : \mathit{Act} \to \mathbb{E}$ is called a *refinement function (for event structures)* iff $\forall a \in \mathit{Act} : \emptyset \notin \surd_{\mathit{ref}(a)}$.

(ii) Let $\mathcal{E} \in \mathbb{E}$ and let *ref* be a refinement function.
Then the *refinement of $\mathcal{E}$ by ref*, $\mathit{ref}(\mathcal{E})$, is the event structure defined by
– $E_{\mathit{ref}(\mathcal{E})} := \{(e, e') \mid e \in E_{\mathcal{E}}, e' \in E_{\mathit{ref}(l_{\mathcal{E}}(e))}\}$,
– $\widetilde{X} \in \mathit{Con}_{\mathit{ref}(\mathcal{E})}$ iff $\pi_1(\widetilde{X}) \in \mathit{Con}_{\mathcal{E}}$ and $\forall e \in \pi_1(\widetilde{X}) : \pi_2^e(\widetilde{X}) \in \mathit{Con}_{\mathit{ref}(l_{\mathcal{E}}(e))}$,
– $\widetilde{X} \vdash_{\mathit{ref}(\mathcal{E})} (e, e')$ iff $\mathit{ready}(\widetilde{X}) \vdash_{\mathcal{E}} e$ and $\pi_2^e(\widetilde{X}) \vdash_{\mathit{ref}(l_{\mathcal{E}}(e))} e'$,
– $\widetilde{X} \in \surd_{\mathit{ref}(\mathcal{E})}$ iff $\pi_1(\widetilde{X}) \in \surd_{\mathcal{E}}$ and $\forall e \in \pi_1(\widetilde{X}) : \pi_2^e(\widetilde{X}) \in \surd_{\mathit{ref}(l_{\mathcal{E}}(e))}$,
– $l_{\mathit{ref}(\mathcal{E})}(e, e') := l_{\mathit{ref}(l_{\mathcal{E}}(e))}(e')$,
with $\pi_1(\widetilde{X}) := \{e \mid \exists f : (e, f) \in \widetilde{X}\}$, $\pi_2^e(\widetilde{X}) := \{f \mid (e, f) \in \widetilde{X}\}$ and $\mathit{ready}(\widetilde{X}) := \{e \in \pi_1(\widetilde{X}) \mid \pi_2^e(\widetilde{X}) \in \surd_{\mathit{ref}(l_{\mathcal{E}}(e))}\}$ for $\widetilde{X} \subseteq E_{\mathit{ref}(\mathcal{E})}$.

As usual, we verify that $\mathit{ref}(\mathcal{E})$ is well-defined, even when isomorphic event structures are identified. In addition we check that stability of $\mathcal{E}$ and *ref* is preserved.

Proposition 4.2

(i) If $\mathcal{E} \in \mathbb{E}$ and *ref* is a refinement function then $\mathit{ref}(\mathcal{E})$ is an event structure indeed.

(ii) If $\mathcal{E} \in \mathbb{E}$ and *ref*, *ref'* are refinement functions with $\mathit{ref}(a) \cong \mathit{ref}'(a)$ for all $a \in \mathit{Act}$ then $\mathit{ref}(\mathcal{E}) \cong \mathit{ref}'(\mathcal{E})$.

(iii) If $\mathcal{E}, \mathcal{F} \in \mathbb{E}$, $\mathcal{E} \cong \mathcal{F}$, and *ref* is a refinement function then $\mathit{ref}(\mathcal{E}) \cong \mathit{ref}(\mathcal{F})$.

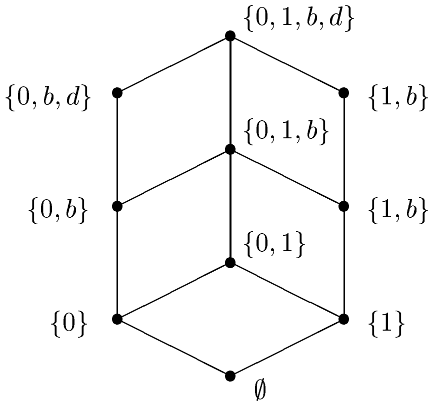
(iv) If $\mathcal{E}$ is stable and $\mathit{ref}(a)$ is stable for all $a \in \mathit{Act}$ then $\mathit{ref}(\mathcal{E})$ is stable.

Proof Straightforward. ■

As observed in the footnote to Example 4.1, it might be interesting to allow also non-maximal configurations to terminate, which however is forbidden in Definition 4.1. Now we investigate what happens if we just drop the corresponding requirement on the termination predicate in Definition 4.1.

Example 4.2

Let $\mathcal{E}$ be the parallel switch of Example 4.1, but with $\surd = \{\{0,b\},\{1,b\},\{0,1,b\}\}$, and let d be the event structure that performs a single action d and terminates. As in Section 3 (cf. Proposition 3.4), sequential composition of event structures with $\emptyset \notin \surd$ can be defined as a special case of action refinement. Now $\mathcal{E}; d$ has 4 events, which, preserving isomorphism, may be named 0, 1, b and d, labelled with themselves. The configurations of $\mathcal{E}; d$ are



This structure has a configuration $\{0,b,d\}$ with $\{0,b,d\} \vdash 1$. Here an action from the first component can happen after the second component of the sequential composition has already started, contradicting the notions of sequential composition and action refinement investigated here.

The problem raised in this example can be solved by a more elaborate definition of action refinement (and sequential composition) on event structures, involving duplication of events such as d above. However, this would undo the main advantage of working with event structures that are not prime. Alternatively, one can consider looser notions of sequential composition and action refinement, allowing some degree of parallelism between the components of a sequential composition (cf. [JPZ, JZ-a/b, Zwiers, Janssen, DGR, Wehrheim-a/b, RW, Huhn-a/b]).

As in Section 3, we establish that the behaviour of a refined event structure $\mathit{ref}(\mathcal{E})$ may be deduced compositionally from the behaviour of $\mathcal{E}$ and the behaviour of the refinements of actions. The following proposition generalises Proposition 3.3. However, as for non-stable event structures the notion $<_X$ is not defined, we use the alternative formulation suggested by Lemma 3.1.

Proposition 4.3 Let $\mathcal{E} \in \mathbb{E}$, let *ref* be a refinement function for event structures.

We call $\widetilde{X}$ a *refinement of configuration* $X \in \mathit{Conf}(\mathcal{E})$ *by ref* iff

– $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ where $\forall e \in X : X_e \in \mathit{Conf}(\mathit{ref}(l_{\mathcal{E}}(e))) - \{\emptyset\}$,

– $\forall Y \subseteq busy(\widetilde{X}) : X - Y \in Conf(\mathcal{E})$,
where $busy(\widetilde{X}) := \{e \in X \mid X_e \notin \surd_{ref(l_{\mathcal{E}}(e))}\}$.

Then $Conf\,(ref\,(\mathcal{E})) = \{\widetilde{X} \mid \widetilde{X}$ is a refinement of a configuration $X \in Conf\,(\mathcal{E})\}$. Moreover, $\widetilde{X}$ is terminated iff X is terminated and $\forall e \in X : X_e$ is terminated.

Proof "$\subseteq$": Let $\widetilde{X} \in Conf(ref(\mathcal{E}))$. We will show that

(i) $ready(\widetilde{X}) \subseteq Z \subseteq \pi_1(\widetilde{X}) \Rightarrow Z \in Conf(\mathcal{E})$ and

(ii) $e \in \pi_1(\widetilde{X}) \Rightarrow \pi_2^e(\widetilde{X}) \in Conf(ref(l_{\mathcal{E}}(e))) - \{\emptyset\}$.

Considering that $busy(\widetilde{X}) = \pi_1(\widetilde{X}) - ready(\widetilde{X})$ and that $\widetilde{X} = \bigcup_{e \in \pi_1(\widetilde{X})}\{e\} \times \pi_2^e(\widetilde{X})$, it then immediately follows that $\widetilde{X}$ is a refinement of $\pi_1(\widetilde{X}) \in Conf(\mathcal{E})$.

(ad (i)). Let $ready(\widetilde{X}) \subseteq Z \subseteq \pi_1(\widetilde{X})$. By Definition 4.5 we have $\pi_1(\widetilde{X}) \in Con_{\mathcal{E}}$, so by Definition 4.1 also Z is consistent. It remains to be shown that Z is secured. Let $\widetilde{X} = \{(e_1, e'_1), \ldots, (e_n, e'_n)\}$ such that $\forall i < n : \{(e_1, e'_1), \ldots, (e_i, e'_i)\} \vdash_{ref(\mathcal{E})} (e_{i+1}, e'_{i+1})$. We show that the sequence $d_1, ..., d_k$ obtained from $e_1, ..., e_n$ by deleting the events that are not in Z has the required properties. As $Z \subseteq \pi_1(\widetilde{X}) = \{e_1, ..., e_n\}$ we have $Z = \{d_1, ..., d_k\}$. Let $j < k$. Then $d_{j+1} = e_{i+1}$ for certain $i < n$. We have $\{(e_1, e'_1), \ldots, (e_i, e'_i)\} \vdash_{ref(\mathcal{E})} (e_{i+1}, e'_{i+1})$. Hence, by Definition 4.5, $ready(\{(e_1, e'_1), \ldots, (e_i, e'_i)\}) \vdash_{\mathcal{E}} e_{i+1} = d_{j+1}$. As $ready(\{(e_1, e'_1), \ldots, (e_i, e'_i)\}) \subseteq \{e_1, ..., e_i\}$ and $ready(\{(e_1, e'_1), \ldots, (e_i, e'_i)\}) \subseteq ready(\widetilde{X}) \subseteq Z$, we have $ready(\{(e_1, e'_1), \ldots, (e_i, e'_i)\}) \subseteq \{d_1, ..., d_j\} \in Con_{\mathcal{E}}$, so, by Definition 4.1, $\{d_1, ..., d_j\} \vdash d_{j+1}$. It follows that $Z \in Conf(\mathcal{E})$.

(ad (ii)). Let $e \in \pi_1(\widetilde{X})$. Then $\pi_2^e(\widetilde{X}) \neq \emptyset$, and, by Definition 4.5, $\pi_2^e(\widetilde{X})$ is consistent. It remains to be shown that $\pi_2^e(\widetilde{X})$ is secured. Let $\widetilde{X} = \{(e_1, e'_1), \ldots, (e_n, e'_n)\}$ such that $\forall i < n : \{(e_1, e'_1), \ldots, (e_i, e'_i)\} \vdash_{ref(\mathcal{E})} (e_{i+1}, e'_{i+1})$. We show that the sequence $d'_1, ..., d'_k$ obtained from $e'_1, ..., e'_n$ by deleting the events e'_i for which $e_i \neq e$ has the required properties. By definition $\pi_2^e(\widetilde{X}) = \{d'_1, ..., d'_k\}$. Let $j < k$. Then $d'_{j+1} = e'_{i+1}$ for certain $i < n$. We have $\{(e_1, e'_1), \ldots, (e_i, e'_i)\} \vdash_{ref(\mathcal{E})} (e_{i+1}, e'_{i+1})$. Hence, by Definition 4.5, $\{d'_1, ..., d'_j\} = \pi_2^e(\{(e_1, e'_1), \ldots, (e_i, e'_i)\}) \vdash_{ref(l_{\mathcal{E}}(e))} e'_{i+1} = d'_{j+1}$.

"$\supseteq$": Let $\widetilde{X}$ be a refinement of a configuration $X \in Conf(\mathcal{E})$. Then $\pi_1(\widetilde{X}) = X \in Con_{\mathcal{E}}$ and $\widetilde{X} = \bigcup_{e \in X}\{e\} \times X_e$ with $\forall e \in X : X_e \in Conf\,(ref\,(l_{\mathcal{E}}(e)))$. For all $e \in \pi_1(\widetilde{X})$ we have $\pi_2^e(\widetilde{X}) = X_e \in Con_{ref(l_{\mathcal{E}}(e))}$. Hence $\widetilde{X} \in Con_{ref(\mathcal{E})}$. It remains to be shown that $\widetilde{X}$ is secured. Using that $ready(\widetilde{X}) \in Conf(\mathcal{E})$, write $ready(\widetilde{X}) = \{e_1, ..., e_m\}$ such that $\forall i < m : \{e_1, ..., e_i\} \vdash_{\mathcal{E}} e_{i+1}$. For all $e \in busy(\widetilde{X})$ we have $ready(\widetilde{X}) \cup \{e\} \in Conf(\mathcal{E})$, so it must be that $ready(\widetilde{X}) \vdash e$. Let $busy(\widetilde{X}) = \{e_{m+1}, ..., e_n\}$, i.e. $X = \{e_1, ..., e_n\}$, with $n \geq m$. For $i = 1, ..., n$ let $X_{e_i} = \{e'_{i1}, ..., e'_{ik_i}\}$ such that $\forall j < k_i : \{e'_{i1}, ..., e'_{ij}\} \vdash_{ref(l_{\mathcal{E}}(e))} e'_{i(j+1)}$. We claim that the sequence $(e_1,e'_{11}), ..., (e_1,e'_{1k_1}), (e_2,e'_{21}), ..., (e_2,e'_{2k_2}), \ldots, (e_n,e'_{n1}), ..., (e_n,e'_{nk_n})$ secures $\widetilde{X}$. This follows from Definition 4.5. Namely, let $i < n$ and $j < k_i$. Then $ready(\{(e_1,e'_{11}), ..., (e_i, e'_{ik_i}), (e_{i+1},e'_{(i+1)1}), ..., (e_{i+1},e'_{(i+1)j})\}) = \{e_1, ..., e_{max\{i,m\}}\} \vdash e_{i+1}$ and $\pi_2^{e_{j+1}}(\{(e_1,e'_{11}), ..., (e_i, e'_{ik_i}), (e_{i+1},e'_{(i+1)1}), ..., (e_{i+1},e'_{(i+1)j})\}) = \{e'_{(i+1)1}, ..., e'_{(i+1)j}\} \vdash e'_{j+1}$.

The last statement of Proposition 4.3 follows immediately from Definition 4.5. ■

The stable event structure associated to a refined flow event structure $\mathcal{E}$ is not always equal to the corresponding refinement of the stable event structure associated to $\mathcal{E}$, i.e. there are flow event structures $\mathcal{E}$ and refinement functions *ref* with $\mathcal{S}(ref(\mathcal{E})) \neq ref_{\mathcal{S}}(\mathcal{S}(\mathcal{E}))$ where $ref_{\mathcal{S}}(a) := \mathcal{S}(ref(a))$ for all $a \in Act$.

Example 4.3

Take $\mathcal{E} := a \overset{\#}{\longrightarrow} b$, $ref(a) := a$ and $ref(b) := c \longrightarrow d$, where the names of events are equal to their labels. Then

$$ref(\mathcal{E}) = \begin{tikzcd} a \arrow[r, "\#"] \arrow[dr] \arrow[dr, phantom, "\#"'] & c \arrow[d] \\ & d \end{tikzcd}$$

and in $\mathcal{S}(ref(\mathcal{E}))$ we have $\{(a,a)\} \vdash_{\mathcal{S}(ref(\mathcal{E}))} (b,d)$. On the other hand, in $ref_{\mathcal{S}}(\mathcal{S}(\mathcal{E}))$ we have $\{(a,a)\} \nvdash_{ref_{\mathcal{S}}(\mathcal{S}(\mathcal{E}))} (b,d)$, as $\emptyset \nvdash_{\mathcal{S}(ref(b))} d$ in $\mathcal{S}(ref(b))$.

However, the difference between $\mathcal{S}(ref(\mathcal{E}))$ and $ref_{\mathcal{S}}(\mathcal{S}(\mathcal{E}))$ is not essential; see Corollary 5.1.

5 Configuration structures and action refinement

In the previous sections we have shown that different forms of event structures may be used for refinement of actions. Although the refinement operators we have defined depend on the particular "syntax" of the chosen event structures, in all cases the relevant behaviour of an event structure is determined by its set of configurations, a predicate on configurations indicating successful termination, and the labelling of events. As we will see in Part II, all relevant equivalence relations for event structures can be formulated in terms of these notions only.

In this section, we define a refinement operator for a general and more abstract event oriented model of concurrent systems, in which a system is represented by its set of configurations, a termination predicate and a labelling function. To the extent that refinement operators for any particular brand of event structures are determined by a compositionality property such as expressed by Propositions 2.2, 3.3 and 4.3, these operators are obtained as special cases.

5.1 Configuration structures

Definition 5.1

A *(labelled) configuration structure (over an alphabet Act)* is a triple $\mathcal{C} = (C, \surd, l)$ where C is a family of finite sets (the *configurations*), $\surd \subseteq C$ a *termination predicate*, satisfying $X \in \surd \wedge X \subseteq Y \in C_{\mathcal{C}} \Rightarrow X = Y$ (i.e. terminating configurations must be maximal), and $l : \bigcup_{X \in C} X \to \mathit{Act}$ is a *labelling function.*

In [GG-c] we introduced configuration structures without the termination predicate $\surd$, and required them to satisfy three closure properties, ensuring that they were exactly the labelled versions of the families of finite configurations of Winskel's (non-stable) event structures. Following [GP] we drop these closure properties here, thereby obtaining a simpler and more general model of concurrency, closely resembling the *Chu spaces* of [Pratt-b]. In addition, we add the termination predicate, enabling us to distinguish between deadlock and successful termination. In Section 5.3 we will show that our refinement operator preserves the closure properties of [GG-c], as well as an extra property restricting attention to the families of finite configurations of stable event structures.

The requirement on the termination predicate in Definition 5.1 says that in a successfully terminating configuration no further events can be executed. As in the previous section, this requirement enables a clear notion of action refinement.

Let $\mathbb{C}$ denote the domain of configuration structures labelled over *Act*. The set $E_{\mathcal{C}}$ of *events* of $\mathcal{C} \in \mathbb{C}$ is defined by $E_{\mathcal{C}} := \bigcup_{X \in C_{\mathcal{C}}} X$.

As usual, we will not distinguish configuration structures which are isomorphic in the sense that they only differ with respect to names of events.

Definition 5.2

Two configuration structures $\mathcal{C}$ and $\mathcal{D}$ are *isomorphic* ($\mathcal{C} \cong \mathcal{D}$) iff there exists a bijective mapping $f : E_{\mathcal{C}} \to E_{\mathcal{D}}$ such that
– $X \in C_{\mathcal{C}} \Leftrightarrow f(X) \in C_{\mathcal{D}}$ for $X \subseteq E_{\mathcal{C}}$,
– $X \in \surd_{\mathcal{C}} \Leftrightarrow f(X) \in \surd_{\mathcal{D}}$ for $X \subseteq E_{\mathcal{C}}$,
– and $l_{\mathcal{C}}(e) = l_{\mathcal{D}}(f(e))$ for $e \in E_{\mathcal{C}}$.

We may now associate a configuration structure with each event structure.

Definition 5.3 Let $\mathcal{E}$ be a prime, flow or stable event structure.

The configuration structure of $\mathcal{E}$, $\mathcal{C}(\mathcal{E})$, is defined as

$$\mathcal{C}(\mathcal{E}) := (\mathit{Conf}(\mathcal{E}),\ \surd(\mathcal{E}),\ l_{\mathcal{E}} \restriction \bigcup_{X \in \mathit{Conf}(\mathcal{E})} X).$$

where $\surd(\mathcal{E})$ denotes the set of terminated configurations of $\mathcal{E}$.

There is no unique correspondence in general: different event structures may have the same configuration structure.[5] It follows immediately from Proposition 4.1 that, for a flow event structure $\mathcal{E}$, $\mathcal{C}(\mathcal{S}(\mathcal{E})) = \mathcal{C}(\mathcal{E})$.

5.2 Refinement of actions

Next, we define refinement of actions for configuration structures. Again we only consider non-forgetful refinements here, hence $\mathit{ref}\ (a) \neq \varepsilon$ for all $a \in \mathit{Act}$, where ε denotes the configuration structure with $C_\varepsilon = \surd_\varepsilon = \{\emptyset\}$, indicating a successfully terminating process that performs no actions.

Definition 5.4

(i) A function $\mathit{ref} : \mathit{Act} \to \mathbb{C} - \{\varepsilon\}$ is called a *refinement function (for configuration structures)*.

(ii) Let $\mathcal{C}$ be a configuration structure and let ref be a refinement function.
We call $\widetilde{X}$ a *refinement of a configuration* $X \in C_{\mathcal{C}}$ *by* ref iff

– $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ where $\forall e \in X : X_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))} - \{\emptyset\}$,

– $\forall Y \subseteq \mathit{busy}\ (\widetilde{X}) :\ X - Y \in C_{\mathcal{C}}$, where $\mathit{busy}\ (\widetilde{X}) := \{e \in X \mid X_e \notin \surd_{\mathit{ref}(l_{\mathcal{C}}(e))}\}$.

[5]Although for prime event structures the correspondence is unique [NPW].

Such a refinement is *terminated* iff $\forall e \in X : X_e \in \surd_{ref(l_{\mathcal{C}}(e))}$, i.e. iff $busy(\widetilde{X}) = \emptyset$.
The *refinement of $\mathcal{C}$ by ref* is defined as $ref(\mathcal{C}) = (C_{ref(\mathcal{C})}, \surd_{ref(\mathcal{C})}, l_{ref(\mathcal{C})})$ with

- $C_{ref(\mathcal{C})} := \{\widetilde{X} \mid \widetilde{X}$ is a refinement of some $X \in C_{\mathcal{C}}$ by $ref\}$,
- $\surd_{ref(\mathcal{C})} := \{\widetilde{X} \mid \widetilde{X}$ is a terminated refinement of some $X \in \surd_{\mathcal{C}}$ by $ref\}$
- and $l_{ref(\mathcal{C})}(e, e') := l_{ref(l_{\mathcal{C}}(e)}(e')$ for all $(e, e') \in E_{ref(\mathcal{C})}$.

This definition corresponds exactly to the previous characterisations of refinement for event structures (cf. Propositions 2.2 and 3.3, Lemma 3.1 and Proposition 4.3).

Next we show that refinement is a well-defined operation on configuration structures, even when isomorphic configuration structures are identified.

Proposition 5.1

(i) If $\mathcal{C} \in \mathbb{C}$ and *ref* is a refinement function then also $ref(\mathcal{C})$ is a configuration structure.

(ii) If $\mathcal{C} \in \mathbb{C}$ and *ref*, *ref'* are refinement functions with $ref(a) \cong ref'(a)$ for all $a \in Act$ then $ref(\mathcal{C}) \cong ref'(\mathcal{C})$.

(iii) If $\mathcal{C}, \mathcal{D} \in \mathbb{C}$, *ref* is a refinement function and $\mathcal{C} \cong \mathcal{D}$ then $ref(\mathcal{C}) \cong ref(\mathcal{D})$.

Proof Straightforward. ■

Finally, we show that the more structural refinement operators for prime, flow and stable event structures defined in Sections 2, 3 and 4 are consistent with the refinement operator for configuration structures proposed here.

Theorem 5.1

Let $\mathcal{E}$ be a prime, flow or stable event structure and let *ref* be a refinement function for such event structures.

Then $\mathcal{C}(ref(\mathcal{E})) = ref_{\mathcal{C}}(\mathcal{C}(\mathcal{E}))$, where $ref_{\mathcal{C}}(a) := \mathcal{C}(ref(a))$ for all $a \in Act$.

Proof By Definition 5.3 we have $C_{\mathcal{C}(\mathcal{E})} = Conf(\mathcal{E})$, $C_{ref_{\mathcal{C}}(l_{\mathcal{C}(\mathcal{E})}(e))} = Conf(ref(l_{\mathcal{E}}(e)))$ and $\surd_{ref_{\mathcal{C}}(l_{\mathcal{C}(\mathcal{E})}(e))} = \surd(ref(l_{\mathcal{E}}(e)))$. Hence a refinement of a configuration $X \in C_{\mathcal{C}(\mathcal{E})} = Conf(\mathcal{E})$ by *ref* (as defined in Propositions 2.2, 3.3 resp. 4.3) is the same as a refinement of X by $ref_{\mathcal{C}}$ (as in Definition 5.4); in the case of flow (and prime) event structures Lemma 3.1 is used here. It follows that

- $C_{\mathcal{C}(ref(\mathcal{E}))} \overset{5.3}{=} Conf(ref(\mathcal{E})) \overset{\substack{2.2\\3.3\\4.3}}{=} \{\widetilde{X} \mid \widetilde{X}$ is a refinement of some $X \in Conf(\mathcal{E})$ by $ref\}$ $\overset{\text{above}}{=} \{\widetilde{X} | \widetilde{X}$ is a refinement of some $X \in C_{\mathcal{C}(\mathcal{E})}$ by $ref_{\mathcal{C}}\} \overset{5.4}{=} C_{ref_{\mathcal{C}}(\mathcal{C}(\mathcal{E}))}$,

- $\surd_{\mathcal{C}(ref(\mathcal{E}))} \stackrel{5.3}{=} \surd(ref(\mathcal{E})) \stackrel{\substack{2.2\\3.3\\4.3}}{=} \{\widetilde{X} \mid \widetilde{X}$ is a ref. of some $X \in \surd(\mathcal{E})$ with $busy(\widetilde{X}) = \emptyset\}$ $\stackrel{\text{above}}{=} \{\widetilde{X} | \widetilde{X}$ is a terminated refinement of some $X \in \surd_{\mathcal{C}(\mathcal{E})}$ by $ref_{\mathcal{C}}\} \stackrel{5.4}{=} \surd_{ref_{\mathcal{C}}(\mathcal{C}(\mathcal{E}))}$,
- $l_{\mathcal{C}(ref(\mathcal{E}))}(e, e') \stackrel{5.3}{=} l_{ref(\mathcal{E})}(e, e') \stackrel{\substack{2.4\\3.5\\4.5}}{=} l_{ref(l_{\mathcal{E}}(e))}(e') \stackrel{5.3}{=} l_{ref_{\mathcal{C}}(l_{\mathcal{C}(\mathcal{E})}(e))}(e') \stackrel{5.4}{=} l_{ref_{\mathcal{C}}(\mathcal{C}(\mathcal{E}))}(e, e')$. ■

Thus, first performing action refinement on an event structure and then translating the result to a configuration structure yields the same result as first moving to configuration structures and then performing action refinement. In contrast, Example 4.3 showed that performing action refinement and transforming flow into stable event structures does not always commute, i.e. we do not always have $\mathcal{S}(ref(\mathcal{E})) = ref_{\mathcal{S}}(\mathcal{S}(\mathcal{E}))$. However, interpreting both sides as configuration structures always yields the same result.

Corollary 5.1 Let $\mathcal{E}$ be a flow event structure and *ref* a refinement function.

Then $\mathcal{C}(\mathcal{S}(ref(\mathcal{E}))) = \mathcal{C}(ref_{\mathcal{S}}(\mathcal{S}(\mathcal{E})))$.

Proof As remarked at the end of Section 5.1, we always have $\mathcal{C}(\mathcal{S}(\mathcal{E})) = \mathcal{C}(\mathcal{E})$. Hence $\mathcal{C}(ref_{\mathcal{S}}(\mathcal{S}(\mathcal{E}))) = (ref_{\mathcal{S}})_{\mathcal{C}}(\mathcal{C}(\mathcal{S}(\mathcal{E}))) = ref_{\mathcal{C}}(\mathcal{C}(\mathcal{E})) = \mathcal{C}(ref(\mathcal{E})) = \mathcal{C}(\mathcal{S}(ref(\mathcal{E})))$, using Theorem 5.1. ■

The following lemma says that if in Definition 5.4 we would not have insisted that the sets X_e are nonempty, the resulting notion of $ref(\mathcal{C})$ would be the same. Note however, that in this case X need longer be given as $\{e \in E_{\mathcal{C}} \mid \exists e' : (e, e') \in \widetilde{X}\}$, and the notion $busy(\widetilde{X})$ ought to be parametrised by X.

Lemma 5.1 Let $\mathcal{C}$ be a configuration structure and let *ref* be a refinement function.

Then $\widetilde{X} \in C_{ref(\mathcal{C})}$ iff there is a configuration $X \in C_{\mathcal{C}}$ such that

- $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ where $\forall e \in X : X_e \in C_{ref(l_{\mathcal{C}}(e))}$,
- $\forall Y \subseteq busy_X(\widetilde{X}) : \ X - Y \in C_{\mathcal{C}}$, where $busy_X(\widetilde{X}) := \{e \in X \mid X_e \notin \surd_{ref(l_{\mathcal{C}}(e))}\}$.

Proof "Only if" is given by Definition 5.4, but "if" is not, because we do not require here that $X_e \neq \emptyset$. So suppose $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ where $\forall e \in X : X_e \in C_{ref(l_{\mathcal{C}}(e))}$, and $\forall Y \subseteq busy_X(\widetilde{X}) : \ X - Y \in C_{\mathcal{C}}$. Let $X' := \{e \in X \mid X_e \neq \emptyset\}$. Then $\widetilde{X} = \bigcup_{e \in X'} \{e\} \times X_e$ where $\forall e \in X' : X_e \in C_{ref(l_{\mathcal{C}}(e))} - \{\emptyset\}$. Let $Y \subseteq busy(\widetilde{X}) = busy_{X'}(\widetilde{X}) \subseteq busy_X(\widetilde{X})$. As *ref* is not forgetful, we have $X - X' \subseteq busy_X(\widetilde{X})$. Hence, $\forall Y \subseteq busy_{X'}(\widetilde{X}) : \ X' - Y = X - (Y \cup (X - X')) \in C_{\mathcal{C}}$. It follows that $\widetilde{X} \in C_{ref(\mathcal{C})}$. ■

5.3 Stable configuration structures

In this section we characterise the configuration structures with the property that the causal dependencies in configurations can faithfully be represented by means of partial orders. It turns out that these are exactly the configuration structures associated to stable event structures; hence we call them *stable configuration structures.* We show that this class is closed under action refinement.

Definition 5.5 A configuration structure $\mathcal{C} \in \mathbb{C}$ is

- *rooted* iff $\emptyset \in C_{\mathcal{C}}$,
- *connected* iff $\emptyset \neq X \in C_{\mathcal{C}} \Rightarrow \exists e \in X : X - \{e\} \in C_{\mathcal{C}}$,
- *closed under bounded unions* iff $X, Y, Z \in C_{\mathcal{C}},\ X \cup Y \subseteq Z \Rightarrow X \cup Y \in C_{\mathcal{C}}$,
- *closed under bounded intersections* iff $X, Y, Z \in C_{\mathcal{C}},\ X \cup Y \subseteq Z \Rightarrow X \cap Y \in C_{\mathcal{C}}$.

$\mathcal{C}$ is *stable* iff it is rooted, connected and closed under bounded unions and intersections. $\mathbb{C}_{stable}$ denotes the domain of stable configuration structures.

The following lemma shows a number of properties that are equivalent to connectedness. The second of these says that a configuration structure is connected iff every configuration is reachable from the empty configuration (modelling the initial state of the represented system) by executing one event at a time. It also implies that in a connected configuration structure rootedness is equivalent to nonemptyness. Properties 3 and 5 will be used further on.

Lemma 5.2 Let $\mathcal{C} \in \mathbb{C}$ be closed under bounded unions.

The following are equivalent:

1. $\mathcal{C}$ is connected.
2. For all $X \in C_{\mathcal{C}}$ there are $X_0, ..., X_n \in C_{\mathcal{C}}$ with $\emptyset = X_0 \subset X_1 \subset \cdots \subset X_n = X$ and $\forall i < n : |X_{i+1} - X_i| = 1$.
3. $\mathcal{C}$ satisfies the property of *coincidence-freeness*:

   $$X \in C \wedge d, e \in X,\ d \neq e \Rightarrow \exists Y \in C \text{ with } Y \subseteq X \text{ and } (d \in Y \Leftrightarrow e \notin Y),$$

4. If $X, Y \in C_{\mathcal{C}}$, $X \subseteq Y$ and $|Y - X| \geq 2$, then $\exists Z \in C_{\mathcal{C}}$ with $X \subset Z \subset Y$ (i.e. $X \neq Z \neq Y$).
5. If $X, Y \in C_{\mathcal{C}}$ and $X \subseteq Y$, then there are $X_0, ..., X_n \in C_{\mathcal{C}}$ with $X = X_0 \subset X_1 \subset \cdots \subset X_n = Y$ and $\forall i < n : |X_{i+1} - X_i| = 1$.

Proof

$1 \Rightarrow 2$ By repeated application of connectedness.

$2 \Rightarrow 3$ Trivial.

$3 \Rightarrow 4$ Take $d, e \in Y - X$, $d \neq e$. Then $\exists Z \in C_{\mathcal{C}}$, $Z \subseteq Y$ with $(d \in Z \Leftrightarrow e \notin Z)$, say $d \in Z$ and $e \notin Z$. As $\mathcal{C}$ is closed under bounded unions, $X \cup Z \in C_{\mathcal{C}}$. Moreover $X \subset X \cup Z \subset Y$, $d \in (X \cup Z) - X$ and $e \in Y - (X \cup Z)$.

$4 \Rightarrow 5$ By repeated application.

$5 \Rightarrow 1$ Trivial. ■

Next we show that in stable configuration structures causality can be faithfully represented by means of partial orders.

Definition 5.6 Let $\mathcal{C} \in \mathbb{C}_{stable}$ and $X \in C_{\mathcal{C}}$.

The *causality relation on* X is given by $d <_X e$ iff $d \leq_X e$ and $d \neq e$, where $d \leq_X e$ iff for all $Y \in C_{\mathcal{C}}$ with $Y \subseteq X$ we have $e \in Y \Rightarrow d \in Y$. The *concurrency relation on* X is given by $d\ co_X\ e$ iff $\neg(d <_X e \vee e <_X d)$. Finally, $pomset(X) := [(X, <_X, l_{\mathcal{C}} \restriction X)]_{\cong}$.

Proposition 5.2 Let $\mathcal{C} \in \mathbb{C}_{stable}$ and $X \in C_{\mathcal{C}}$.

$\leq_X$ is a partial order.

Proof The transitivity and reflexivity of $\leq_X$ follow immediately from Definition 5.6, and antisymmetry is a restatement of coincidence-freeness (cf. Lemma 5.2). ■

Proposition 5.3 Let $\mathcal{C} \in \mathbb{C}_{stable}$ and $X \in C_{\mathcal{C}}$.

A set $Y \subseteq X$ is a subconfiguration of X, i.e. $Y \in C_{\mathcal{C}}$, iff Y is left-closed w.r.t. $<_X$.

Proof That subconfigurations of X are left-closed w.r.t. $<_X$ follows immediately from Definition 5.6. Conversely, suppose $Y \subseteq X$ is left-closed w.r.t. $<_X$. In case $Y = X$, trivially $Y \in C_{\mathcal{C}}$, and in case $Y = \emptyset$ this follows from the rootedness of $\mathcal{C}$. Hence we may assume that $X - Y \neq \emptyset$ and $Y \neq \emptyset$. For every pair of events d, e with $d \in Y$ and $e \in X - Y$ we have $\neg(e <_X d)$, i.e. there is a $Z_{de} \in C_{\mathcal{C}}$ with $Z_{de} \subseteq X$, $d \in Z_{de}$ and $e \notin Z_{de}$. We have $Y = \bigcup_{d \in Y} \bigcap_{e \in X-Y} Z_{de}$. As $X - Y \neq \emptyset$ and $Y \neq \emptyset$, these unions and intersections are non-empty; as $Z_{de} \subseteq X$ they are bounded; and they are finite because X is finite. Hence, using that $\mathcal{C}$ is closed under (binary) bounded unions and intersections, we find that $Y \in C_{\mathcal{C}}$. ■

Proposition 5.4 Let $\mathcal{C} \in \mathbb{C}_{stable}$ and $X, Y \in C_{\mathcal{C}}$ with $Y \subseteq X$.

Then $<_Y = <_X \restriction Y$.

Proof This follows immediately from Definition 5.6, since for any $Z \in C_{\mathcal{C}}$, $Z \subseteq X$, also $Z \cap Y \in C_{\mathcal{C}}$ by stability. ■

We now establish that, whenever in a nonempty configuration structure causality can be faithfully represented by means of partial orders, it must be stable.

Theorem 5.2 Let $\mathcal{C} \in \mathbb{C}$ be rooted.

For all $X \in C_{\mathcal{C}}$ let $<_X$ be defined as in Definition 5.6. Then Propositions 5.2 and 5.3 are satisfied iff $\mathcal{C}$ is stable.

Proof "If" has already been established, so suppose that $<_X$ satisfies Propositions 5.2 and 5.3 for all $X \in C_{\mathcal{C}}$. Let $X, Y, Z \in C_{\mathcal{C}}$ and $X \cup Y \subseteq Z$. As X and Y are subconfigurations of Z, they must be left-closed w.r.t. $<_Z$. Therefore $X \cup Y$ and $X \cap Y$ are left-closed w.r.t. $<_Z$, and hence must be subconfigurations of Z. Thus $\mathcal{C}$ is closed under bounded unions and intersections. As $\leq_X$ is antisymmetric for all $X \in C_{\mathcal{C}}$, $\mathcal{C}$ is coincidence-free, and hence, by Lemma 5.2, connected. ■

The following theorem characterises the classes of configuration structures that arise as the sets of configurations of (stable) event structures.

Theorem 5.3 Let $\mathcal{C} \in \mathbb{C}$.

$\mathcal{C}$ is of the form $\mathcal{C}(\mathcal{E})$ with $\mathcal{E} \in \mathbb{E}$ iff it is rooted, connected and closed under bounded unions; it is of the form $\mathcal{C}(\mathcal{E})$ with $\mathcal{E} \in \mathbb{E}$ stable iff $\mathcal{C}$ is stable.

Proof This is a reformulation of Theorems 1.1.9, 1.1.13 and 1.1.16 of [Winskel] (except that in [Winskel] the requirement of rootedness, or, equivalently, nonemptyness of $C_{\mathcal{C}}$, is missing), taking into account that our configurations are finite, whereas the ones in [Winskel] may be infinite. Cf. [GP] for how the properties above are obtained from the ones in [Winskel]. ■

Next, we verify that the causality relations $<_X$ on the configurations X of stable configuration structures agree with the ones defined earlier for stable event structures.

Proposition 5.5 Let $\mathcal{E} \in \mathbb{E}$ be stable and $X \in Conf(\mathcal{E})$.

Then the causality relations $<_X$ defined on $\mathcal{E}$ and $\mathcal{C}(\mathcal{E})$ coincide.

Proof For any given $e \in X$ let $\downarrow e := \{d \in X \mid d \leq_X e$ according to Definition 4.3$\}$. Using Definition 4.2 it follows that $\downarrow e \in Conf(\mathcal{E})$. Thus whenever $d \leq_X e$ according to Definition 5.6, it must be that $d \in \downarrow e$, and hence $d \leq_X e$ according to Definition 4.3.

Now suppose that $d \leq_X e$ according to Definition 4.3. If follows immediately from Definition 4.2 that every subconfiguration of X that contains e must also contain d. Hence $d \leq_X e$ according to Definition 5.6. ■

We now show that each of the properties of Definition 5.5 is preserved under action refinement; in particular the class of stable configuration structures is closed under refinement.

Proposition 5.6 Let $\mathcal{C} \in \mathbb{C}$ and let *ref* be a refinement function.

(i) If $\mathcal{C}$ is rooted, then so is $\mathit{ref}(\mathcal{C})$.

(ii) If $\mathcal{C}$ and all $\mathit{ref}(a)$ for $a \in \mathit{Act}$ are connected, then so is $\mathit{ref}(\mathcal{C})$.

(iii) If $\mathcal{C}$ and all $\mathit{ref}(a)$ for $a \in \mathit{Act}$ are closed under bounded unions, then so is $\mathit{ref}(\mathcal{C})$.

(iv) If $\mathcal{C}$ and all $\mathit{ref}(a)$, $a \in \mathit{Act}$ are closed under bounded intersections, then so is $\mathit{ref}(\mathcal{C})$.

Proof

(i) Trivial.

(ii) Let $\emptyset \neq \widetilde{X} \in C_{\mathit{ref}(\mathcal{C})}$. Then $\widetilde{X} = \bigcup_{e \in X}\{e\} \times X_e$ with $\forall e \in X : X_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))} - \{\emptyset\}$ and $\forall Y \subseteq \mathit{busy}(\widetilde{X}) : X - Y \in C_{\mathcal{C}}$, where $\mathit{busy}\ (\widetilde{X}) = \{e \in X \mid X_e \notin \surd_{\mathit{ref}(l_{\mathcal{C}}(e))}\}$. In case $\mathit{busy}(\widetilde{X}) \neq \emptyset$ take $e \in \mathit{busy}(\widetilde{X})$. As $\emptyset \neq X_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))}$ and $\mathit{ref}(l_{\mathcal{C}}(e))$ is connected, there must be an $e' \in X_e$ with $X_e - \{e'\} \in C_{\mathit{ref}(l_{\mathcal{C}}(e))}$. Using Lemma 5.1 it now follows that $\widetilde{X} - \{(e, e')\} \in C_{\mathit{ref}(\mathcal{C})}$.

Next assume $\mathit{busy}(\widetilde{X}) = \emptyset$. As $\mathcal{C}$ is connected and $\emptyset \neq X \in C_{\mathcal{C}}$, there must be an $e \in X$ with $X - \{e\} \in C_{\mathcal{C}}$. As also $\mathit{ref}(l_{\mathcal{C}}(e))$ is connected and $\emptyset \neq X_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))}$, there must be an $e' \in X_e$ with $X_e - \{e'\} \in C_{\mathit{ref}(l_{\mathcal{C}}(e))}$. With Lemma 5.1 it again follows that $\widetilde{X} - \{(e, e')\} \in C_{\mathit{ref}(\mathcal{C})}$.

(iii) Let $\widetilde{X}, \widetilde{Y}, \widetilde{Z} \in C_{\mathit{ref}(\mathcal{C})}$ and $\widetilde{X} \cup \widetilde{Y} \subseteq \widetilde{Z}$. Then $\widetilde{X} = \bigcup_{e \in X}\{e\} \times X_e$ with $\forall e \in X : X_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))}$ and $\forall X^* \subseteq \mathit{busy}_X(\widetilde{X}) : X - X^* \in C_{\mathcal{C}}$. Likewise $\widetilde{Y} = \bigcup_{e \in Y}\{e\} \times Y_e$ with $\forall e \in Y : Y_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))}$ and $\forall Y^* \subseteq \mathit{busy}_Y(\widetilde{Y}) : Y - Y^* \in C_{\mathcal{C}}$, and $\widetilde{Z} = \bigcup_{e \in Z}\{e\} \times Z_e$ with $Z \in C_{\mathcal{C}}$ and $\forall e \in Z : Z_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))}$. For $e \in Y - X$ write $X_e := \emptyset$ and for $e \in X - Y$ write $Y_e := \emptyset$. Then $\widetilde{X} \cup \widetilde{Y} = \bigcup_{e \in X \cup Y}\{e\} \times (X_e \cup Y_e)$. As $X \cup Y \subseteq Z$ we have $X \cup Y \in C_{\mathcal{C}}$, and as $X_e \cup Y_e \subseteq Z_e$ we have $X_e \cup Y_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))}$ for all $e \in X \cup Y$. Let $W \subseteq \mathit{busy}_{X \cup Y}(\widetilde{X} \cup \widetilde{Y})$. Then $W \cap X \subseteq \mathit{busy}_X(\widetilde{X})$ and $W \cap Y \subseteq \mathit{busy}_Y(\widetilde{Y})$. Hence $X - W \in C_{\mathcal{C}}$ and $Y - W \in C_{\mathcal{C}}$, so $(X \cup Y) - W = (X - W) \cup (Y - W) \in C_{\mathcal{C}}$. Using Lemma 5.1 it follows that $\widetilde{X} \cup \widetilde{Y} \in C_{\mathit{ref}(\mathcal{C})}$.

(iv) Let $\widetilde{X}, \widetilde{Y}, \widetilde{Z} \in C_{\mathit{ref}(\mathcal{C})}$ and $\widetilde{X} \cup \widetilde{Y} \subseteq \widetilde{Z}$. Then $\widetilde{X} = \bigcup_{e \in X}\{e\} \times X_e$ with $\forall e \in X : X_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))}$ and $\forall X^* \subseteq \mathit{busy}_X(\widetilde{X}) : X - X^* \in C_{\mathcal{C}}$. Likewise $\widetilde{Y} = \bigcup_{e \in Y}\{e\} \times Y_e$ with $\forall e \in Y : Y_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))}$ and $\forall Y^* \subseteq \mathit{busy}_Y(\widetilde{Y}) : Y - Y^* \in C_{\mathcal{C}}$, and $\widetilde{Z} = \bigcup_{e \in Z}\{e\} \times Z_e$ with $Z \in C_{\mathcal{C}}$ and $\forall e \in Z : Z_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))}$. Now $\widetilde{X} \cap \widetilde{Y} = \bigcup_{e \in X \cap Y}\{e\} \times (X_e \cap Y_e)$. As $X \cup Y \subseteq Z$ we have $X \cap Y \in C_{\mathcal{C}}$, and as $X_e \cup Y_e \subseteq Z_e$ we have $X_e \cap Y_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))}$ for all $e \in X \cap Y$. Let $W \subseteq \mathit{busy}_{X \cap Y}(\widetilde{X} \cap \widetilde{Y})$. Then it must be possible to write W as $W = X^* \cup Y^*$ with $X^* \subseteq \mathit{busy}_X(\widetilde{X})$ and $Y^* \subseteq \mathit{busy}_Y(\widetilde{Y})$. Now $X - X^* \in C_{\mathcal{C}}$ and $Y - Y^* \in C_{\mathcal{C}}$, so $(X \cap Y) - W = (X - X^*) \cap (Y - Y^*) \in C_{\mathcal{C}}$. Using Lemma 5.1 it follows that $\widetilde{X} \cap \widetilde{Y} \in C_{\mathit{ref}(\mathcal{C})}$. ∎

Finally we establish that for stable configuration structures the causality relations on refined structures are completely determined by the causality relations on the original structures and the ones on the structures refining actions.

Proposition 5.7 Let $\mathcal{C} \in \mathbb{C}_{stable}$, let $X \in C_{\mathcal{C}}$.

Let $\widetilde{X}$ be a refinement of X by a refinement function *ref* — $\widetilde{X} = \bigcup_{f \in X} \{f\} \times X_f$. Then $(d, d') <_{\widetilde{X}} (e, e') \Leftrightarrow (d <_X e) \vee (d = e \wedge d' <_{X_e} e')$.

Proof "$\Rightarrow$": Assume $(d, d') <_{\widetilde{X}} (e, e')$, i.e. for all $\widetilde{Y} \subseteq \widetilde{X}$ with $(e, e') \in \widetilde{Y} \in C_{ref(\mathcal{C})}$ we have $(d, d') \in \widetilde{Y}$.

Suppose $d \neq e$. Let $Y \subseteq X$ with $e \in Y \in C_{\mathcal{C}}$. We have to show that $d \in Y$. Let $\widetilde{Y} := \bigcup_{f \in Y} \{f\} \times X_f$. Since $\widetilde{X}$ is a refinement of X, $X_f \in C_{ref(l_{\mathcal{C}}(f))} - \{\emptyset\}$ for all $f \in Y$. By construction of $\widetilde{Y}$ we have $busy(\widetilde{Y}) \subseteq busy(\widetilde{X})$, using that $Y \subseteq X$. Now let $Z \subseteq busy(\widetilde{Y}) \subseteq busy(\widetilde{X})$. Then $Y - Z = (X - Z) \cap Y \in C_{\mathcal{C}}$, since $\widetilde{X}$ is a refinement of X and $\mathcal{C}$ is closed under bounded intersections. Hence $\widetilde{Y} \in C_{ref(l_{\mathcal{C}}(e))}$. By construction $\widetilde{Y} \subseteq \widetilde{X}$ and $(e, e') \in \widetilde{Y}$, using that $e \in Y$. So, by assumption, $(d, d') \in \widetilde{Y}$, hence $d \in Y$.

Now suppose $d = e$. Let $Y' \subseteq X_e$ with $e' \in Y' \in C_{ref(l_{\mathcal{C}}(e))}$. We have to show that $d' \in Y'$. Let $Y = \{f \in X \mid f <_X e\}$ and $\widetilde{Y} := \bigcup_{f \in Y}(\{f\} \times X_f) \cup (\{e\} \times Y')$. Since $\widetilde{X}$ is a refinement of X, $X_f \in C_{ref(l_{\mathcal{C}}(f))} - \{\emptyset\}$ for all $f \in Y$; $Y' \in C_{ref(l_{\mathcal{C}}(e))} - \{\emptyset\}$ by assumption. By construction of $\widetilde{Y}$ we have $busy(\widetilde{Y}) \subseteq busy(\widetilde{X}) \cup \{e\}$. We claim that even $busy(\widetilde{Y}) \subseteq \{e\}$. Namely, let $f \in busy(\widetilde{Y}) - \{e\} \subseteq busy(\widetilde{X})$. Then $X - \{f\} \in C_{\mathcal{C}}$ by Definition 5.4. However, as $f \in Y$ we have $f <_X e \in X$, so $X - \{f\}$ is not left-closed w.r.t. $<_X$, contradicting Proposition 5.3. Thus $busy(\widetilde{Y}) \subseteq \{e\}$. As both $Y \cup \{e\} \in C_{\mathcal{C}}$ and $Y \in C_{\mathcal{C}}$ by Proposition 5.3, we have $\widetilde{Y} \in C_{ref(l_{\mathcal{C}}(e))}$. By construction $\widetilde{Y} \subseteq \widetilde{X}$ and $(e, e') \in \widetilde{Y}$, using that $e' \in Y'$. So, by assumption, $(d, d') \in \widetilde{Y}$, hence $d' \in Y'$.

"$\Leftarrow$": Assume $d, e \in X$, $d' \in X_d$, $e' \in X_e$ and $d <_X e$, i.e. for all $Y \subseteq X$ with $e \in Y \in C_{\mathcal{C}}$ one has $d \in Y$. Let $\widetilde{Y} \in C_{ref(l_{\mathcal{C}}(e))}$ with $(e, e') \in \widetilde{Y} \subseteq \widetilde{X}$. We have to show that $(d, d') \in \widetilde{Y}$. Let $\widetilde{Y} := \bigcup_{f \in Y} \{f\} \times Y_f$. As $(e, e') \in \widetilde{Y} \subseteq \widetilde{X}$ we have $e \in Y \subseteq X$. Hence $d \in Y$. As $Y - \{d\}$ is not left-closed w.r.t. $<_X$, $Y - \{d\} \notin C_{\mathcal{C}}$ by Proposition 5.3. Hence $d \notin busy(\widetilde{Y})$, and $Y_d \in \surd_{ref(l_{\mathcal{C}}(e))}$. We have $Y_d \subseteq X_d \in C_{ref(l_{\mathcal{C}}(e))}$, and as by Definition 5.1 terminating configurations are maximal, $Y_d = X_d$. Since $d' \in X_d = Y_d$ it follows that $(d, d') \in \widetilde{Y}$.

Now assume $e \in X$, $d', e' \in X_e$ and $d' <_{X_e} e'$, i.e. for all $Y' \subseteq X_e$ with $e' \in Y' \in C_{ref(l_{\mathcal{C}}(e))}$ one has $d' \in Y'$. Let $\widetilde{Y} \in C_{ref(l_{\mathcal{C}}(e))}$ with $(e, e') \in \widetilde{Y} \subseteq \widetilde{X}$. We have to show that $(e, d') \in \widetilde{Y}$. Let $\widetilde{Y} := \bigcup_{f \in Y} \{f\} \times Y_f$. We have $e' \in Y_e \subseteq X_e$. Hence $d' \in Y_e$ and $(e, d') \in \widetilde{Y}$. ■

Part II

Equivalence notions

In this part of the paper, we will consider the spectrum of equivalence notions outlined in the introduction and investigate which equivalences are preserved under refinement of actions, and which are even a congruence for action refinement.

Definition II.1 Let $\approx$ be an equivalence relation on event or configuration structures.

(i) For refinement functions ref and ref' we write $\mathit{ref} \approx \mathit{ref}'$ iff $\mathit{ref}(a) \approx \mathit{ref}'(a)$ for all $a \in \mathit{Act}$.

(ii) $\approx$ is *preserved under action refinement* iff $\mathcal{C} \approx \mathcal{D} \Rightarrow \mathit{ref}(\mathcal{C}) \approx \mathit{ref}(\mathcal{D})$ for all systems $\mathcal{C}$ and $\mathcal{D}$ and all refinement functions ref.

(iii) $\approx$ is *a congruence for action refinement* iff $\mathcal{C} \approx \mathcal{D} \wedge \mathit{ref} \approx \mathit{ref}' \Rightarrow \mathit{ref}(\mathcal{C}) \approx \mathit{ref}'(\mathcal{D})$ for all systems $\mathcal{C}$ and $\mathcal{D}$ and all refinement functions ref and ref'.

Following the classification discussed in the introduction, the semantic equivalences considered in this paper can be positioned in a two-dimensional diagram as shown below.

conflict structure \ runs		*interleaving semantics*	*step semantics*	. . .	*causal semantics*
		sequences of actions	sequences of steps		partial orders (pomsets)
branching time · · · *linear time*	sets of traces	interleaving trace equivalence $\approx_{it}$	step trace equivalence $\approx_{st}$		pomset trace equivalence $\approx_{pt}$
	⋮				
	bisimulation	interleaving bisimulation equivalence $\approx_{ib}$	step bisimulation equivalence $\approx_{sb}$		• pomset bisimulation equiv. $\approx_{pb}$ • weak history preserving eq. $\approx_{wh}$ • the combination of both: $\approx_{whpb}$ • history preserving equiv. $\approx_h$ • hereditary hist. pres. eq. $\approx_{hh}$

[dark grey] means: not preserved under refinement — [light grey] means: preserved under refinement

The canonical representatives of linear time and branching time interleaving equivalences are *interleaving trace equivalence* ($\approx_{it}$) and *interleaving bisimulation equivalence* ($\approx_{ib}$), respectively. Interleaving trace equivalence generalises in a canonical way to step and partial order semantics, yielding *step trace equivalence* ($\approx_{st}$) and *pomset trace equivalence* ($\approx_{pt}$). Likewise *step bisimulation equivalence* ($\approx_{sb}$) is the canonical generalisation of $\approx_{ib}$ to step semantics. However, several generalisations of $\approx_{ib}$ to partial order semantics have been defined in the literature. The two original proposals are the *pomset bisimulation equivalence* ($\approx_{pb}$) of [BC-a] (there called *equipollence*), defined on event structures, and the *NMS partial ordering equivalence* of [DDM-a], defined on *nondeterministic measurement systems*. We rephrase the latter as an equivalence on stable event and configuration structures under the name *weak history preserving equivalence* ($\approx_{wh}$). A strengthening of this equivalence is the *BS-bisimulation* of [RT], defined on *behaviour structures*. Again we provide a definition for this notion on stable event and configurations structures, and call it *history preserving (bisimulation) equivalence* ($\approx_h$).

As our system model is equipped with a predicate indicating successful termination, each equivalence can be defined in two ways: a termination sensitive variant, taking this predicate into account, and a termination insensitive variant, abstracting from this information. Systematically, we will use the termination insensitive variant of linear time equivalences, and the termination sensitive variant of branching time equivalences. In this way, most notions studied in the literature are lying in between.

Between the extremes considered here (interleaving semantics versus partial order semantics, trace semantics versus bisimulation semantics) other paradigms are being investigated in the literature. We will comment on these in the conclusion, giving a more complete version of the above diagram.

As mentioned in the introduction, we will show that none of the interleaving and step equivalences of the spectrum (also not those between trace and bisimulation semantics) is preserved under action refinement, whereas pomset trace equivalence is; its termination sensitive variant is even a congruence for action refinement. Then we give examples, showing that pomset bisimulation equivalence and weak history preserving bisimulation are not preserved under refinement of actions. We also show that weak history preserving equivalence does not imply pomset bisimulation equivalence and vice versa; these notions are incomparable. Next, we provide an example demonstrating that the obvious combination of both equivalences ($\approx_{whpb}$) is also not preserved under action refinement. Finally we show that history preserving equivalence is finer than each of these notions and is preserved under refinement. Again its termination sensitive variant is a congruence. For systems without autoconcurrency we show that history preserving and weak history preserving equivalence coincide. We also compare history preserving equivalence with a strengthening proposed by [Bednarczyk] under the name *hereditary history preserving equivalence* ($\approx_{hh}$), which is also preserved under action refinement.

Our definitions and theorems will be formulated in terms of stable configuration structures, this being the most general and abstract event oriented model which allows the representation of causality by means of partial orders. They then also apply to prime, flow and stable event structures. In particular, an equivalence $\approx$ on stable configuration structures is inherited by these models of event structures by writing $\mathcal{E} \approx \mathcal{F}$ iff $\mathcal{C}(\mathcal{E}) \approx \mathcal{C}(\mathcal{F})$ for prime, flow or stable event structures $\mathcal{E}$ and $\mathcal{F}$. In fact, our definitions lift to such event structures verbatim. On the other hand, our counterexamples, showing that certain equivalences are not preserved under action refinement, are phrased in terms of prime event structures, and therefore also apply to the other, more general, models.

6 Interleaving semantics

In this section, we define the two interleaving equivalences on both ends of the linear time – branching time spectrum: *(termination insensitive) interleaving trace equivalence* and *(termination sensitive) interleaving bisimulation equivalence*. We show that these two equivalences, as well as all (interleaving) equivalences lying in between, are not preserved under action refinement.

We start by defining a transition relation for configurations.

Definition 6.1 Let $\mathcal{C} \in \mathbb{C}_{stable}$.

$X \xrightarrow{a}_{\mathcal{C}} X'$ iff $a \in Act$, $X, X' \in C_{\mathcal{C}}$, $X \subseteq X'$ and $X' - X = \{e\}$ with $l_{\mathcal{C}}(e) = a$.

Here $X \xrightarrow{a}_{\mathcal{C}} X'$ says that if $\mathcal{C}$ is in the state represented by X, then it may perform an action a and reach a state represented by X'. This transition relation associates a labelled transition system with each configuration structure. Hence every (interleaving) equivalence defined on labelled transition systems immediately induces a corresponding equivalence on configuration structures. Here we just give explicit characterisations of the so obtained interleaving versions of trace and bisimulation equivalence.

Definition 6.2

A sequence $a_1 \cdots a_n \in Act^*$ is a *(sequential) trace* of $\mathcal{C} \in \mathbb{C}_{stable}$ iff there exist configurations $X_0, \ldots, X_n \in C_{\mathcal{C}}$ such that $X_0 = \emptyset$ and $X_{i-1} \xrightarrow{a_i}_{\mathcal{C}} X_i$ $(i = 1, ..., n)$.
$SeqTr\,(\mathcal{C})$ denotes the set of all sequential traces of $\mathcal{C} \in \mathbb{C}_{stable}$.
$\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$ are *interleaving trace equivalent* ($\mathcal{C} \approx_{it} \mathcal{D}$) iff $SeqTr(\mathcal{C}) = SeqTr(\mathcal{D})$.

Definition 6.3 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$.

A relation $R \subseteq C_{\mathcal{C}} \times C_{\mathcal{D}}$ is called an *(termination sensitive) interleaving bisimulation between $\mathcal{C}$ and $\mathcal{D}$* iff $(\emptyset, \emptyset) \in R$ and if $(X, Y) \in R$ then

- $X \xrightarrow{a}_{\mathcal{C}} X'$, $a \in Act \Rightarrow \exists Y'$ with $Y \xrightarrow{a}_{\mathcal{D}} Y'$ and $(X', Y') \in R$,
- $Y \xrightarrow{a}_{\mathcal{D}} Y'$, $a \in Act \Rightarrow \exists X'$ with $X \xrightarrow{a}_{\mathcal{C}} X'$ and $(X', Y') \in R$,
- $X \in \surd_{\mathcal{C}} \Leftrightarrow Y \in \surd_{\mathcal{D}}$.

$\mathcal{C}$ and $\mathcal{D}$ are *interleaving bisimulation equivalent* ($\mathcal{C} \approx_{ib}^{\surd} \mathcal{D}$) iff there exists an interleaving bisimulation between $\mathcal{C}$ and $\mathcal{D}$.

Note that adding the third requirement turns the usual termination insensitive notion of bisimulation [Milner-b] into a termination sensitive variant [BW].

Clearly, $\mathcal{C} \approx_{ib}^{\surd} \mathcal{D}$ implies $\mathcal{C} \approx_{it} \mathcal{D}$. Most other interleaving equivalences studied in the literature can be positioned in between [vG-c] (recall that we do not consider abstraction from internal actions).

Example 6.1

We now recall the example of [CDP], showing that both $\approx_{it}$ and $\approx_{ib}^{\surd}$ are not preserved under refinement (cf. also Example 1.4). Consider the two systems $P := a|b$ and $Q := a;b + b;a$, representable by the following prime event structures.

$$\mathcal{E}_P \;=\; a \quad b \quad , \quad \mathcal{E}_Q \;=\; \begin{array}{ccc} a & \# & b \\ \downarrow & & \downarrow \\ b & & a \end{array}$$

In all known interleaving semantics, P and Q are considered equivalent; we have $\mathcal{E}_P \approx_{ib}^{\surd} \mathcal{E}_Q$. However, if we refine the action a into the pomset $a_1 \rightarrow a_2$ we obtain the two systems

$$ref(\mathcal{E}_P) \;=\; \begin{array}{cc} a_1 & b \\ \downarrow & \\ a_2 & \end{array} \quad , \quad ref(\mathcal{E}_Q) \;=\; \begin{array}{ccc} a_1 & \# & b \\ \downarrow & & \downarrow \\ a_2 & & a_1 \\ \downarrow & & \downarrow \\ b & & a_2 \end{array}$$

which are not even interleaving trace equivalent: $\mathcal{E}_{P'}$ allows for the sequence $a_1\, b\, a_2$ whereas $\mathcal{E}_{Q'}$ doesn't.

This shows that both interleaving trace equivalence and interleaving bisimulation equivalence are not preserved under action refinement. Even more, the same follows for all equivalences identifying $\mathcal{E}_P$ and $\mathcal{E}_Q$ and distinguishing $ref(\mathcal{E}_P)$ and $ref(\mathcal{E}_Q)$, in particular for all equivalences $\approx$ in the linear time – branching time spectrum with $\approx_{ib}^{\surd} \subseteq \approx \subseteq \approx_{it}$.

As equivalences which are preserved under refinement one can consider isomorphism on the various models of event structures (Propositions 2.1, 3.2 and 4.2) or isomorphism on configuration structures (Proposition 5.1). However, the main purpose of introducing an equivalence notion is to abstract from certain details in a system representation. For example, we would like to express that the processes a and $a+a$ exhibit the same behaviour. Furthermore, we would like to identify processes like $(b|(a+c)) + (a|b) + ((b+c)|a)$ and $(b|(a+c)) + ((b+c)|a)$ (absorption law, see [BC-a]). This is not possible when using isomorphism. Hence, in the sequel we will consider various equivalence notions in between these two extremes (the interleaving equivalences and the isomorphisms), taking into account the concurrency and conflict structure in more and more detail.

7 Step semantics

A more discriminating view of concurrent systems than that offered by interleaving semantics is obtained by modelling concurrency as either arbitrary interleaving or simultaneous execution. In this section we generalise the single action transitions $X \xrightarrow{a} X'$ from the previous section to transitions of the form $X \xrightarrow{A} X'$ where A is a multiset over Act, representing actions occurring concurrently. In particular, we allow actions to occur concurrently with themselves ("autoconcurrency"). Using this new kind of transitions, various equivalences can be obtained as straightforward generalisations of the corresponding interleaving equivalences, see e.g. [Pomello]. As before, we explicitly define just the two extreme ones, namely *(termination insensitive) step trace equivalence* and *(termination sensitive) step bisimulation equivalence.* Then we provide examples showing that these two equivalences, as well as all (step) equivalences lying in between, are not preserved under action refinement.

Definition 7.1 Let $\mathcal{C} \in \mathbb{C}_{stable}$.

> $X \xrightarrow{A}_{\mathcal{C}} X'$ iff $A \in \mathbb{N}^{Act}$ (A is a multiset over Act), $X, X' \in C_{\mathcal{C}}$, $X \subseteq X'$ and $X' - X = G$ such that $\forall d, e \in G:\ d\ co_{X'}\ e$ and $l_{\mathcal{C}}(G) = A$. Here $l_{\mathcal{C}}(G) \in \mathbb{N}^{Act}$ is given by $l_{\mathcal{C}}(G)(a) := |\{e \in G \mid l_{\mathcal{C}}(e) = a\}|$.

Definition 7.2

> A sequence $A_1 \cdots A_n$ where $A_i \in \mathbb{N}^{Act}$ $(i = 1, ..., n)$ is a *step trace* of $\mathcal{C} \in \mathbb{C}_{stable}$ iff there exist $X_0, \cdots, X_n \in C_{\mathcal{C}}$ such that $X_0 = \emptyset$ and $X_{i-1} \xrightarrow{A_i} X_i$ $(i = 1, ..., n)$.
> $StepTr\,(\mathcal{C})$ denotes the set of all step traces of $\mathcal{C} \in \mathbb{C}_{stable}$.
> $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$ are *step trace equivalent* ($\mathcal{C} \approx_{st} \mathcal{D}$) iff $StepTr\,(\mathcal{C}) = StepTr\,(\mathcal{D})$.

Definition 7.3 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$.

> A relation $R \subseteq C_{\mathcal{C}} \times C_{\mathcal{D}}$ is called a *(termination sensitive) step bisimulation between $\mathcal{C}$ and $\mathcal{D}$* iff $(\emptyset, \emptyset) \in R$ and if $(X, Y) \in R$ then
>
> - $X \xrightarrow{A}_{\mathcal{C}} X'$, $A \in \mathbb{N}^{Act} \Rightarrow \exists Y'$ with $Y \xrightarrow{A}_{\mathcal{D}} Y'$ and $(X, Y) \in R$,
> - $Y \xrightarrow{A}_{\mathcal{D}} Y'$, $A \in \mathbb{N}^{Act} \Rightarrow \exists X'$ with $X \xrightarrow{A}_{\mathcal{C}} X'$ and $(X, Y) \in R$,
> - $X \in \surd_{\mathcal{C}} \Leftrightarrow Y \in \surd_{\mathcal{D}}$.
>
> $\mathcal{C}$ and $\mathcal{D}$ are *step bisimulation equivalent* ($\mathcal{E} \approx_{sb}^{\surd} \mathcal{D}$) iff there exists a step bisimulation between $\mathcal{C}$ and $\mathcal{D}$.

Clearly the two event structures $\mathcal{E}_P$ and $\mathcal{E}_Q$ in Example 6.1 are not equivalent in step semantics. The step $\{a, b\}$ is possible in $\mathcal{E}_P$ but not in $\mathcal{E}_Q$.

As for interleaving, $\mathcal{C} \approx_{sb}^{\surd} \mathcal{D}$ implies $\mathcal{C} \approx_{st} \mathcal{D}$. Moreover (as far as we know) all other interesting step equivalence notions are positioned somewhere in between.

The following example shows that step trace semantics is in general not preserved under refinement.

Example 7.1

We consider the two systems

$$\mathcal{E} := \begin{tikzcd}[row sep=small, column sep=small] a \arrow[d] & c \\ b \end{tikzcd} \quad \text{and} \quad \mathcal{F} := \begin{tikzcd}[row sep=small, column sep=small] a \arrow[dr] & & c \arrow[dl] \\ & b & \end{tikzcd} + \begin{tikzcd}[row sep=small, column sep=small] & a \arrow[dl] \arrow[dr] & \\ b & & c \end{tikzcd} .$$

The +-sign in $\mathcal{F}$ may easily be "implemented" by putting all events in the first component in conflict with all events in the second component.

These two systems are step trace equivalent. However, when refining c into $c_1 \rightarrow c_2$, the resulting systems

$$ref(\mathcal{E}) = \begin{tikzcd}[row sep=small, column sep=small] a \arrow[d] & c_1 \arrow[d] \\ b & c_2 \end{tikzcd} \quad \text{and} \quad ref(\mathcal{F}) = \begin{tikzcd}[row sep=small, column sep=small] & & c_1 \arrow[d] \\ a \arrow[dr] & & c_2 \arrow[dl] \\ & b & \end{tikzcd} + \begin{tikzcd}[row sep=small, column sep=small] & a \arrow[dl] \arrow[dr] & \\ b & & c_1 \arrow[d] \\ & & c_2 \end{tikzcd}$$

are not step trace equivalent (not even interleaving trace equivalent).

This example shows that $\approx_{st}$ is not preserved under refinement. However, the example is not adequate for step bisimulation equivalence since $\mathcal{E}$ and $\mathcal{F}$ are not step bisimulation equivalent (after performing a, the b is always possible in $\mathcal{E}$ but not always in $\mathcal{F}$). The next example shows that also $\approx_{sb}^{\surd}$ is not preserved under refinement.

Example 7.2

Consider $P := a|b$ and $Q := (a|b) + a;b$,

$$\mathcal{E}_P = a \quad b \quad , \quad \mathcal{E}_Q = \begin{tikzcd}[row sep=small, column sep=small] a \ \# & a \arrow[d] & \# \ b \\ & b & \end{tikzcd} .$$

It is easy to verify that $\mathcal{E}_P \approx_{sb}^{\surd} \mathcal{E}_Q$. However, refining a into $a_1 \rightarrow a_2$ yields

$$ref(\mathcal{E}_P) = \begin{tikzcd}[row sep=small, column sep=small] a_1 \arrow[d] & b \\ a_2 & \end{tikzcd} \quad , \quad ref(\mathcal{E}_Q) = \begin{tikzcd}[row sep=small, column sep=small] a_1 \arrow[d] & \# & a_1 \arrow[d] & \# & b \\ a_2 & & a_2 \arrow[d] & & \\ & & b & & \end{tikzcd}$$

After the step $\{a_1\}$, the step $\{b\}$ is always possible in $\mathit{ref}(\mathcal{E}_P)$. However, in $\mathit{ref}(\mathcal{E}_Q)$, it may be the case that the step $\{b\}$ is impossible after executing a_1 (choosing the branch $a_1 \rightarrow a_2 \rightarrow b$). Hence $\mathit{ref}(\mathcal{E}_P)$ and $\mathit{ref}(\mathcal{E}_Q)$ are not step bisimulation equivalent (not even interleaving bisimulation equivalent).

However, this example is still not suitable for disqualifying the whole range of equivalence notions included between $\approx_{st}$ and $\approx_{sb}^{\surd}$, as Example 6.1 does in the interleaving case, since the refined systems $\mathit{ref}(\mathcal{E}_P)$ and $\mathit{ref}(\mathcal{E}_Q)$ turn out to be step trace equivalent. A slightly more complicated example is given below, disqualifying all equivalence notions between $\approx_{sb}^{\surd}$ and $\approx_{st}$, including the step failure equivalence of [TV].

Example 7.3

First consider the following three systems:

$$\mathcal{E}_1 := \begin{tikzcd}[column sep=small] & a \arrow[dl] \arrow[dr] & \\ b & & c \end{tikzcd}, \quad \mathcal{E}_2 := \begin{tikzcd}[column sep=small] a \arrow[dr] & & c \arrow[dl] \\ & b & \end{tikzcd}, \quad \mathcal{E}_3 := \begin{tikzcd}[column sep=small] a \arrow[d] & c \\ b & \end{tikzcd}.$$

Now we consider the two composed systems

$$\mathcal{E} := \mathcal{E}_1 + \mathcal{E}_2, \qquad \mathcal{F} := \mathcal{E}_1 + \mathcal{E}_2 + \mathcal{E}_3.$$

We have $\mathcal{E} \approx_{sb}^{\surd} \mathcal{F}$ [GV-a]. However, when refining c into $c_1 \rightarrow c_2$ only the refinement of $\mathcal{F}$ may perform the sequence of actions $c_1\ a\ b\ c_2$. The resulting systems $\mathit{ref}(\mathcal{E})$ and $\mathit{ref}(\mathcal{F})$ are not even interleaving trace equivalent.
So let $\approx$ be any equivalence included between $\approx_{st}$ and $\approx_{sb}^{\surd}$, then also $\mathcal{E} \approx \mathcal{F}$, but $\mathit{ref}(\mathcal{E}) \not\approx \mathit{ref}(\mathcal{F})$.

Thus we have shown that all the currently known versions of step equivalence are not preserved under refinement.

8 Linear time partial order semantics

In [CDP] it was claimed that equivalence based on considering partially ordered executions is preserved under refinement. In this section we will make this claim more precise. We will show that this is indeed true in linear time semantics, formalising the proof sketch from [CDP] in terms of configuration structures. However, in the next section, we will consider equivalence notions for branching time semantics, and it will turn out that in this case the claim is not so obvious.

In Part I, we discussed that the possible executions of a system may be represented as isomorphism classes of labelled partial orders (*pomsets*), thus taking full account of the causality relation for event occurrences.

Definition 8.1

For $\mathcal{C} \in \mathbb{C}_{stable}$ let $Pomsets(\mathcal{C}) := \{pomset(X) \mid X \in C_{\mathcal{C}}\}$.
$\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$ are *pomset trace equivalent* ($\mathcal{C} \approx_{pt} \mathcal{D}$) iff $Pomsets\ (\mathcal{C}) = Pomsets\ (\mathcal{D})$.

Clearly, pomset trace equivalence implies step trace equivalence. Example 7.1 shows that pomset trace equivalence is strictly finer than step trace equivalence. On the other hand, pomset trace equivalence and step bisimulation equivalence (or interleaving bisimulation equivalence) are incomparable: $a;(b+c) \underset{\not\approx_{sb}}{\approx_{pt}} a;b+a;c$, and for $\mathcal{E}_P$ and $\mathcal{E}_Q$ of Example 7.2, $\mathcal{E}_P \approx_{sb} \mathcal{E}_Q$ but $\mathcal{E}_P \not\approx_{pt} \mathcal{E}_Q$.

The following theorem shows that pomset trace equivalence is preserved under refinement.

Theorem 8.1 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$.

Then $\mathcal{C} \approx_{pt} \mathcal{D}$ implies $ref(\mathcal{C}) \approx_{pt} ref(\mathcal{D})$ for any refinement function *ref*.

Proof Let $\mathcal{C} \approx_{pt} \mathcal{D}$ and let *ref* be a refinement function. We have to show

$$Pomsets\ (ref(\mathcal{C})) = Pomsets\ (ref(\mathcal{D})).$$

"$\subseteq$": Let $u \in Pomsets\ (ref(\mathcal{C}))$.
Then $u = [(\widetilde{X}, <_{\widetilde{X}}, l_{ref(\mathcal{C})} \upharpoonright \widetilde{X})]_{\cong}$ where $\widetilde{X} \in C_{ref(\mathcal{C})}$.

With Definition 5.4 we have that $\widetilde{X}$ is a refinement of some configuration X of $\mathcal{C}$. Since $Pomsets\ (\mathcal{C}) = Pomsets\ (\mathcal{D})$, there exists $Y \in C_{\mathcal{D}}$ such that $(X, <_X, l_{\mathcal{C}} \upharpoonright X)$ and $(Y, <_Y, l_{\mathcal{D}} \upharpoonright Y)$ are isomorphic. Since isomorphism preserves labelling, we can refine Y to a configuration $\widetilde{Y}$ (by choosing identical refinements for corresponding events) such that

$$(\widetilde{X}, <_{\widetilde{X}}, l_{ref(\mathcal{C})} \upharpoonright \widetilde{X}) \cong (\widetilde{Y}, <_{\widetilde{Y}}, l_{ref(\mathcal{D})} \upharpoonright \widetilde{Y}),$$

hence $u \in Pomsets\ (ref(\mathcal{D}))$.

"$\supseteq$": by symmetry. ■

On the domain of prime event structures, $\approx_{pt}$ is even a congruence for refinement of actions (by non-empty, finite, conflict-free prime event structures). This follows from Proposition 2.1(ii) and Theorem 8.1, since two finite, conflict-free prime event structures are pomset trace equivalent iff they are isomorphic, and hence $\mathit{ref} \approx_{pt} \mathit{ref}' \Leftrightarrow \mathit{ref} \cong \mathit{ref}'$.

However, for flow event structures (or more general models) $\approx_{pt}$ is not a congruence.

Example 8.1

Let $\mathcal{E}$ be the flow event structure $a \longrightarrow b$.
Let $\mathit{ref}(a) := c$, $\mathit{ref}'(a) := c \longrightarrow \boxed{d}$ and $\mathit{ref}(a') := a' =: \mathit{ref}'(a')$ for $a' \neq a$.
Then $\mathit{ref} \approx_{pt} \mathit{ref}'$, but $\mathit{ref}(\mathcal{E}) \not\approx_{pt} \mathit{ref}'(\mathcal{E})$.

This problem can be solved by using a termination sensitive variant of pomset trace equivalence.

Definition 8.2

For $\mathcal{C} \in \mathbb{C}_{stable}$ let $\surd$-$\mathit{Pomsets}(\mathcal{C}) := \{\mathit{pomset}(X) \mid X \in \surd_{\mathcal{C}}\}$.
$\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$ are *termination sensitive pomset trace equivalent* ($\mathcal{C} \approx_{pt}^{\surd} \mathcal{D}$) iff $\mathit{Pomsets}\ (\mathcal{C}) = \mathit{Pomsets}\ (\mathcal{D})$ and $\surd$-$\mathit{Pomsets}\ (\mathcal{C}) = \surd$-$\mathit{Pomsets}\ (\mathcal{D})$.

Theorem 8.2 Let $\mathcal{C} \in \mathbb{C}_{stable}$ and let ref, ref' be refinement functions.

Then $\mathit{ref} \approx_{pt}^{\surd} \mathit{ref}'$ implies $\mathit{ref}(\mathcal{C}) \approx_{pt}^{\surd} \mathit{ref}'(\mathcal{C})$.

Proof Let $\mathit{ref} \approx_{pt}^{\surd} \mathit{ref}'$. We have to show that $\mathit{Pomsets}(\mathit{ref}(\mathcal{C})) = \mathit{Pomsets}(\mathit{ref}'(\mathcal{C}))$ and $\surd$-$\mathit{Pomsets}(\mathit{ref}(\mathcal{C})) = \surd$-$\mathit{Pomsets}(\mathit{ref}'(\mathcal{C}))$.

"$\subseteq$": Let $u \in \mathit{Pomsets}(\mathit{ref}(\mathcal{C}))$. Then $u = [(\widetilde{X}, <_{\widetilde{X}}, l_{\mathit{ref}(\mathcal{C})} \restriction \widetilde{X})]_{\cong}$ where $\widetilde{X} \in C_{\mathit{ref}(\mathcal{C})}$.

With Definition 5.4 we have that $\widetilde{X}$ is a refinement of some configuration X of $\mathcal{C}$, i.e. $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ where $\forall e \in X : X_e \in C_{\mathit{ref}(l_{\mathcal{C}}(e))} - \{\emptyset\}$ and $\forall Y \subseteq \mathit{busy}\ (\widetilde{X}) :\ X - Y \in C_{\mathcal{C}}$.
As $\mathit{ref} \approx_{pt}^{\surd} \mathit{ref}'$, for all $e \in X$ there is an $Y_e \in C_{\mathit{ref}'(l_{\mathcal{C}}(e))} - \{\emptyset\}$ such that $\mathit{pomset}(X_e) = \mathit{pomset}(Y_e)$ and $Y_e \in \surd_{\mathit{ref}'(l_{\mathcal{C}}(e))}$ iff $X_e \in \surd_{\mathit{ref}(l_{\mathcal{C}}(e))}$.
It follows that $\widetilde{Y} := \bigcup_{e \in X} \{e\} \times Y_e \in C_{\mathit{ref}'(\mathcal{C})}$, since in particular $\mathit{busy}(\widetilde{X}) = \mathit{busy}(\widetilde{Y})$.
It remains to be shown that $\mathit{pomset}(\widetilde{Y}) = \mathit{pomset}(\widetilde{X})$.
For each $e \in X$, let $f_e : X_e \rightarrow Y_e$ be an isomorphism between $(X_e, <_{X_e}, l_{\mathit{ref}(l_{\mathcal{C}}(e))})$ and $(Y_e, <_{Y_e}, l_{\mathit{ref}'(l_{\mathcal{C}}(e))})$.
Define $\widetilde{f} : \widetilde{X} \rightarrow \widetilde{Y}$ by $\widetilde{f}(e, e') := (e, f_e(e'))$.

- $\widetilde{f}$ is a bijection, since all the f_e's are bijections.

- $(d, d') <_{\widetilde{X}} (e, e') \Leftrightarrow d <_X e \vee (d = e \wedge d' <_{X_e} e')$
$\Leftrightarrow d <_X e \vee (d = e \wedge f_e(d') <_{X_e} f_e(e')) \Leftrightarrow \widetilde{f}(d, d') <_{\widetilde{Y}} \widetilde{f}(e, e')$, using Proposition 5.7.
- $l_{\mathit{ref}'(\mathcal{C})}(\widetilde{f}(e, e')) = l_{\mathit{ref}'(\mathcal{C})}(e, f_e(e')) = l_{\mathit{ref}'(l_{\mathcal{C}}(e))}(f_e(e')) = l_{\mathit{ref}(l_{\mathcal{C}}(e))}(e') = l_{\mathit{ref}(\mathcal{C})}(e, e')$.

Hence $\widetilde{f}$ is an isomorphism between $(\widetilde{X}, <_{\widetilde{X}}, l_{ref(\mathcal{C})} \restriction \widetilde{X})$ and $(\widetilde{Y}, <_{\widetilde{Y}}, l_{ref'(\mathcal{C})} \restriction \widetilde{Y})$, which had to be shown.

Furthermore, $\widetilde{Y} \in \surd_{\mathit{ref}'(\mathcal{C})}$ iff $\widetilde{X} \in \surd_{\mathit{ref}(\mathcal{C})}$, since $Y_e \in \surd_{\mathit{ref}'(l_{\mathcal{C}}(e))}$ iff $X_e \in \surd_{\mathit{ref}(l_{\mathcal{C}}(e))}$.

"$\supseteq$": by symmetry. ■

Together, Theorems 8.1 and 8.2 say that $\approx_{pt}^{\surd}$ is a congruence for refinement.

9 Branching time partial order semantics

In this section, we discuss several suggestions to define equivalence notions based on partial orders and recording where choices are made. We show that most of these fail in general to be preserved under action refinement. Finally we show that the last and finest notion is indeed invariant under refinement.

9.1 Pomset bisimulation equivalence

In [BC-a] it was suggested to generalise the idea of bisimulation by considering transitions labelled by pomsets. So we consider now transitions $X \xrightarrow{u} X'$ where u is a pomset over Act.

Definition 9.1 Let $\mathcal{C} \in \mathbb{C}_{stable}$.

> $X \xrightarrow{u}_{\mathcal{C}} X'$ iff u is a pomset over Act, $X, X' \in C_{\mathcal{C}}$, $X \subseteq X'$ and $X' - X = H$ with $u = [(H, <_{X'} \cap (H \times H), l_{\mathcal{C}} \upharpoonright H)]_{\cong}$.

Definition 9.2 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$.

> A relation $R \subseteq C_{\mathcal{C}} \times C_{\mathcal{D}}$ is called a *(termination sensitive) pomset bisimulation between $\mathcal{C}$ and $\mathcal{D}$* iff $(\emptyset, \emptyset) \in R$ and if $(X, Y) \in R$ then
>
> - $X \xrightarrow{u}_{\mathcal{C}} X'$, u pomset over $Act \Rightarrow \exists Y'$ with $Y \xrightarrow{u}_{\mathcal{D}} Y'$ and $(X', Y') \in R$,
> - $Y \xrightarrow{u}_{\mathcal{D}} Y'$, u pomset over $Act \Rightarrow \exists X'$ with $X \xrightarrow{u}_{\mathcal{C}} X'$ and $(X', Y') \in R$,
> - $X \in \surd_{\mathcal{C}} \Leftrightarrow Y \in \surd_{\mathcal{D}}$.
>
> $\mathcal{C}$ and $\mathcal{D}$ are *pomset bisimulation equivalent* ($\mathcal{C} \approx_{pb}^{\surd} \mathcal{D}$) iff there exists a pomset bisimulation between $\mathcal{C}$ and $\mathcal{D}$.

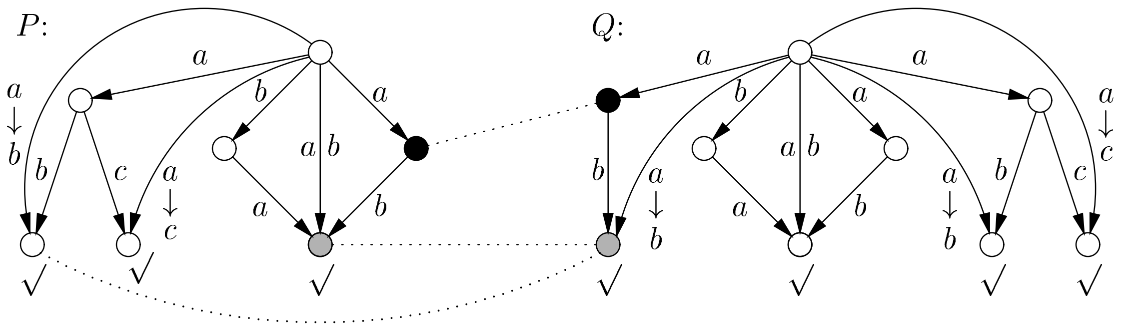
This equivalence notion is clearly finer than both step bisimulation equivalence and pomset trace equivalence: $\mathcal{C} \approx_{pb}^{\surd} \mathcal{D}$ implies $\mathcal{C} \approx_{sb}^{\surd} \mathcal{D}$ and $\mathcal{C} \approx_{pt}^{\surd} \mathcal{D}$; moreover, the processes $a|b$ and $(a|b) + a;b$ considered in Example 7.2 are sb-equivalent but not pb-equivalent; $a;(b + c)$ and $a;b + a;c$ are pomset trace equivalent but not pb-equivalent.

However, pb-equivalence is not preserved under refinement.

Example 9.1

> Consider the systems $P := a;(b+c)+(a|b)$ and $Q := a;(b+c)+(a|b)+a;b$. For convenience in checking that P and Q are pomset bisimulation equivalent, we represent them as *pomset transition systems*. Here the circles represent configurations and the arrows pomset-labelled transitions; terminating configurations are marked with $\surd$.

A pomset bisimulation between P and Q is given by relating each configuration of P to the corresponding configuration of Q (using that P is a subsystem of Q), together with the relation indicated by dotted lines. It is easily checked that this relation satisfies the three clauses of Definition 9.2. Hence $P \approx_{pb}^{\surd} Q$.



However, when refining a into $a_1 \twoheadrightarrow a_2$ and executing a_1, we may arrive in a situation in the second system where a_2 and b may be only executed sequentially and where c is excluded. This is not possible in the first system.

In [GV-a], pomset bisimulation was criticised for violating "the real combination of causality and branching time". The criticism is that only the first system of Example 9.1 has the property that any action a that is causally preceding b is also preceding the choice between b and c. Therefore they suggested a generalised pomset bisimulation equivalence, that is finer then pomset bisimulation equivalence, does not identify the two systems of Example 9.1, and still satisfies $a = a + a$ and the absorption law of Section 6.

However, generalised pomset bisimulation equivalence is also not preserved under refinement.

Example 9.2

$$\mathcal{E} := \begin{array}{l} a \\ \downarrow \\ b \ \# \ b \end{array} \qquad\qquad \mathcal{F} := \begin{array}{l} a \\ \downarrow \\ b \ \# \ b \end{array} \quad + \quad \begin{array}{c} a \\ \downarrow \\ b \end{array}$$

As observed in [GV-a], these two systems are generalised pomset bisimulation equivalent. However, when refining a into $a_1 \twoheadrightarrow a_2$, the resulting systems

$$ref(\mathcal{E}) = \begin{array}{l} a_1 \\ \downarrow \\ a_2 \\ \downarrow \\ b \ \# \ b \end{array} \qquad \text{and} \qquad ref(\mathcal{F}) = \begin{array}{l} a_1 \\ \downarrow \\ a_2 \\ \downarrow \\ b \ \# \ b \end{array} \quad + \quad \begin{array}{c} a_1 \\ \downarrow \\ a_2 \\ \downarrow \\ b \end{array}$$

are not even interleaving bisimulation equivalent. After the action a_1 the action b is always possible in $ref(\mathcal{E})$. However in $ref(\mathcal{F})$ it may be the case that b is impossible after executing a_1 (choosing the branch $a_1 \twoheadrightarrow a_2 \twoheadrightarrow b$).

9.2 History preserving bisimulation

In [DDM-a] another generalisation of bisimulation equivalence was proposed by considering *states* labelled by pomsets. The idea is to relate two states only if they have the same causal history. This so-called *NMS partial order equivalence* was defined on *Nondeterministic Measurement Systems*. We rephrase the definition here in terms of configuration structures as follows.

Definition 9.3 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$.

> A relation $R \subseteq C_\mathcal{C} \times C_\mathcal{D}$ is called a *(termination sensitive) weak history preserving bisimulation between $\mathcal{C}$ and $\mathcal{D}$* iff $(\emptyset, \emptyset) \in R$ and if $(X, Y) \in R$ then
>
> - there is an isomorphism between $(X, <_X, l_\mathcal{C} \upharpoonright X)$ and $(Y, <_Y, l_\mathcal{D} \upharpoonright Y)$,
> - $X \xrightarrow{a}_\mathcal{C} X'$, $a \in Act \Rightarrow \exists Y'$ with $Y \xrightarrow{a}_\mathcal{D} Y'$ and $(X', Y') \in R$,
> - $Y \xrightarrow{a}_\mathcal{D} Y'$, $a \in Act \Rightarrow \exists X'$ with $X \xrightarrow{a}_\mathcal{C} X'$ and $(X', Y') \in R$,
> - $X \in \surd_\mathcal{C} \Leftrightarrow Y \in \surd_\mathcal{D}$.
>
> $\mathcal{C}$ and $\mathcal{D}$ are *weakly history preserving equivalent* ($\mathcal{C} \approx_{wh}^{\surd} \mathcal{D}$) iff there exists a weak history preserving bisimulation between $\mathcal{C}$ and $\mathcal{D}$.

In Definition 9.3 we could equivalently write $X \subseteq X' \in C_\mathcal{C}$ for $X \xrightarrow{a}_\mathcal{C} X'$, $a \in Act$ and $Y \subseteq Y' \in C_\mathcal{D}$ for $Y \xrightarrow{a}_\mathcal{D} Y'$, $a \in Act$ (using Lemma 5.2(5), compare Proposition 9.1); the isomorphism requirement then guarantees that the labels of the events in $X' - X$ and $Y' - Y$ correspond as well.

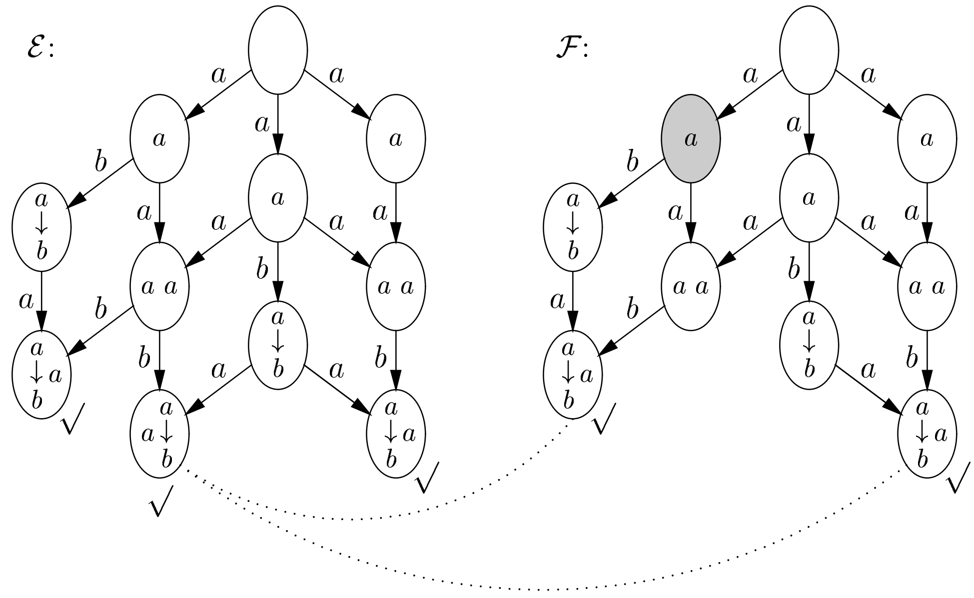
The two systems considered in Example 9.1 are pomset bisimulation equivalent but not weakly history preserving equivalent. The latter follows since any bisimulation between P and Q must relate the black configurations, and hence also the grey ones. However, these induce different pomsets. Nevertheless, wh-equivalence is not finer than pomset bisimulation, as shown by the following example; the two notions are in general incomparable. It will turn out later that wh-equivalence does imply pomset bisimulation for systems without autoconcurrency.

The following example will also show that wh-equivalence is in general not preserved under refinement. This example was suggested to us by Rabinovich. He used it for showing that $\approx_{wh}^{\surd}$ is not a congruence with respect to a TCSP-like parallel composition.

Example 9.3

$$\text{Let } \mathcal{E} := \begin{array}{ccccc} a & \# & a & & a \\ & & \downarrow & & \downarrow \\ & & b & \# & b \end{array} \quad \text{and } \mathcal{F} := \begin{array}{ccccc} a & \# & a & & a \\ & & \downarrow & \# & \downarrow \\ & & b & & b \end{array} .$$

In order to check that $\mathcal{E} \approx_{wh}^{\surd} \mathcal{F}$, consider the representations of $\mathcal{E}$ and $\mathcal{F}$ below. In these pictures every ellipse represents a configuration, inscribed with its associated pomset. Between these configurations the single action relations are displayed. Terminating configurations are marked with $\surd$. A weak history preserving bisimulation between $\mathcal{E}$ and $\mathcal{F}$ is given by relating each configuration of $\mathcal{F}$ to the corresponding configuration of $\mathcal{E}$ (using that $\mathcal{F}$ can be regarded as a subsystem of $\mathcal{E}$), together with the relation indicated by dotted lines. It is easily checked that this relation satisfies the four clauses of Definition 9.3; in particular the first clause is satisfied because related configurations are inscribed by the same pomset.

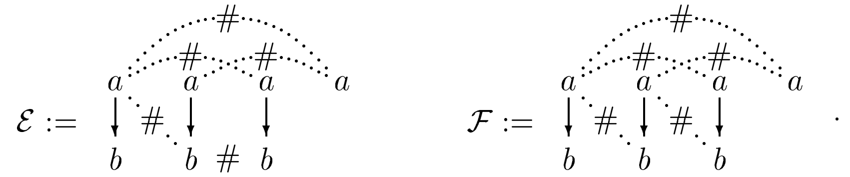


However, $\mathcal{E}$ and $\mathcal{F}$ are not pomset bisimulation equivalent. After executing a, it is alway possible to execute $a \rightarrow b$ in $\mathcal{E}$, whereas in $\mathcal{F}$ it may be impossible to execute $a \rightarrow b$ after a (namely in the grey configuration). When refining a into $a_1 \rightarrow a_2$, the resulting systems are no longer wh-equivalent, not even interleaving bisimulation equivalent. This can be proven by providing a formula in Hennessy-Milner logic [HM] that is satisfied by the refinement of $\mathcal{F}$, but not by the refinement of $\mathcal{E}$. Such a formula is:

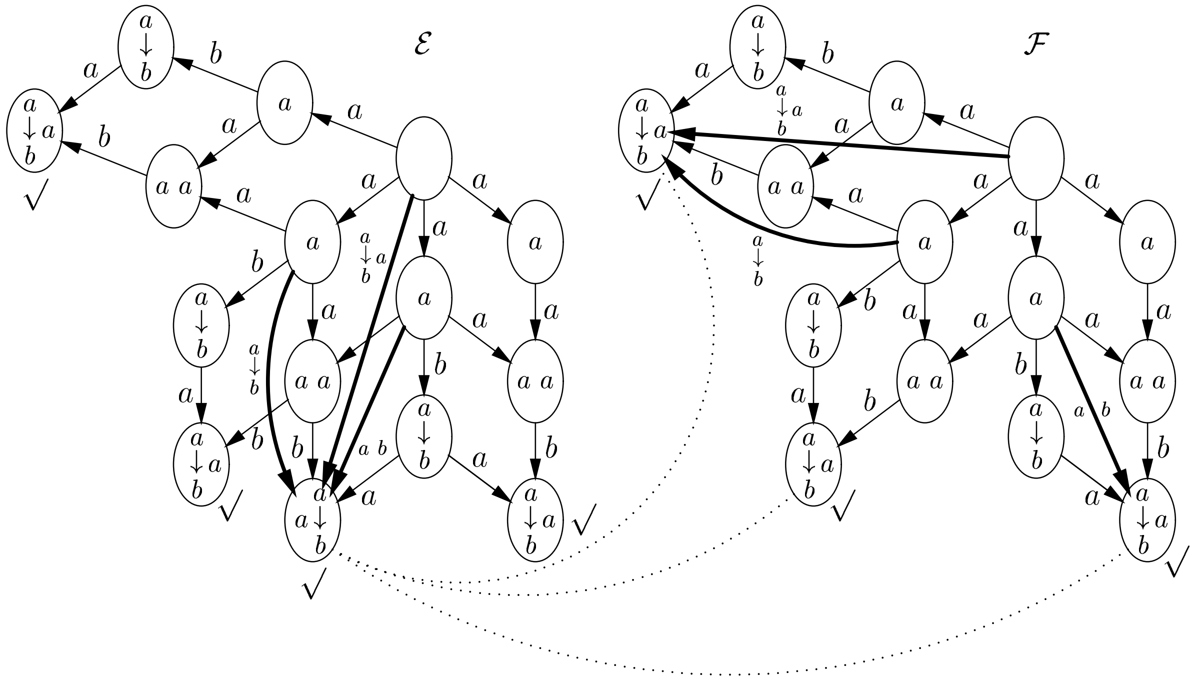
$$\langle a_1 \rangle \langle a_2 \rangle \left(\langle b \rangle T \wedge \langle a_1 \rangle \neg \langle b \rangle T \right).$$

An equivalence finer than both pomset bisimulation and wh-equivalence may be considered by extending the definition of pomset bisimulation with the requirement that, for any $(X, Y) \in R$, $(X, <_X, l_{\mathcal{C}} \restriction X)$ and $(Y, <_Y, l_{\mathcal{C}} \restriction Y)$ should be isomorphic, i.e. by replacing the single action transitions $\xrightarrow{a}$ in Definition 9.3 by pomset transitions $\xrightarrow{u}$. However, the following example shows that also this equivalence would not be preserved under refinement.

Example 9.4



In order to check that $\mathcal{E}$ and $\mathcal{F}$ are equivalent according to the equivalence notion proposed above, we represent these systems and the bisimulation between them following the same conventions as in Example 9.3. However, this time we also have to check that every pomset-labelled transition possible in the one system can be matched by the other. The only three non-trivial instances of this are indicated by the boldface transitions in the figures below.



Again, after refining a into $a_1 \rightarrow a_2$ the systems are not even interleaving bisimulation equivalent. The formula $\langle a_1\rangle\langle a_1\rangle[a_2]\langle b\rangle T$ is satisfied by the refinement of $\mathcal{E}$, but not by the refinement of $\mathcal{F}$.

A finer version of history preserving equivalence has been proposed in [RT] in terms of *behaviour structures.* These are representations of concurrent systems consisting of a labelled transition system in which the states are annotated with pomsets, just like the representations used in Examples 9.3 and 9.4, and additionally the transitions are annotated with embeddings of the pomset associated with its source into a prefix of the pomset associated to its target. [RT] show that a prime event structures can, up to isomorphism, be completely recovered from its representation as a behaviour structure, and vice versa. The *behaviour structure bisimulation* of [RT] is a bisimulation on the underlying transition systems, reflecting the additional structure precisely. Here we reformulate this notion on configuration structures under the name *history preserving bisimulation*. We will show that this equivalence is preserved under refinement. For systems without autoconcurrency, this equivalence coincides with $\approx_{wh}^{\surd}$. This will imply the result that $\approx_{wh}^{\surd}$ is invariant under refinement for systems without autoconcurrency.

Definition 9.4 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$.

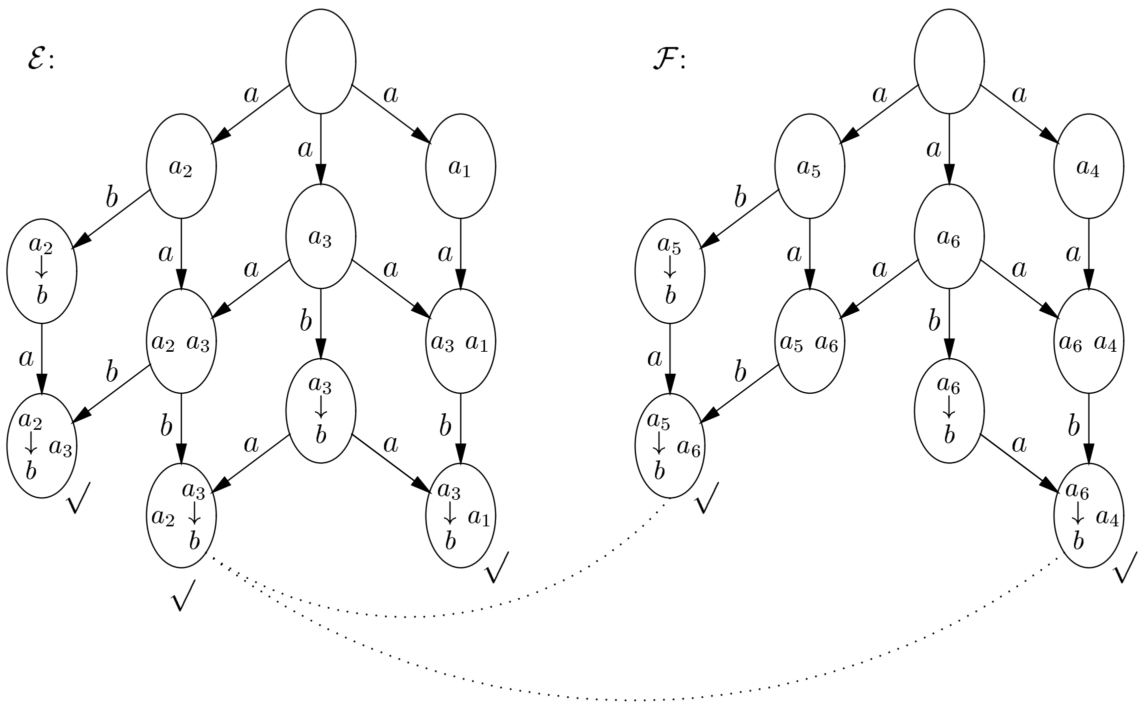
A relation $R \subseteq C_{\mathcal{C}} \times C_{\mathcal{D}} \times \mathcal{P}(E_{\mathcal{C}} \times E_{\mathcal{D}})$ is called a *(termination sensitive) history preserving bisimulation between $\mathcal{C}$ and $\mathcal{D}$* iff $(\emptyset, \emptyset, \emptyset) \in R$ and whenever $(X, Y, f) \in R$ then

- f is an isomorphism between $(X, <_X, l_{\mathcal{C}} \upharpoonright X)$ and $(Y, <_Y, l_{\mathcal{C}} \upharpoonright Y)$,
- $X \xrightarrow{a}_{\mathcal{C}} X'$, $a \in Act \Rightarrow \exists Y', f'$ with $Y \xrightarrow{a}_{\mathcal{D}} Y'$, $(X', Y', f') \in R$ and $f' \upharpoonright X = f$,
- $Y \xrightarrow{a}_{\mathcal{D}} Y'$, $a \in Act \Rightarrow \exists X', f'$ with $X \xrightarrow{a}_{\mathcal{C}} X'$, $(X', Y', f') \in R$ and $f' \upharpoonright X = f$,
- $X \in \surd_{\mathcal{C}} \Leftrightarrow Y \in \surd_{\mathcal{D}}$.

$\mathcal{C}$ and $\mathcal{D}$ are *history preserving equivalent* ($\mathcal{C} \approx_h^{\surd} \mathcal{D}$) iff there exists a history preserving bisimulation between $\mathcal{C}$ and $\mathcal{D}$.

Example 9.3 (continued)

In order to see that $\approx_h^{\surd}$ distinguishes the two system of Example 9.3, it is helpful to indicate the names of the a-events in the simplified behaviour structure representation given earlier. To this end we name the a-events a_1 to a_6, numbered from left to right in the event structure representations of $\mathcal{E}$ and $\mathcal{F}$, respectively.



Let $\mathcal{E}$ start by performing a_2, thereby reaching a state from which it is possible to do a b-transition. This move must be mimicked by $\mathcal{F}$ by performing a_5 or a_6, since a_4 leads to a state from which no immediate b-transition can be done. Suppose $\mathcal{F}$ answers with a_5. Then a supposed history preserving bisimulation between $\mathcal{E}$ and $\mathcal{F}$ relates the configurations $\{a_2\}$ and $\{a_5\}$, and supplies an isomorphism f_1 relating the events a_2 and a_5. Next let $\mathcal{E}$ perform a_3. Then $\mathcal{F}$ can only answer by doing a_6. Thus $\{a_2, a_3\}$ will be related to $\{a_5, a_6\}$, and the isomorphism f_1 is extended to f_2 by additionally relating event a_3 to a_6. Now $\mathcal{E}$ can move to the state $a_2 \begin{smallmatrix} a_3 \\ \downarrow \\ b \end{smallmatrix}$. The best $\mathcal{F}$ can do is to move to the state $\begin{smallmatrix} a_5 \\ \downarrow \\ b \end{smallmatrix} a_6$. But then f_2 must be extended to f_3, relating the two b-events, which is not an isomorphism: we have $a_3 < b$ but not $f(a_3) < f(b)$. A similar contradiction can be obtained if $\mathcal{F}$ tries to simulate a_2 by doing a_6. When $\mathcal{E}$ continues with a_3, $\mathcal{F}$ has a choice between a_5 and a_4. In the first case, the isomorhism will relate a_2 to a_6 and a_3 to a_5. This time it is the other possible b-move of $\mathcal{E}$ that leads to a contradiction. In the latter case, $\mathcal{E}$ can do a b that depends on a_3 but $\mathcal{F}$ cannot do a b that depends on the related event a_4. Hence, there exists no history preserving bisimulation between $\mathcal{E}$ and $\mathcal{F}$.

In the same way one shows that the two systems of Example 9.4 are not history preserving equivalent either.

The following proposition shows that considering more complex transitions in the definition of $\approx_h^{\surd}$ does not change the obtained equivalence notion, and that the label of the transitions is not relevant.

Proposition 9.1

Let $\approx_{sh}^{\surd}$ be the equivalence obtained by replacing the single action transitions $\xrightarrow{a}$ in Definition 9.4 by step transitions $\xrightarrow{A}$, let $\approx_{ph}^{\surd}$ be obtained by replacing them by pomset transitions $\xrightarrow{u}$. Let $\approx_{ih}^{\surd}$ be obtained by replacing $X \xrightarrow{a}_{\mathcal{C}} X'$, $a \in Act$ by $X \subseteq X' \in C_{\mathcal{C}}$, and $Y \xrightarrow{a}_{\mathcal{D}} Y'$, $a \in Act$ by $Y \subseteq Y' \in C_{\mathcal{D}}$ in Definition 9.4.

Then, for all $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$, $\mathcal{C} \approx_h^{\surd} \mathcal{D} \Leftrightarrow \mathcal{C} \approx_{sh}^{\surd} \mathcal{D} \Leftrightarrow \mathcal{C} \approx_{ph}^{\surd} \mathcal{D} \Leftrightarrow \mathcal{C} \approx_{ih}^{\surd} \mathcal{D}$.

Proof As a single action is a special kind of step, and a step a special kind of pomset, we have $\mathcal{C} \approx_{ph}^{\surd} \mathcal{D} \Rightarrow \mathcal{C} \approx_{sh}^{\surd} \mathcal{D} \Rightarrow \mathcal{C} \approx_h^{\surd} \mathcal{D}$. The implication $\mathcal{C} \approx_h^{\surd} \mathcal{D} \Rightarrow \mathcal{C} \approx_{ih}^{\surd} \mathcal{D}$ immediately follows from the observation that, for $X, X' \in C_{\mathcal{C}}$, $\mathcal{C} \in \mathbb{C}_{stable}$, whenever $X \subseteq X'$ there exist configurations $X_0, \ldots, X_n$ and actions $a_1, \ldots, a_n$ ($n \in \mathbb{N}$) such that $X = X_0 \xrightarrow{a_1}_{\mathcal{C}} X_1 \xrightarrow{a_2}_{\mathcal{C}} \ldots \xrightarrow{a_n}_{\mathcal{C}} X_n = X'$ (Lemma 5.2(5)). Finally, the implication $\mathcal{C} \approx_{ih}^{\surd} \mathcal{D} \Rightarrow \mathcal{C} \approx_{ph}^{\surd} \mathcal{D}$ follows from the isomorphism requirements in Definition 9.4: Let R be an ih-bisimulation, and suppose $(X, Y, f) \in R$ and $X \xrightarrow{u} X'$. Then $X \subseteq X'$, thus $\exists Y', f'$ with $Y \subseteq Y', (X', Y', f') \in R$ and $f' \upharpoonright X = f$. Since f' is an isomorphism and $f' \upharpoonright X = f$, $range(f' \upharpoonright (X' - X)) = range(f') - range(f) = Y' - Y$; so $f' \upharpoonright (X' - X)$ is an isomorphism between $X' - X$ and $Y' - Y$. Hence $Y \xrightarrow{u} Y'$, so R satisfies the first clause of a ph-bisimulation. The second clause follows by symmetry. ■

Clearly, we have $\mathcal{C} \approx_h^{\surd} \mathcal{D} \Rightarrow \mathcal{C} \approx_{wh}^{\surd} \mathcal{D}$. However the two systems of Example 9.3 are not h-equivalent. As a corollary of Proposition 9.1 we obtain $\mathcal{C} \approx_h^{\surd} \mathcal{D} \Rightarrow \mathcal{C} \approx_{pb}^{\surd} \mathcal{F}$. That also this implication is strict follows since the two systems of Example 9.1 are not h-equivalent. From Proposition 9.1 we learn that h-bisimulation not only implies pomset bisimulation but even the previous proposal, combining weak history preserving equivalence and pomset bisimulation. Thus $\approx_h^{\surd}$ is the finest equivalence considered so far (except for isomorphism). Nevertheless it is possible to abstract from certain details in a system representation: we have $a \approx_h^{\surd} a + a$ and $(b|(a+c)) + (a|b) + ((b+c)|a) \approx_h^{\surd} (b|(a+c)) + ((b+c)|a)$ (absorption law).

Next we show that $\approx_h^{\surd}$ is a congruence for action refinement; so in particular it is preserved under refinement.

Theorem 9.1 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$ and let *ref*, *ref'* be refinement functions.

If $\mathcal{C} \approx_h^{\surd} \mathcal{D}$ and $ref \approx_h^{\surd} ref'$ then $ref(\mathcal{C}) \approx_h^{\surd} ref'(\mathcal{D})$.

Proof For the sake of simplicity, we use the alternative characterisation $\approx_{ih}^{\surd}$ of $\approx_h^{\surd}$ from Proposition 9.1. So let $R \subseteq C_{\mathcal{C}} \times C_{\mathcal{D}} \times \mathcal{P}(E_{\mathcal{C}} \times E_{\mathcal{D}})$ be an ih-bisimulation between $\mathcal{C}$ and $\mathcal{D}$, and, for $a \in Act$, let R_a be an ih-bisimulation between $ref(a)$ and $ref'(a)$. Define the relation $\widetilde{R}$ by

$\widetilde{R} := \{(\widetilde{X}, \widetilde{Y}, \widetilde{f}) \in C_{ref(\mathcal{C})} \times C_{ref'(\mathcal{D})} \times \mathcal{P}(E_{ref(\mathcal{C})} \times E_{ref'(\mathcal{D})}) \mid \exists (X, Y, f) \in R$ such that

- $\widetilde{X}$ is a refinement of X by *ref* — $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ with $X_e \neq \emptyset$,
- $\widetilde{Y}$ is a refinement of Y by *ref'* — $\widetilde{Y} = \bigcup_{d \in Y} \{d\} \times Y_d$ with $Y_d \neq \emptyset$,
- $\forall e \in X,\ \exists f_e : X_e \to Y_{f(e)}$ such that $(X_e, Y_{f(e)}, f_e) \in R_{l_{\mathcal{C}}(e)}$
- and $\widetilde{f} : \widetilde{X} \to \widetilde{Y}$ is the bijection satisfying $\widetilde{f}(e, e') = (f(e), f_e(e'))\}$.

We show that $\widetilde{R}$ is an ih-bisimulation between $ref(\mathcal{C})$ and $ref'(\mathcal{D})$.

i. $(\emptyset, \emptyset, \emptyset) \in \widetilde{R}$ since $(\emptyset, \emptyset, \emptyset) \in R$.

ii. Suppose $(\widetilde{X}, \widetilde{Y}, \widetilde{f}) \in \widetilde{R}$. Take $(X, Y, f) \in R$ such that

- $\widetilde{X}$ is a refinement of X by *ref* — $\widetilde{X} = \bigcup_{e \in X} \{e\} \times X_e$ with $X_e \neq \emptyset$,
- $\widetilde{Y}$ is a refinement of Y by *ref'* — $\widetilde{Y} = \bigcup_{d \in Y} \{d\} \times Y_d$ with $Y_d \neq \emptyset$,
- $\forall e \in X,\ \exists f_e : X_e \to Y_{f(e)}$ such that $(X_e, Y_{f(e)}, f_e) \in R_{l_{\mathcal{C}}(e)}$
- and $\widetilde{f} : \widetilde{X} \to \widetilde{Y}$ is the bijection satisfying $\widetilde{f}(e, e') = (f(e), f_e(e'))$.

Now four things have to be established:

1. $\tilde{f}$ satisfies $(d,d') <_{\widetilde{X}} (e,e') \Leftrightarrow \tilde{f}(d,d') <_{\widetilde{Y}} \tilde{f}(e,e')$ and $l_{ref'(\mathcal{D})}(\tilde{f}(e,e')) = l_{ref(\mathcal{C})}(e,e')$.
2. $\widetilde{X} \subseteq \widetilde{X'} \in C_{ref(\mathcal{C})} \Rightarrow \exists \widetilde{Y'}, \tilde{f}'$ such that $\widetilde{Y} \subseteq \widetilde{Y'} \in C_{ref'(\mathcal{D})}$, $\tilde{f}' \restriction \widetilde{X} = \tilde{f}$ and $(\widetilde{X'}, \widetilde{Y'}, \tilde{f}') \in \tilde{R}$.
3. $\widetilde{Y} \subseteq \widetilde{Y'} \in C_{ref'(\mathcal{D})} \Rightarrow \exists \widetilde{X'}, \tilde{f}'$ such that $\widetilde{X} \subseteq \widetilde{X'} \in C_{ref(\mathcal{C})}$, $\tilde{f}' \restriction \widetilde{X} = \tilde{f}$ and $(\widetilde{X'}, \widetilde{Y'}, \tilde{f}') \in \tilde{R}$.
4. $\widetilde{X} \in \surd_{ref(\mathcal{C})} \Leftrightarrow \widetilde{Y} \in \surd_{ref'(\mathcal{D})}$.

ad 1. Straightforward with Proposition 5.7 (cf. the proof of Theorem 8.2).

ad 2. Suppose $\widetilde{X} \subseteq \widetilde{X'} \in C_{ref(\mathcal{C})}$.

Let $\widetilde{X'} = \bigcup_{e \in X'} \{e\} \times X'_e$ where $X' \in C_{\mathcal{C}}$ and $\forall e \in X' : X'_e \in C_{ref(l_{\mathcal{C}}(e))} - \{\emptyset\}$.
Since $\widetilde{X} \subseteq \widetilde{X'}$ we have $X \subseteq X'$. As R is an ih-bisimulation, $\exists Y', f'$ with $Y \subseteq Y' \in C_{\mathcal{D}}$, $f' \restriction X = f$ and $(X', Y', f') \in R$.

Furthermore, for $e \in X'$, we have $X_e \subseteq X'_e \in C_{ref(l_{\mathcal{C}}(e))}$ (taking $X_e := \emptyset$ if $e \in X' - X$). Since $R_{l_{\mathcal{C}}(e)}$ is an ih-bisimulation, taking $Y_{f'(e)} := \emptyset$ if $e \in X' - X$, $\exists Y'_{f'(e)}, f'_e$ with $Y_{f'(e)} \subseteq Y'_{f'(e)} \in C_{ref'(l_{\mathcal{C}}(e))}$, $f'_e \restriction X_e = f_e$ and $(X'_e, Y'_{f'(e)}, f'_e) \in R_{l_{\mathcal{C}}(e)}$.

Let $\widetilde{Y'} := \{(f'(e), f'_e(e')) | (e,e') \in \widetilde{X'}\}$ and $\tilde{f}' := \{((e,e'), (f'(e), f'_e(e'))) | (e,e') \in \widetilde{X'}\}$.
It now suffices to show that $\widetilde{Y'}$ is a refinement of Y' by ref', since then it follows immediately with Definition 5.4 that $\widetilde{Y} \subseteq \widetilde{Y'} \in C_{ref'(\mathcal{D})}$ (using that $f' \restriction X = f$ and, for $e \in X'$, $f'_e \restriction X_e = f_e$), $\tilde{f}' \restriction X = \tilde{f}$ (likewise) and $(\widetilde{X'}, \widetilde{Y'}, \tilde{f}') \in \tilde{R}$.

- By construction $\widetilde{Y'} = \bigcup_{e \in X'} \{f'(e)\} \times f'_e(X'_e) = \bigcup_{d \in Y'} \{d\} \times Y'_d$ where $\forall d \in Y' : Y'_d \in C_{ref(l_{\mathcal{D}}(d))} - \{\emptyset\}$.
- $d \in busy(\widetilde{Y'}) = \{d \in Y' | Y'_d \notin \surd_{ref'(l_{\mathcal{D}}(d))}\}$ iff $f'^{-1}(d) \in busy(\widetilde{X'}) = \{e \in X' | X'_e \notin \surd_{ref(l_{\mathcal{C}}(e))}\}$, since $(X'_e, Y'_{f'(e)}, f'_e) \in R_{l_{\mathcal{C}}(e)}$.
  Furthermore, d maximal in Y' iff $f'^{-1}(d)$ maximal in X', since f' is an isomorphism.
  Hence $d \in busy(\widetilde{Y'})$ implies d maximal in Y', since $\widetilde{X'}$ is a refinement of X'.

From this it follows that $\widetilde{Y'}$ is a refinement of Y'.

ad 3. By symmetry.

ad 4. Straightforward with Definition 5.4. ■

Termination insensitive history preserving equivalence ($\approx_h$) is defined as $\approx_h^{\surd}$, but without the last clause. Clearly $\mathcal{C} \approx_h^{\surd} \mathcal{D} \Rightarrow \mathcal{C} \approx_h \mathcal{D}$, and on prime event structures the two equivalences coincide.

Proposition 9.2 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$ and let ref be a refinement function.
If $\mathcal{C} \approx_h \mathcal{D}$ then $ref(\mathcal{C}) \approx_h ref(\mathcal{D})$.

Proof Trivially, $ref \approx_h^{\surd} ref$. Now the proof of Theorem 9.1 applies, omitting item 4. ■

However, again using Example 8.1, $\approx_h$ is not a congruence for refinement.

Finally we show that $\approx_{wh}^{\surd}$ and $\approx_{h}^{\surd}$ coincide for stable configuration structures where concurrent events may not carry the same label. As a corollary we then have that in this case also $\approx_{wh}^{\surd}$ implies pomset bisimulation and is preserved under refinement.

Definition 9.5 $\mathcal{C} \in \mathbb{C}_{stable}$ is *without autoconcurrency* iff

$\forall X \in C_{\mathcal{C}},\ \forall d, e \in X : (d\ co_X\ e \text{ and } l(d) = l(e)) \Rightarrow d = e.$

Theorem 9.2 Let $\mathcal{C},\ \mathcal{D} \in \mathbb{C}_{stable}$ be without autoconcurrency.

Then $\mathcal{C} \approx_{wh}^{\surd} \mathcal{D} \Leftrightarrow \mathcal{C} \approx_{h}^{\surd} \mathcal{F}$.

Proof First note that a wh-bisimulation can be regarded as a h-bisimulation without the requirements that $f' \restriction X = f$. The isomorphisms that are required to exist anyway are then included as a third component in the bisimulation.

Now "$\Leftarrow$" is trivial.
In order to establish "$\Rightarrow$" we first make two observations.

1. Let $\mathcal{C} \in \mathbb{C}_{stable},\ X, X' \in C_{\mathcal{C}}, X \subseteq X'$ and $e \in X$. Then by Propositions 5.3 and 5.4 we have
$$(d \in X' \wedge d <_{X'} e) \Leftrightarrow (d \in X \wedge d <_X e)\ .$$
2. If g is an isomorphism between two labelled partial orders $(X, <_X, l_X)$ and $(Y, <_Y, l_Y)$, and $e \in X$ then
$$|\{d \in X \mid d <_X e\}| \ = \ |\{d \in Y \mid d <_Y g(e)\}|\,.$$

Now suppose $\mathcal{C} \approx_{wh}^{\surd} \mathcal{D}$. Let R be a h-bisimulation between $\mathcal{C}$ and $\mathcal{D}$ without the requirements $f' \restriction X = f$. We proof that these requirements are met nevertheless. Assume that $(X, Y, f) \in R$ and $X \xrightarrow{a}_{\mathcal{C}} X'$. Then there exists $(X', Y', f') \in R$ with $Y \xrightarrow{a}_{\mathcal{D}} Y'$. Suppose $f' \restriction X \neq f$. Then there exists an $e \in X$ with $f'(e) \neq f(e)$. With the observations above it follows that

$$|\{d \in Y' \mid d <_{Y'} f(e)\}| \overset{1}{=} |\{d \in Y \mid d <_Y f(e)\}| \overset{2}{=}$$
$$|\{d \in X \mid d <_X e\}| \overset{1}{=} |\{d \in X' \mid d <_{X'} e\}| \overset{2}{=} |\{d \in Y' \mid d <_{Y'} f'(e)\}|.$$

Hence we cannot have $f'(e) <_{Y'} f(e)$ or $f(e) <_{Y'} f'(e)$. Thus $f'(e)\ co_{Y'}\ f(e)$.
Moreover, f' and f preserve labelling, so $l_{\mathcal{D}}(f'(e)) = l_{\mathcal{C}}(e) = l_{\mathcal{D}}(f(e))$.
This is a contradiction since $\mathcal{D}$ was assumed to have no autoconcurrency. ■

Corollary 9.1 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$ be without autoconcurrency.

(i) If $\mathcal{C} \approx_{wh}^{\surd} \mathcal{D}$ then $\mathcal{C} \approx_{pb}^{\surd} \mathcal{D}$.

(ii) Let *ref* and *ref'* be refinement functions.
If $\mathcal{C} \approx_{wh}^{\surd} \mathcal{D}$ and $ref \approx_{wh}^{\surd} ref'$ then $ref(\mathcal{C}) \approx_{wh}^{\surd} ref'(\mathcal{D})$. ■

9.3 Hereditary history preserving bisimulation

The following strengthening of $\approx_h^{\surd}$ was proposed in [Bednarczyk].

Definition 9.6 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$.

A history preserving bisimulation R between $\mathcal{C}$ and $\mathcal{D}$ is *hereditary* iff it satisfies

– $(X, Y, f) \in R \wedge X' \xrightarrow{a}_{\mathcal{C}} X$, $a \in Act \Rightarrow (X', f(X'), f \restriction X') \in R$.

$\mathcal{C}$ and $\mathcal{D}$ are *hereditary history preserving equivalent* ($\mathcal{C} \approx_{hh}^{\surd} \mathcal{D}$) iff there exists a hereditary history preserving bisimulation between $\mathcal{C}$ and $\mathcal{D}$.

To be precise, [Bednarczyk] calls our history preserving equivalence *strong* history preserving bisimulation equivalence, and hence calls $\approx_{hh}$ *hereditary strong history preserving bisimulation equivalence*. Moreover, he studies the termination insensitive variant only. Note that the clause above is equivalent to

$$(X, Y, f) \in R \wedge X' \xrightarrow{a}_{\mathcal{C}} X \Rightarrow \exists Y', f' \text{ with } Y' \xrightarrow{a}_{\mathcal{D}} Y,\ (X', Y', f') \in R \text{ and } f \restriction X' = f'.$$

i.e. the same condition as in Definition 9.4, but going "backwards" along the a-transition. This clause is also equivalent to its symmetric counterpart

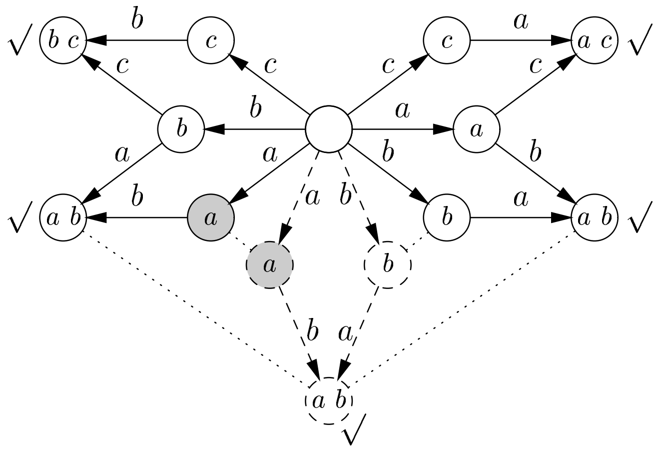
$$(X, Y, f) \in R \wedge Y' \xrightarrow{a}_{\mathcal{D}} Y \Rightarrow \exists X', f' \text{ with } X' \xrightarrow{a}_{\mathcal{C}} X,\ (X', Y', f') \in R \text{ and } f \restriction X' = f'.$$

Finally, as in Proposition 9.1, the property is invariant under replacement of the single action transitions $\xrightarrow{a}$ by step or pomset transitions, or by inclusion of configurations.

By definition $\mathcal{C} \approx_{hh}^{\surd} \mathcal{D} \Rightarrow \mathcal{C} \approx_h^{\surd} \mathcal{D}$. This implication is strict, because $\approx_{hh}^{\surd}$ does not satisfy the absorption law mentioned at the end of Section 6:

Example 9.6

Below the system $(b|(a+c)) + (a|b) + ((b+c)|a)$ is represented as a simplified behaviour structure, following the conventions of Example 9.3. The related system $(b|(a+c)) + ((b+c)|a)$ is represented by exactly the same structure, but without the dashed parts.



A bisimulation between the two systems consists of the identity relation (using that one system can be seen as a subsystem of the other), together with the relation indicated by dotted lines. This bisimulation is surely history preserving; as the systems are without autoconcurrency it suffices to check that related configurations are inscribed with the same pomset. Note that the a from $a|b$ (grey) can only be matched by the a from $b|(a+c)$ (also grey); the a from $(b+c)|a$ leads to a state where c still is possible. The b from $a|b$ on the other hands needs to be matched by the b from $(b+c)|a$. The underlying idea is that whenever even the smallest part of the process $a|b$ happens, either a small part of the a-event, or a small part of the b-event must have happened (or both). In the first case, executing an equally small part of the a-event in $a|(b+c)$ already rules out the possibility of ever executing c, and the futures of the related processes are the same.

In order to see that the two systems are not hereditary history preserving equivalent, execute the a from $a|b$ followed by the b, thus reaching the configuration $a\,b$; from there go back to the b-configuration along the a-transition. This can only be matched in the other system by travelling to the grey a-configuration, doing a b from there, and backtracking to the b-configuration. The latter allows a c-move, however, which is not possible in the original system.

The following theorem says that also $\approx_{hh}^{\surd}$ is a congruence for action refinement.

Theorem 9.3 [Bednarczyk] Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$ and let *ref*, *ref'* be refinement functions.

If $\mathcal{C} \approx_{hh}^{\surd} \mathcal{D}$ and $ref \approx_{hh}^{\surd} ref'$ then $ref(\mathcal{C}) \approx_{hh}^{\surd} ref'(\mathcal{D})$.

Proof Suppose the bisimulations R and R_a mentioned in the proof of Theorem 9.1 are hereditary. Then the bisimulation $\widetilde{R}$ constructed in that proof is hereditary as well. ■

[Bednarczyk] establishes that any hereditary history preserving bisimulation between two prime event structures with binary conflict $\mathcal{E}$ and $\mathcal{F}$ can itself be regarded as a prime event structure, which is hereditary history preserving equivalent to $\mathcal{E}$ and $\mathcal{F}$. This result applies to stable configuration structures as well, as we will show below. Moreover, a hereditary history preserving bisimulation between two stable configuration structures can not only be regarded as a stable configuration structure itself, but even as a prime event structure in the sense of [Winskel], upgraded with a termination predicate.

Definition 9.7 Let R be a hh-bisimulation between $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$.

Then $\mathcal{E}_R := (E_R, \leq_R, Con_R, l_R)$ is the prime event structure given by

- $E_R := \{(X, Y, f) \in R \mid X$ has a greatest element $max(X)$ w.r.t. $<_X\}$,
- $(X', Y', f') \leq_R (X, Y, f)$ iff $X' \subseteq X \wedge Y' \subseteq Y \wedge f' \subseteq f$,
- a set of events $\{(X_i, Y_i, f_i) \mid i \in I\}$ is *consistent* iff $(\bigcup_{i\in I} X_i, \bigcup_{i\in I} Y_i, \bigcup_{i\in I} f_i) \in R$, notation $Con(\{(X_i, Y_i, f_i) \mid i \in I\})$,
- $l_R(X, Y, f) = l_{\mathcal{C}}(max(X))$.

Moreover, a consistent set of events $\{(X_i, Y_i, f_i) \mid i \in I\}$ is *terminated* iff $\surd(\bigcup_{i\in I} X_i)$.

Note that, for $(X,Y,f) \in R$, X has a greatest element $max(X)$ w.r.t. $<_X$ iff Y has a greatest element $max(Y)$ w.r.t. $<_Y$. Moreover, $f(max(X)) = max(Y)$ and hence $l_\mathcal{C}(max(X)) = l_\mathcal{D}(max(Y))$. Finally, if $\{(X_i, Y_i, f_i) \mid i \in I\}$ is a consistent set of events, then $\surd(\bigcup_{i\in I} X_i) \Leftrightarrow \surd(\bigcup_{i\in I} Y_i)$. Thus the definition is symmetric in $\mathcal{C}$ and $\mathcal{D}$.

Proposition 9.3 Let R be a hh-bisimulation between $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$.
Then $\mathcal{E}_R$ is a prime event structure in the sense of [Winskel].

Proof It is easy to check that $\mathcal{E}_R$ satisfies the four properties required in [Winskel]. ■

However, $\mathcal{E}_R$ is not a prime event structure with binary conflict as in Definition 2.1.

Definition 9.8
The (finite) configurations of a prime event structure in the sense of [Winskel] are those finite sets of events that are left-closed and consistent. Now define the *configuration structure associated to a hereditary history preserving bisimulation* R as $\mathcal{C}_R := (C_R, \surd_R, l_R)$, where C_R is the set of configurations of $\mathcal{E}_R$, and $\surd_R$ the set of those configurations that are terminated.

Definition 9.9 Let R be a hh-bisimulation between $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$.
Let $\pi_1 : E_R \to E_\mathcal{C}$ be given by $\pi_1(X,Y,f) := max(X)$ and likewise $\pi_2(X,Y,f) := max(Y)$. For $(X,Y,f) \in R$ write $C(X,Y,f) := \{(\downarrow e, \downarrow f(e), f \restriction \downarrow e) \mid e \in X\}$, in which $\downarrow e := \{d \in X \mid d \leq_X e\}$.

By Proposition 5.3, $\downarrow e \in C_\mathcal{C}$ for $e \in X \in C_\mathcal{C}$. So, as R is hereditary, every element of $C(X,Y,f)$ is an event in E_R. Moreover, $(\bigcup_{e\in X} \downarrow e, \bigcup_{e\in X} \downarrow f(e), \bigcup_{e\in X} f \restriction \downarrow e) = (X,Y,f)$, and hence $C(X,Y,f) \in C_R$. It is easy to see that every configuration in C_R is of this form, i.e.

$$C_R = \{C(X,Y,f) \mid (X,Y,f) \in R\}.$$

So C establishes a bijective correspondence between R and C_R. Furthermore, for every $(X,Y,f) \in R$, $\pi_1 \restriction C(X,Y,f)$ is a bijection between $C(X,Y,f)$ and X preserving $<$ and l:

$$(\downarrow d, \downarrow f(d), f \restriction \downarrow d) <_R (\downarrow e, \downarrow f(e), f \restriction \downarrow e) \Leftrightarrow d <_X e \text{ and } l_R(\downarrow e, \downarrow f(e), f \restriction \downarrow e) = l_\mathcal{C}(e).$$

Now the following is the (straightforward) generalisation of Theorem 5.6 of [Bednarczyk] to stable configuration structures.

Proposition 9.4 Let R be a hh-bisimulation between $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$.
Then $\mathcal{C}_R \approx_{hh}^{\surd} \mathcal{C} \approx_{hh}^{\surd} \mathcal{D}$.

Proof $\{(C(X,Y,f), X, \pi_1 \restriction C(X,Y,f)) \mid (X,Y,f) \in R\}$ is a hereditary history preserving bisimulation between $\mathcal{C}_R$ and $\mathcal{C}$, as is straightforward to check. ■

Corollary 9.1 It follows that every stable configuration structure is $\approx_{hh}^{\surd}$-equivalent to a prime event structure as in [Winskel] upgraded with a predicate for successful termination.

Hereditary history preserving bisimulation from open maps

In [JNW] a general definition of bisimilarity is given that applies to any category $\mathbf{M}$ with subcategory $\mathbf{P}$. The idea is that the objects in $\mathbf{M}$ represent concurrent systems and the morphisms simulations of one system by another. The *path category* $\mathbf{P}$ is supposed to contain the systems that have only one (finite) run. The purpose of $\mathbf{P}$ is to formalise the notion of a *(partial) run* or *computation path* of a system $\mathcal{A} \in \mathbf{M}$. This is defined as a morphism p from a one-run system $P \in \mathbf{P}$ to $\mathcal{A}$. In order to define bisimilarity, [JNW] define the following concept of an *open morphism*:

Definition 9.10 Let $\mathbf{M}$ be a category with subcategory $\mathbf{P}$.

> A morphism (or *map*) $f : \mathcal{A} \to \mathcal{B}$ in $\mathbf{M}$ is $\mathbf{P}$-*open* if for every $P, Q \in \mathbf{P}$ and morphisms $p : P \to \mathcal{A}$, $m : P \to Q$ and $q : Q \to \mathcal{B}$ such that $f \circ p = q \circ m$, there exists a morphism $p' : Q \to \mathcal{A}$ such that $p' \circ m = p$ and $f \circ p' = q$. As a diagram:
>
> $$\begin{tikzcd} & & \mathcal{A} \arrow[dd, "f"] \\ P \arrow[urr, "p"] \arrow[r, "m"] \arrow[drr, "{f \circ p \atop = q \circ m}"'] & Q \arrow[dr, "q"] & \\ & & \mathcal{B} \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} & & \mathcal{A} \arrow[dd, "f"] \\ P \arrow[urr, "p"] \arrow[r, "m"] \arrow[drr, "{f \circ p \atop = q \circ m}"'] & Q \arrow[ur, "p'"] \arrow[dr, "q"] & \\ & & \mathcal{B} \end{tikzcd} .$$

In this definition p is a computation path of the system $\mathcal{A}$ and hence $f \circ p$ is one of $\mathcal{B}$. These paths are *matched* by f. The latter path embeds in a computation path q of $\mathcal{B}$. The definition of an open map requires that a computation path p' can be found in $\mathcal{A}$, matching q and suitably containing p. Thus an open map induces a kind of bisimulation between $\mathcal{A}$ and $\mathcal{B}$. In general however, not every bisimulation between two systems can be realised as an open map, for open morphisms yield only *functional* bisimulations, in which a run of the first system is related to only one run of the second. Therefore [JNW] defines $\mathbf{P}$-*bisimilarity* as follows:

Definition 9.11 Let $\mathbf{M}$ be a category with subcategory $\mathbf{P}$.

> Two objects $\mathcal{A}, \mathcal{B} \in \mathbf{M}$ are $\mathbf{P}$-*bisimilar* if there is a *span of open maps* $f : \mathcal{C} \to \mathcal{A}$, $g : \mathcal{C} \to \mathcal{B}$ for some $\mathcal{C} \in \mathbf{M}$.

They show that if this definition is applied to the category of transition systems with functional simulations, taking $\mathbf{P}$ to be the full subcategory of finite one-branch trees, then $\mathbf{P}$-bisimilarity coincides with the classical bisimulation equivalence of [Milner-b]. Moreover, when applied to the category of prime event structures [Winskel], taking $\mathbf{P}$ to be the full subcategory of finite pomsets, $\mathbf{P}$-bisimilarity turns out to be $\approx_{hh}$.

Below, we generalise the latter result to stable configuration structures, obtaining $\approx_{hh}^{\surd}$ as the resulting notion of bisimilarity.

Definition 9.12 Let $\mathcal{C}, \mathcal{D}$ be configuration structures.

A *morphism* from $\mathcal{C}$ to $\mathcal{D}$ is a map $f : E_{\mathcal{C}} \to E_{\mathcal{D}}$ such that

- $l_{\mathcal{D}}(f(e)) = l_{\mathcal{C}}(e)$ for all $e \in E_{\mathcal{C}}$,
- $f \upharpoonright X$ is injective and $f(X) \in C_{\mathcal{D}}$ for any configuration $X \in C_{\mathcal{C}}$.
- $f(X) \in \surd_{\mathcal{D}}$ for any terminating configuration $X \in \surd_{\mathcal{C}}$.

This is the standard notion of a synchronous morphism on stable families of configurations from [Winskel], upgraded to reflect labelling and the termination predicate. It makes $\mathbb{C}$ into a category $\mathbf{C}$, and $\mathbb{C}_{stable}$ into a category $\mathbf{C}_{stable}$. As stable path category we take the full subcategory **Pom** of $\mathbf{C}_{stable}$ consisting of those configuration structures with a largest configuration. These structures arise as the configuration structures associated to finite, conflict-free prime event structures (pomsets), except that the unique maximal configuration need not be terminating. A morphism $p : P \to \mathcal{C}$ with $E_P \in \surd_P$ (for $P \in \mathbf{Pom}$) represents a complete run, rather than a partial one.

Definition 9.13 Let $\mathcal{C} \in \mathbb{C}$ and $X \in C_{\mathcal{C}}$.

The *restriction of* $\mathcal{C}$ *to* X is given by
- $C_{\mathcal{C} \upharpoonright X} := \{Z \in C_{\mathcal{C}} \mid Z \subseteq X\}$,
- $\surd_{\mathcal{C} \upharpoonright X} := \{Z \in \surd_{\mathcal{C}} \mid Z \subseteq X\}$,
- $l_{\mathcal{C} \upharpoonright X} := l_{\mathcal{C}} \upharpoonright X$.

Note that $\mathcal{C} \upharpoonright X \in \mathbf{Pom}$ for all $\mathcal{C} \in \mathbb{C}_{stable}$ and all $X \in C_{\mathcal{C}}$. In fact, **Pom** consists exactly of those stable configuration structures that are of the form $\mathcal{C} \upharpoonright X$.

Proposition 9.5 Let $\mathcal{C}, \mathcal{D}$ be stable configuration structures.

A morphism f from $\mathcal{C}$ to $\mathcal{D}$ is **Pom**-*open* iff for all $X \in C_{\mathcal{C}}$ it satisfies

$$d <_X e \Leftrightarrow f(d) <_{f(X)} f(e)$$

$$f(X) \subseteq Y \in C_{\mathcal{D}} \Rightarrow \exists X' \in C_{\mathcal{C}} : X \subseteq X' \wedge f(X') = Y$$

$$f(X) \in \surd_{\mathcal{D}} \Rightarrow X \in \surd_{\mathcal{C}}.$$

Proof "if": Suppose $f : E_{\mathcal{C}} \to E_{\mathcal{D}}$ is a morphism that satisfies the three conditions above. Let $p : P \to \mathcal{C}$ and $q : Q \to \mathcal{D}$ be runs and $m : P \to Q$ a morphism with $q \circ m = f \circ p$. Write $X := p(E_P)$ and $Y := q(E_Q)$. As $m(E_P) \subseteq E_Q$ we have $f(X) \subseteq Y$. So there is configuration $X' \in C_{\mathcal{C}}$ with $X \subseteq X'$ and $f(X') = Y$. Define $p' : Q \to \mathcal{C}$ by $p'(e) := (f \upharpoonright X')^{-1}(q(e))$. That $f \circ p' = q$ is immediate. Let $e \in E_P$. Then $p'(m(e)) = (f \upharpoonright X')^{-1}(q(m(e)) = (f \upharpoonright X')^{-1}(f(p(e)) = p(e)$, so $p' \circ m = p$. It remains to check that p' is a morphism. First of all, $l_{\mathcal{C}}(p'(e)) = l_{\mathcal{D}}(q(e)) = l_Q(e)$ for all $e \in E_Q$. As Q has a largest configuration, q is injective, and so is p'. Next, let $Z \in C_Q$. Then $q(Z) \in C_{\mathcal{D}}$. Hence, by Proposition 5.3, $q(Z)$ is left-closed w.r.t. $<_Y$, so $p'(Z)$ must be left-closed w.r.t. $<_{X'}$. Again using Proposition 5.3, it follows that $p'(Z) \in C_{\mathcal{C}}$. Finally, suppose $Z \in \surd_Q$. Then $f(p'(Z)) = q(Z) \in \surd_{\mathcal{D}}$, and hence $p'(Z) \in \surd_{\mathcal{C}}$.

"only if": Suppose $d \not<_X e$. By Definition 5.6 this means that there is a configuration $Y \subset X$ of $\mathcal{C}$ with $e \in Y \not\ni d$. By Definition 9.12 $f(Y) \subset f(X)$ is a configuration of $\mathcal{D}$ with $f(e) \in f(Y) \not\ni f(d)$. Hence $f(d) \not<_{f(Y)} f(e)$. For this direction we did not need openness.

Now suppose f is a **Pom**-open morphism from $\mathcal{C}$ to $\mathcal{D}$ and $X \in C_{\mathcal{C}}$. Then apply Definition 9.10 with $p : \mathcal{C}\upharpoonright X \to \mathcal{C}$ the identity function on X, $q : \mathcal{D}\upharpoonright f(X) \to \mathcal{D}$ the identity function on $f(X)$, and $m := f\upharpoonright X$. Clearly $q \circ m = m = f \circ p$. It follows that there exists a morphism p' from $\mathcal{D}\upharpoonright f(X)$ to $\mathcal{C}$ mapping $f(X)$ to X. Using this morphism, the implication $f(d) \not<_{f(X)} f(e) \Rightarrow d \not<_X e$ follows exactly as the reverse direction above. It also follows that if $f(X) \in \surd_{\mathcal{D}}$ then $X \in \surd_{\mathcal{C}}$.

Finally, suppose that $f(X) \subseteq Y \in C_{\mathcal{D}}$. Apply Definition 9.10 with $p : \mathcal{C}\upharpoonright X \to \mathcal{C}$ the identity function on X, $q : \mathcal{D}\upharpoonright Y \to \mathcal{D}$ the identity function on Y, and $m := f\upharpoonright X$. It follows that there exists a morphism p' from $\mathcal{D}\upharpoonright Y$ to $\mathcal{C}$ such that $p' \circ m = p$ and $f \circ p' = q$. Let $X' := p'(Y)$. As $p' \circ m = p$ we have $X = p(X) = p'(m(X)) = p'(f(X)) \subseteq p'(Y) = X'$, and as $f \circ p' = q$ we have $f(X') = f(p'(Y)) = q(Y) = Y$. ■

Proposition 9.6 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$ and f a **Pom**-open morphism from $\mathcal{C}$ to $\mathcal{D}$.

Then $\{(X, f(X), f\upharpoonright X) \mid X \in C_{\mathcal{C}}\}$ is a hereditary history preserving bisimulation.

Proof We check the conditions of Definition 9.4. First of all $\emptyset \in C_{\mathcal{C}}$ and $f(\emptyset) = \emptyset$. Now take $X \in C_{\mathcal{C}}$. That $f\upharpoonright X : X \to f(X)$ is a labelling preserving bijection is given by Definition 9.12. Proposition 9.5 implies that it is even an isomorphism. The second clause in Definition 9.4 follows easily from Definition 9.12 by taking $Y' := f(X')$, and the third from Proposition 9.5 (using also Proposition 9.1). The last clause follows in one direction from Definition 9.12 and in the other from Proposition 9.5.

That this bisimulation is hereditary follows immediately from its construction. ■

Corollary 9.2 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$

A function $f : E_{\mathcal{C}} \to E_{\mathcal{D}}$ is a **Pom**-open map iff $\{(X, f(X), f\upharpoonright X) \mid X \in C_{\mathcal{C}}\}$ is a (hereditary) history preserving bisimulation.

Proof "Only if" is Proposition 9.6; "if" follows directly from Proposition 9.5 and Definitions 9.4 and 9.12. ■

By definition, an open map preserves the internal structure of stable configuration structures to a very large extent. It essentially does nothing else then collapsing certain events that are pairwise in conflict, in the sense that they can not occur in the same configuration. This may also cause different configurations to be identified, but only if the possible extensions of these configurations are identified as well.

The following result characterises hereditary history preserving equivalence in terms of **Pom**-open maps. In generalises a similar result of [JNW] to stable configuration structures with termination predicates.

Theorem 9.4 Let $\mathcal{C}, \mathcal{D} \in \mathbb{C}_{stable}$.

Then $\mathcal{C}$ and $\mathcal{D}$ are **Pom**-bisimilar iff $\mathcal{C} \approx_{hh}^{\surd} \mathcal{D}$.

Proof "only if" follows immediately from Proposition 9.6 and the transitivity of $\approx_{hh}^{\surd}$. "if" follows from the considerations leading up to Proposition 9.4; π_i is an open map from $\mathcal{C}_R$ to $\mathcal{C}$, and π_2 from C_R to $\mathcal{D}$. ■

This theorem reveals an important difference between $\approx_{h}^{\surd}$ and $\approx_{hh}^{\surd}$. Both satisfy the CCS law $x + x = x$, and, judging by Definition 9.4, both completely respect the causal and branching structure of concurrent systems and their interplay. However, $\approx_{hh}^{\surd}$ may be the finest reasonable equivalence with these properties — it thoroughly respects the internal structure of related systems — whereas $\approx_{h}^{\surd}$ may be the coarsest equivalence of this kind — Definition 9.4 essentially says that it preserves nothing else than causality, branching, and their interplay, and Example 9.6 illustrates that it still makes nontrivial identifications.

Conclusion

In Part I of this paper we defined a compositional action refinement operator on three kinds of event structures and on the new model of configuration structures. We started with prime event structures [NPW], on which the definition of such an operator is fairly straightforward as long as refinement by conflicting or infinite processes is excluded. However, we indicated that in order to treat general refinements it is convenient to use a more expressive model. Here we have taken flow event structures [BC-b, Boudol-b] as well as stable event structures [Winskel] (and we extended the definition to non-stable ones too). Both are also very suited for giving semantics to CCS-like languages. Flow event structures allow for a straightforward generalisation of the definition of action refinement on prime event structures, but their dynamic behaviour is relatively complicated. Moreover, as shown in [CZ], the canonical definition of parallel composition on flow event structures is meaningful on a subclass only. We showed that the subclass provided in [CZ] is not closed under our refinement operator. In a forthcoming paper we will define a larger subclass on which parallel composition is well-behaved, that is closed under action refinement and other relevant process algebraic operators. Stable event structures admit an easier treatment of their dynamic behaviour and do not need to be restricted in order to define the standard process algebra operators in a meaningful way, but necessitate a more complicated definition of action refinement. Alternatively, we could have used *bundle event structures* [Langerak], which are also very suitable as a semantic domain for languages like CCS. This would have given rise to a similar trade-off as for stable event structures.

In order to define action refinement properly we had to distinguish deadlock from successful termination. In the setting of prime event structures we avoided this issue by assuming that all systems under consideration are deadlock-free. This strategy works fine as long as no operators are considered that could introduce deadlocks (e.g. the restriction operator of CCS or the parallel composition of TCSP). In order to deal with deadlock behaviour in the setting of stable event structures, we extended them with an explicit termination predicate. For flow event structures this was not needed, as such a predicate could be derived from their structural properties.

The dynamic behaviour of event structures is entirely determined by their configurations, their deadlock behaviour, given by a subclass of terminating configurations, and the labelling of events. Therefore we have defined a more abstract and general model of concurrency, where a system is represented by exactly these three ingredients. This model of *configuration structures* serves as a common stepping stone between the various notions of event structures and the equivalence relations defined on them. We defined an action refinement operator on this model and proved it consistent with the operators for action refinement on event structures, thus providing a general framework to investigate refinement independently of a particular event structure representation.

Many semantic concepts are defined only for those systems in which causality can be faithfully represented by means of partial orders. In Section 5.3 we characterised the class of configuration structures for which this is the case. It turned out that this is exactly the class of configuration structures that can be obtained as the abstract representations of stable event structures, i.e. Winskel's notion of an event structure is broad enough to capture all configuration structures with this property. This class strictly contains the configuration structures representing flow, bundle and prime event structures.

In Part II we have shown that equivalences based on interleaving of atomic actions or of steps (multisets of concurrently executable actions) are not preserved when changing the level of atomicity of actions. On the other hand, we could show that certain equivalences based on modelling causal relations explicitly by partial orders are indeed preserved under refinement of actions. We considered "linear time" approaches, where the behaviour of a system is equated to the set of possible runs, and "branching time" approaches, where the conflict structure of systems is taken into account. We could show the negative results about the interleaving approaches regardless of the level of detail in modelling the conflict behaviour. However, for the positive results about the partial order approaches, the conflict structure turned out to be crucial. It turned out that linear time partial order semantics (*pomset trace equivalence*) is preserved under action refinement, but the two original approaches to branching time partial order semantics (*pomset bisimulation* [BC-a] and the *NMS partial ordering equivalence* [DDM-a]), as well as their combination, are not. However, we found that a finer branching time partial order semantics due to [RT] — which we call *history preserving bisimulation* — is robust under refinement.

For each of the equivalences considered we could define a termination sensitive variant, distinguishing between deadlock and successful termination, and a termination insensitive variant. We proved that the termination sensitive variants of pomset trace and history preserving bisimulation equivalence are not only preserved under refinement of actions, but even congruences for action refinement. A counterexample showed that this does not hold for the termination insensitive variants.

In the setting of prime event structures, [Bednarczyk] proposes a strengthening of history preserving equivalence, called *hereditary history preserving equivalence*, that is also preserved under action refinement. [JNW] shows that this equivalence can be obtained as a special case of a general categorical definition of bisimilarity on models of concurrency. In Section 9.3 we have generalised the results of [Bednarczyk] and [JNW] to stable configuration structures, thereby also taking successful termination into account. Additionally we show that every stable configuration structure is hereditary history preserving equivalent to a prime event structure with non-binary conflict [Winskel] upgraded with a termination predicate. We argue that history preserving and hereditary history preserving equivalence both preserve causality, branching, and their interplay, and both abstract from choices between identical alternatives; however, the latter may be the finest reasonable equivalence with these properties — it thoroughly respects the internal structure of related systems — whereas the former may be the coarsest equivalence of this kind, still making nontrivial identifications.

Distinguishing deadlock and termination in other models of concurrency

Both [Milner-a] and [Hoare] indicate how CCS and TCSP can be enriched with $\surd$-actions ($\overline{done}$ actions in CCS) in order to capture the distinction between deadlock and successful termination. [AH-a] achieves such a distinction on CCS by means of a predicate on states (= processes). In [TV], $\surd$-actions are used to model termination in a *step failure semantics* for TCSP. The distinction is also made in the metric approach of [BZ], by using a suitable domain equation. The separation of deadlock and successful termination plays a crucial rôle in the language and semantic models of ACP [BK, BW]. In [BK] it is implemented by the introduction of δ-actions, marking deadlock states; in [BW] termination states are marked. In the realm of non-interleaving semantics, Petri net models distinguishing deadlock and successful termination are proposed in [GV-a]; there termination is modelled by means of the empty marking. In the Petri Box Calculus [BDE], termination is indicated by distinguishing so-called exit places; a process terminates when these places carry tokens. The first (prime) event structure model with a distinction between deadlock and successful termination (using $\surd$-actions) is proposed in [BV]. In [Busi] action refinement is defined on stable event structures with a binary conflict relation. Following our treatment of termination in flow event structures, there a configuration X is considered successfully terminated iff every event not in X is in conflict with an event in X. In this approach, an explicit termination predicate as introduced in Section 4 is not needed. On the other hand, unlike in Section 4 or in the work of Winskel, self-conflicting events play a crucial role in the interpretation of event structures, and may not simply be omitted.

Refinement of actions in other models of concurrency

In [BC-a] a generalisation of prime event structures has been considered that is simpler than flow event structures: just the axioms of conflict heredity and finite causes have been dropped. On this model the simple definition of a general action refinement operator we gave for flow event structures can be applied as well. [DD-c] call an event structure in the sense of [BC-a] Δ-*free* if for every events d, e and f with $d\#e < f$ one has $d\#f$ or $d < f$, and show that the class of Δ-free event structures is closed under this operator. This class contains the prime event structures; it extends them by dropping the principle of finite causes and weakening the axiom of conflict heredity. Like prime event structures, Δ-free event structures may be seen as special cases of deadlock-free flow event structures. They seem to be the weakest extension of prime event structures that is closed under action refinement as defined in Definition 3.5. [DD-c] states a refinement theorem for history preserving bisimulation on this model, using the same class of refinements as considered here. However, for both the event structures from [BC-a] and [DD-c], it is difficult to model full CCS or TCSP, in particular to deal with communication.

In [DD-c], action refinement is also defined on *causal trees* [DD-a]. In this model, processes are represented as trees, thereby capturing their branching structure; causalities are encoded into arc labels. Using a canonical representation of Δ-free event structures as causal trees, it is shown that the definition of action refinement on causal trees is in agreement with the one on Δ-free event structures, which is a special case of our operator on flow event structures. Operational definitions of action refinement are presented in [DG-b, Rensink-a/c, BGG, CD-a/b]. The treatment of [DG-b] is shown to agree with the treatment of [DD-c] in terms of causal trees. Likewise, [Busi] shows that (a variant of) the treatment of [BGG] agrees with (a variant of) our treatments on flow and stable event structures. Action refinement has been defined on sets of pomsets — a linear time variant of the model of prime event structures — in [Gischer] and on process graphs and trees, respectively, modelling only sequential processes, in [GW, DD-b]. In [Rensink-a/c], an action refinement operator is defined on *families of posets*, and shown to agree with his operational definition. There is no conceptual difference between the refinement operators mentioned so far; the latter approach may be closest in style to the one pursued here.

In [Gupta] action refinement is defined for *extensional Chu spaces over 2* [Pratt-b]. When abstracting from the morphisms in the category of Chu spaces, these are, up to isomorphism, exactly the configuration structures of Section 5, but without the labelling function and the termination predicate. [Gupta] treats configurations as terminated iff they are maximal. His definition of action refinement deviates from ours only when refining events by non-rooted configuration structures; the idea is that whenever such an event can happen, there may not be a configuration in which no part of its refinement has happened.

Refinement of actions has also been considered in the theory of Petri nets by investigating several constructions for refining transitions in nets. For an overview see [BGV]. In [Valette, SM, Vogler-a/b, BDKP] restricted refinement operators have been defined; these preclude the refinement by a parallel process that occurred in Example 1.1. A general refinement operator for transitions in nets is defined in [GG-a, JM, BDE]. These techniques have been generalised to higher order nets in [DK]. In [Kiehn-a/b], firing a refined transition is like a procedure call, creating an incarnation of the corresponding subnet; such calls may be recursive. This method leads outside the frame of usual Petri nets. Except for the latter one, all refinement operators on nets mentioned here correspond conceptually, through a suitable unfolding, to our operator on event structures.

Syntactic action refinement

The refinement operation we have considered was defined directly on a semantic domain for concurrent systems. An alternative would be to introduce refinement as an operation on syntactic expressions in CCS-like languages (*syntactic action refinement*).

In principle there are two ways to treat syntactic action refinement in languages like CCS. One of them is to use the CCS-actions for modelling the refinable actions of this paper. In the absence of communication (or synchronisation), refinement can simply be defined as syntactic substitution of an action by a process expression. This approach has been taken in [AH-b] and [NEL]. In the presence of synchronisation defining such a refinement operator is much more difficult, as it involves the study of the interaction between action refinement and the chosen synchronisation mechanism. A first proposal, for the simple case of an operator only splitting actions in two parts to be executed sequentially, can be found in [GV-a]. A detailed investigation of a more general approach is undertaken in [Aceto-a, AH-c]. Syntactic action refinement and its relation to semantic action refinement as pursued here is treated comprehensively in [GGR].

An alternative is to use the actions of CCS for modelling "atomic" or instantaneous actions that cannot be refined, and representing our refinable actions by means of variables or *parameters*. This approach requires a general sequential composition operator and has been carried out in [BT] in the setting of ACP. In particular [BT] shows that there is no problem in defining such a refinement operator while working in interleaving semantics: atomic actions a, b cannot be refined, so the equation $a \mid b = a; b + b; a$ is harmless; parameters x, y can be refined, but there is no equation $x \mid y = x; y + y; x$. Of course the refinement operator, ordinary substitution, is defined in the language (that still contains all information about causal dependence) and not in the associated interleaving model (which would be impossible according to Example 1.4).

Forgetful refinement

Because of the anomaly of Example 1.3, we have excluded refinement by the empty process. We know of three possibilities to deal with forgetful refinement nevertheless. In [DG-b] refinement by an empty process is considered an erroneous step in the top-down development procedure, and is therefore identified with refinement by a deadlocked process. As we will allow refinement by deadlocked processes, our approach is equally general. In [BGG] a form of empty refinement on so-called *ST-transition systems* is proposed, in which actions are abstracted away, but their potential to prevent other actions from happening remains. However, the refined processes according to this approach can in general not be represented by Petri nets or the event oriented models considered here. Finally one could, in the spirit of [Vrancken], model the empty refinement by a renaming into a so-called *ε-transition*. In that case the equivalences of this paper would need to be modified to reflect the transparent nature of ε-transitions. An example illustrating the various approaches can be found in [BGG].

Other concepts of action refinement

Our concept of action refinement is characterised by the following properties:

- Actions are uninterpreted, and there is no concept of the correctness of a refinement.
- The refinement of a given system description is completely determined by a *refinement function*, mapping actions to system descriptions.
- All actions may be refined.

- All relations of causal (in)dependence and conflict that exist between actions before refinement are inherited by the refined system.[6]

The first property is in contrast to [Wirth] and many subsequent papers employing stepwise refinement. There actions have a prescribed input/output behaviour, which makes it possible to prove the correctness of a refinement step with respect to this behaviour.

[DGR] abandons the second and fourth property. This paper presents refinement as a category-theoretic construction on event structures, where for every pair of events it is specified explicitly to what extent relations of causality and conflict are inherited by the event structures substituted for these events.

The third property is abandoned in [CGG], where a conceptual distinction between *atomic* and *compound* actions is made; only the latter ones may be refined. It is shown that when all actions which can decide a choice as well as all actions that can occur concurrently with themselves are atomic, action refinement can be applied in interleaving semantics. As in Example 1.4 there is a choice between a and b, these actions may not be refined, and the problem of Example 1.4 is avoided. In [Boudol-a, GMM, GM, Gorrieri-a/b, DG-a, BV-a] another concept of "atomicity" is introduced such that all actions are atomic, but they may still be refined when preserving atomicity. This results in atomic subprocesses that can not be "interrupted" by other activities, i.e. the refinement of P in Example 1.4 would not have the execution a_1ba_2. As the causal independence of a from b in P is not inherited by a_1 and a_2, this approach abandons the fourth property. In [GMM, GM, Gorrieri-a/b, DG-a, BV-a] this kind of refinement is carried out in interleaving semantics. In [Boudol-a] two aspects of atomicity are considered, namely the *all-or-nothing* property (*recoverability*) and *interference freedom*, and two kinds of action refinement are proposed.

The fourth property has been abandoned in [JPZ, JZ-a/b, Zwiers, Janssen, Wehrheim-a/b, RW, Huhn-a/b] as well. In these papers causal dependence is given by a global dependency relation between actions, as in [Mazurkiewicz]. Now causal dependence is inherited only to the extent that it fits this global dependency relation. In [GW-b] it is shown that, in a setting with global action dependencies, interleaving bisimulation coincides with history preserving bisimulation, and hence, using Theorem 9.1, action refinement can be performed in interleaving semantics. In case a and b are dependent, the event structure $\mathcal{E}_P$ of Example 6.1 does not exist; if they are dependent $\mathcal{E}_Q$ does not exist. [Wehrheim-b, RW] still allow to write both the expressions $a|b$ and $a;b+b;a$, but they denote identical systems. By means of an operational semantics they define action refinement on transition systems. The refinement of $\mathit{tree}(P) = \mathit{tree}(Q)$ in Example 1.4 yields a transition system bisimilar to either $\mathit{tree}(P')$ or $\mathit{tree}(Q')$, depending on whether a and b are dependent or not. Using a new process algebraic language with a linear time semantics, [JPZ, JZ-a/b, Zwiers, Janssen] show how to apply action refinement with regard to global action dependencies to the design of *layered systems*, having a sequential structure which still allows some

[6] This property can also be applied to models that abstract from causal (in)dependence (e.g. interleaving models). It then says that it is possible to regard the process representations as abstractions from systems whose causal (non-)relations are inherited when actions are refined.

parallelism, using the principle of communication closed layers. In [JZ-a, Janssen], it is shown how to use this for deriving an implementation of the Two Phase Commitment Protocol. In [JZ-b, Janssen] a distributed algorithm for computing minimum weight spanning trees in graphs is derived. [Huhn-a/b] considers systems consisting of a parallel composition of a fixed number of sequential agents; local refinement functions allow to refine an action differently in different agents. A corresponding notion of refinement is introduced for a logic based on local modalities.

All properties except the third are abandoned in [Rensink-a/b, RG]. There a refinement function does not determine a unique refinement of a given system. Instead they advocate the use of *vertical implementation relations*, parametrised by refinement functions, that allow several correct refinements. In such a refinement step again not all causal relations need to be inherited. This approach turns out to be compatible with interleaving semantics; under a suitable notion of *vertical bisimulation* both P' and Q' can be correct refinements of P $(=Q)$ in Example 1.4.

Refinement in interleaving semantics

As explained for several approaches discussed above, there are ways to define a compositional action refinement operator on interleaving models. However, for our notion of refinement, characterised by the above four properties, interleaving models are not sufficient, provided that they are expressive enough to model both the processes P and Q and discriminating enough not to identify P' and Q' of Example 1.4.

In [BV-b], action refinement in our sense is carried out in linear time interleaving semantics. However, it is not implemented as an operation on an interleaving model. Instead, a syntactic expression is interpreted by treating the actions as variables: its semantics consists of a function that, given any refinement function of the actions in terms of sets of traces, gives an interpretation of the given expression as a set of traces. In particular, two expressions are defined to be equivalent iff they evaluate to the same set of traces for every instantiation of the actions by trace sets. The resulting semantics is therefore by definition preserved under action refinement. Conceptually, the approach of [BV-b] is similar to that of [BT], except that in [BT] the actions of [BV-b] are called "variables" and in addition there are atomic actions that may not be refined.

Related work on equivalence notions

In response to our findings [GG-a], [DDM-c] shows that history preserving bisimulation fits smoothly in the NMS framework [DDM-a] by combining totally and partially ordered observations into mixed orders, and [Vaandrager-b] shows that it coincides with bisimulation equivalence on causal trees [DD-a], a result that has been generalised to stable event structures in [Aceto-b]. [BDKP] defines the *fully concurrent bisimulation equivalence* on Petri nets and also proves a preservation result for action refinement. Through the correspondence between safe Petri nets and prime event structures due to [NPW], this equivalence can be seen to coincide with history preserving equivalence. However, the treatment of [BDKP] covers unsafe Petri nets as well. In [Devillers-a/b] this work is generalised to a setting with internal moves, whereas such a generalisation of our work appears in [Vogler-d/e]. Furthermore, Vogler settles the decidability of history preserving bisimulation on finite safe nets in [Vogler-c] and generalises this result to a setting with internal actions in [Vogler-f]. An overview on refinement theorems in the setting of Petri nets appears in [BGV].

In [GKP, Bednarczyk] a *back-and-forth bisimulation equivalence* is defined on prime event structures by upgrading Definition 6.3 with clauses requiring that single-action transitions should also be matched when going backwards. Such a bisimulation is shown to be characterised by a variant of the Hennessy-Milner logic with backwards modalities. Hereditary history preserving equivalence can be understood as a similar back-and-forth variant of history preserving equivalence. [Bednarczyk] proves that for systems without autoconcurrency back-and-forth bisimulation equivalence coincides with hereditary history preserving equivalence, and hence is preserved under refinement of actions. In the presence of autoconcurrency, back-and-forth bisimulation fails to distinguish $a|a$ from $a;a$, and hence is not preserved under action refinement. [Cherief-a/b] studies a different concept of back-and-forth bisimulation, in which only those backwards transitions need to be matched that lead to states visited before. In this setting, she characterises history preserving equivalence as back-and-forth pomset bisimulation equivalence. This result is applied to give a modal characterisation of history preserving equivalence, using backwards pomset-labelled modalities. This logic is shown to be equally powerful as a logic studied earlier in [DF]; thus the latter also characterises history preserving equivalence.

In our approach for partial order semantics, we have only considered the extreme points in the linear time - branching time spectrum: pomset trace semantics and bisimulations based on partial orders. The most prominent decorated trace equivalence between interleaving trace and bisimulation equivalence is *interleaving failure equivalence* [BHR] ($\approx_{if}$), also known as *testing equivalence* [DH]. This is the coarsest equivalence that respects interleaving traces annotated with deadlock information and is a congruence for the parallel composition operators $\|_A$ of TCSP. An interesting topic is to generalise this notion to a setting where runs are represented as partial orders. This has been pursued in [ADF] and [GW-a]. Since the systems $\mathcal{E}'$ and $\mathcal{F}'$ of Example 9.2 are not even interleaving failure equivalent, no equivalence that is included between interleaving failure and pomset bisimulation equivalence can be preserved under action refinement. This includes the partial order equivalence of [ADF] ($\approx_{pf}$). In [GW-a] a stronger version of testing is considered where tests are sensitive to causal relations between previous action occurrences and actions which are to be performed next. The resulting *causal testing equivalence* ($\approx_{ct}$) is finer than $\approx_{pt}$ and $\approx_{pf}$, coarser than $\approx_h$, and incomparable with $\approx_{pb}$ and $\approx_{wh}$. It is shown that this notion is preserved under action refinement.

A naturally arising question is to what extent it is actually necessary to move to partial order semantics to achieve preservation of equivalence under refinement (here we have only shown that steps are not sufficient). In [vG-a] it is proved that *ST-bisimulation equivalence* [GV-a] ($\approx_{STb}$) and *ST-trace equivalence* ($\approx_{STt}$) are preserved under refinement of actions. In ST-semantics causal links between actions are not explicitly preserved; causal independence can be detected only through the overlap of durational actions in time. The generalisation of interleaving failure equivalence ($\approx_{if}$) to ST-semantics ($\approx_{STf}$) has been shown in [Vogler-b], in the setting of Petri nets, to be preserved under action refinement as well.[7] These ST-equivalences do not imply pomset trace equivalence. Thus it is not necessary to use the full distinguishing power of partial order semantics to get preservation under action refinement.[8] In [Vogler-b/d/e] it has been shown that ST-semantics is the weakest extension of interleaving semantics that is suitable for action refinement: $\approx_{STf}$ is the coarsest equivalence contained in $\approx_{if}$ that is preserved under action refinement, and similar results are obtained for $\approx_{STt}$ and $\approx_{STb}$.

Split semantics is a variant of interleaving semantics based on interleavings of beginnings and ends of action occurrences, instead of entire action occurrences. It is more discriminating than step semantics. The difference with ST-semantics is that the latter additionally takes into account which ends of actions match with which beginnings. For systems without autoconcurrency, split and ST semantics are the same. In [GV-b] an example is given, showing that neither split bisimulation equivalence ($\approx_{2b}$) nor split trace equivalence ($\approx_{2t}$) nor any equivalence in between is preserved under action refinement. This also shows that the split equivalences are strictly coarser than their ST counterparts. In fact splitting actions in n parts yields coarser equivalences than splitting them in $n+1$ parts for any $n \in \mathbb{N}^+$. [Vogler-g] characterises the limit of all the split-n trace equivalences ($\approx_{\omega t}$) and shows that it is not preserved under action refinement. [GL] establishes that, for systems with bounded autoconcurrency, the limit of the split-n bisimulation equivalences coincides with ST-bisimulation equivalence, and hence is preserved under refinement.

The equivalences mentioned so far are indicated in the following diagram, that extends the one given at the beginning of Part II. Note that the ordering into squares in this diagram is a conceptual classification and does not indicate formal relations between the equivalences in terms of implication. For example, it is not the case that $\approx_{pb}$ is finer than $\approx_{STb}$; in fact these notions are incomparable.

Split bisimulation equivalence has been defined on a syntactical level on a subset of CCS in [Hennessy-a]. In [AH-b] it is established that it is preserved by action refinement on this subset. This does not contradict the negative result of [GV-b], as in [vG-a] it

[7] [Vogler-b] uses a restricted set of refinement functions, precluding the refinement by a parallel process that occurred in Example 1.1. His work is generalised to arbitrary refinement functions in [JM].

[8] [Vogler-d] classifies ST-trace and ST-failure semantics as partial order semantics. This because these semantics can be conveniently described by means of so-called *interval orders* [Vogler-b], which are truly partial. On the other hand, the definition of ST-bisimulation [GV-a] does not manifest partial orders. We prefer to use the phrase partial order semantics only when the partial orders denote causal dependencies. Under this convention ST-semantics do not count as partial order semantics. Vogler's conclusion that the power of partial order semantics *is* necessary for refinement of actions [Vogler-d, page 136] is, apart from the chosen terminology, consistent with ours.

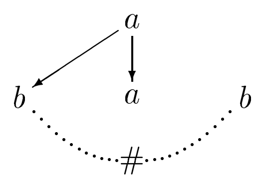
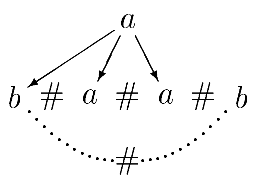
runs / conflict structure		*interleaving semantics*	*step semantics*	*split semantics*	*ST-semantics*	*causal semantics*
		sequences of actions	sequences of steps	sequences of start/ends	interval orders	partial orders (pomsets)
branching time ⋯ linear time	sets of traces	$\approx_{it}$	$\approx_{st}$	$\approx_{2t} \approx_{3t} \cdots \approx_{\omega t}$	$\approx_{STt}$	$\approx_{pt}$
	decorated traces e.g. testing	$\approx_{if}$	$\approx_{sf}$ [TV]		$\approx_{STf}$	$\approx_{pf}$ $\approx_{ct}$
	bisimulation	$\approx_{ib}$	$\approx_{sb}$	$\approx_{2b} \approx_{3b} \cdots$	$\approx_{STb}$	$\approx_{pb}$ $\approx_{wh}$ $\approx_{whpb}$ $\approx_{STpb}$ $\approx_{h}$ $\approx_{hh}$

(dark grey) means: not preserved under refinement   (light grey) means: preserved under refinement

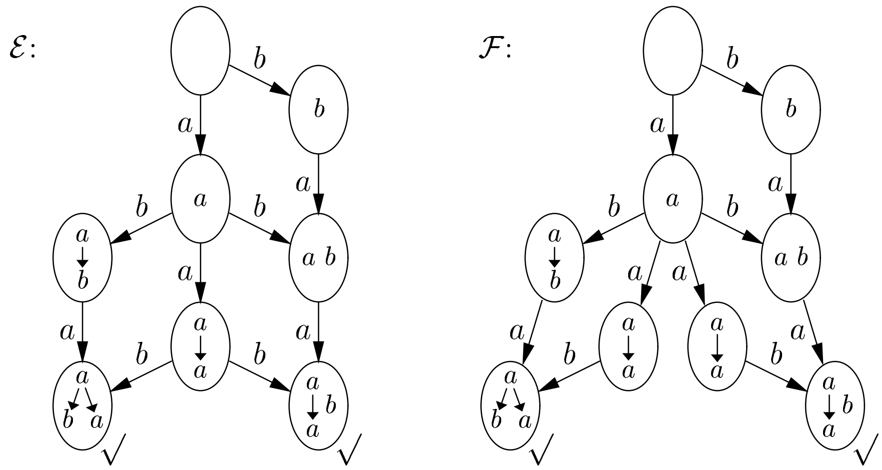
is observed that on that subset, for τ-free systems, $\approx_{2b}$, $\approx_{STb}$ and $\approx_h$ coincide. Thus, the systems making up the counterexample of [GV-b] must fall outside this subset. A linear time variant of ST-equivalence is proven robust under refinement in [NEL], and the same is done for ST-failure equivalence in [AE]. Both papers do this on a simple language, where a syntactic/operational treatment of action refinement is shown to match with a denotational one. Working without autoconcurrency, [NEL] shows that their ST-semantics can be characterised as split semantics. [AE] admits autoconcurrency, but shows that on the studied language, ST-failure semantics is obtained as the limit of split-n failure semantics.

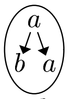
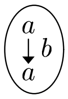
[AH-c, Aceto-a] define *refine equivalence*, also on a syntactical level, but on all of CCS, and prove that it is preserved under syntactic action refinement. This equivalence coincides (at least for τ-free systems) with $\approx_{STb}$. Likewise, [Hennessy-b] defines an *ST-testing* equivalence on CCS and proves preservation under syntactic refinement. This equivalence coincides with $\approx_{STf}$.

Finally, we would like to address the question whether history preserving bisimulation as defined here is the coarsest equivalence finer than pomset bisimulation and being preserved under refinement. We conjectured in [GG-a] that this is not the case, in particular, that for

$\mathcal{E} :=$  and $\mathcal{F} :=$ 

we have $\mathcal{E} \not\approx_h \mathcal{F}$, but for any refinement *ref*, *ref*$(\mathcal{E}) \approx_{pb}$ *ref*$(\mathcal{F})$. That $\mathcal{E} \not\approx_h \mathcal{F}$, in fact even $\mathcal{E} \not\approx_{wh} \mathcal{F}$, can easily be seen from the simplified behaviour structure representations of $\mathcal{E}$ and $\mathcal{F}$ below.



If $\mathcal{E}$ does two a-moves, $\mathcal{F}$ has to answer in kind; however, from the resulting state of $\mathcal{E}$ it is possible to reach both  and , whereas from the matching state of $\mathcal{F}$ only one of them is reachable. In order to see that $ref(\mathcal{E})$ and $ref(\mathcal{F})$ are pomset bisimulation equivalent (for arbitrary ref), relate first all identical refinements of the initial a-events of both structures. Thus as long as only parts of that a-event are executed, any pomset transition of the one structure can be matched by a similar pomset transition of the other. As soon as one of the structures does a move involving part of a b-action (and possibly a part of a second a), a similar move (labelled by an identical pomset) can be made by the other, and the remaining event structures (given by deleting the events that have happened or are in conflict with events that have happened) are isomorphic. Finally, if one event structure performs a part of the second a, without doing any part of a b yet, the other structure can do likewise (in $\mathcal{F}$ it doesn't matter which a is chosen), and the remaining event structures will even be history preserving bisimilar.

In line with the prediction of this example, [Vogler-e] defines an equivalence ($\approx_{STpb}$) that is finer than $\approx_{pb}$ (in fact it combines ST-bisimulation and pomset-bisimulation), relates $\mathcal{E}$ and $\mathcal{F}$, and is preserved under action refinement. It can be argued however that $\approx_{STpb}$ does not respect the interplay of causality and branching: after performing two a-actions in both systems, it may be necessary to match a b that causally depends on one of the a's with one that is independent of both. In fact, judging from its definition, history preserving bisimulation appears to be the coarsest suitable equivalence when it is required to model the interplay of causality and branching in full detail.

Acknowledgements

This research was initiated by a discussion with Albert Meyer and Ernst-Rüdiger Olderog at ICALP 87 in Karlsruhe. Alex Rabinovich helped us by supplying the two systems considered in Example 7.6. Arend Rensink challenged our believe that the Δ-axiom would be preserved under action refinement in flow event structures and helped us find Counterexample 3.7. Kim G. Larsen raised the question whether $\approx_h$ is the coarsest equivalence being preserved under refinement and implying pomset bisimulation. Thanks also to Claudia Goltz for her cooperation.

References

[Aceto-a] L. Aceto (1992): *Action-refinement in process algebras.* Distinguished Dissertations in Computer Science. Cambridge University Press.

[Aceto-b] L. Aceto (1992): *History preserving, causal and mixed-ordering equivalence over stable event structures.* Fundamenta Informaticae XVII(4), pp. 319–331.

[ADF] L. Aceto, R. De Nicola & A. Fantechi (1987): *Testing equivalences for event structures.* In M. Venturini Zilli, editor: *Proceedings Advanced School on Mathematical Models for the Semantics of Parallelism, 1986*, LNCS 280, Springer, pp. 1–20.

[AE] L. Aceto & U. Engberg (1991): *Failures semantics for a simple process language with refinement.* In S. Biswas & K.V. Nori, editors: Proceedings 11th Conference on *Foundations of Software Technology and Theoretical Computer Science,* New Delhi, India, December 1991, LNCS 560, Springer, pp. 89–108.

[AH-a] L. Aceto & M. Hennessy (1992): *Termination, deadlock, and divergence. Journal of the ACM* 39(1), pp. 147–187.

[AH-b] L. Aceto & M. Hennessy (1993): *Towards action-refinement in process algebras. Information and Computation* 103(2), pp. 204–269.

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

[BV] J.C.M. Baeten & F.W. Vaandrager (1992): *An algebra for process creation. Acta Informatica* 29(4), pp. 303–334.

[BW] J.C.M. Baeten & W.P. Weijland (1990): *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press.

[BRR] J.W. de Bakker, W.P. de Roever & G. Rozenberg, editors (1990): Proceedings REX Workshop on *Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness,* Mook, The Netherlands, May/June 1989, LNCS 430. Springer.

[BV-a] J.W. de Bakker & E.P. de Vink (1994): *Bisimulation semantics for concurrency with atomicity and action refinement. Fundamenta Informaticae* XX, pp. 3–34.

[BV-b] J.W. de Bakker & E.P. de Vink (1998): *Full abstractness of an interleaving semantics for action refinement.* Technical Report IR-443, Faculteit der Wiskunde en Informatica, Vrije Universiteit, Amsterdam.

[BZ] J.W. de Bakker & J.I. Zucker (1982): *Processes and the denotational semantics of concurrency. Information and Computation* 54(1/2), pp. 70–120.

[Bednarczyk] M. Bednarczyk (1991): *Hereditary history preserving bisimulations, or what is the power of the future perfect in program logics.* Technical report, Institute of Computer Science, Polish Academy of Sciences, Gdańsk. Available at `ftp://ftp.ipipan.gda.pl/marek/historie.ps.gz`.

[BK] J.A. Bergstra & J.W. Klop (1986): *Algebra of communicating processes.* In J.W. de Bakker, M. Hazewinkel & J.K. Lenstra, editors: *Mathematics and Computer Science*, CWI Monograph 1, North-Holland, Amsterdam, pp. 89–138.

[BT] J.A. Bergstra & J.V. Tucker (1984): *Top down design and the algebra of communicating processes. Science of Computer Programming* 5(2), pp. 171–199.

[BDE] E. Best, R. Devillers & J. Esparza (1993): *General refinement and recursion operators for the Petri box calculus.* In *Proceedings 10th Annual Symposium on Theoretical Aspects of Computer Science,* Würzburg, Germany, February 1993, LNCS 665, Springer, pp. 130–140.

[BDKP] E. Best, R. Devillers, A. Kiehn & L. Pomello (1991): *Concurrent bisimulations in Petri nets. Acta Informatica* 28, pp. 231–264.

[Boudol-a] G. Boudol (1989): *Atomic actions (note). Bulletin of the European Association for Theoretical Computer Science* 38, pp. 136–144.

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

[BC-a] G. Boudol & I. Castellani (1987): *On the semantics of concurrency: partial orders and transition systems.* In H. Ehrig, R. Kowalski, G. Levi & U. Montanari, editors: *Proceedings TAPSOFT 87, Vol. I*, LNCS 249, Springer, pp. 123–137.

[BC-b] G. Boudol & I. Castellani (1989): *Permutation of transitions: an event structure semantics for CCS and SCCS.* In J.W. de Bakker, W.P. de Roever & G. Rozenberg, editors: REX School and Workshop on *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency,* Noordwijkerhout, The Netherlands, May/June 1988, LNCS 354, Springer, pp. 411–427.

[BGV] W. Brauwer, R. Gold & W. Vogler (1991): *A survey of behaviour and equivalence preserving refinements of Petri nets.* In G. Rozenberg, editor: *Advances in Petri nets 1990*, LNCS 483, Springer, pp. 1–46.

[BHR] S.D. Brookes, C.A.R. Hoare & A.W. Roscoe (1984): *A theory of communicating sequential processes. Journal of the ACM* 31(3), pp. 560–599.

[Busi] N. Busi (1993): *Raffinamento di azioni in linguaggi concorrenti.* Master's thesis, Department of Mathematics, University of Bologna, Italy.

[BGG] N. Busi, R.J. van Glabbeek & R. Gorrieri (1994): *Axiomatising ST-bisimulation equivalence.* In E.-R. Olderog, editor: Proceedings IFIP TC2 Working Conference on *Programming Concepts, Methods and Calculi,* San Miniato, Italy, June 1994, IFIP Transactions A-56, North-Holland, pp. 169–188.

[CZ] I. Castellani & G.Q. Zhang (1997): *Parallel product of event structures.* *Theoretical Computer Science* 179(1–2), pp. 203–215.

[CDP] L. Castellano, G. De Michelis & L. Pomello (1987): *Concurrency vs interleaving: an instructive example.* *Bulletin of the European Association for Theoretical Computer Science* 31, pp. 12–15.

[Cherief-a] F. Cherief (1992): *Investigations of back and forth bisimulations on prime event structures.* *Computers and Artificial Intelligence* 5, pp. 481–496.

[Cherief-b] F. Cherief (1992): *Contributions à la sémantique du parallélisme: Bisimulations pour le raffinement et le vrai parallélisme.* PhD thesis, Univ. Grenoble.

[CD-a] J.P. Courtiat & D.E. Saïdouni (1993): *Action refinement in LOTOS.* In A. Danthine, G. Leduc & P. Wolper, editors: Proceedings IFIP TC6 International Symposium on *Proptocal Specification, Testing and Verification, XIII,* Liège, Belgium, May 1993, IFIP Transactions C-16, North-Holland, pp. 341–354.

[CD-b] J.P. Courtiat & D.E. Saïdouni (1995): *Relating maximality-based semantics to action refinement in process algebras.* In D. Hogrefe & S. Leue, editors: Proceedings of the IFIP WG6.1 International Conference on *Formal Description Techniques VII,* Bern, Switserland, October 1994, Chapman & Hall, pp. 293–308.

[CGG] I. Czaja, R.J. van Glabbeek & U. Goltz (1992): *Interleaving semantics and action refinement with atomic choice.* In G. Rozenberg, editor: *Advances in Petri Nets 1992*, LNCS 609, Springer, pp. 89–107.

[DD-a] P. Darondeau & P. Degano (1989): *Causal trees.* In G. Ausiello, M. Dezani-Ciancaglini & S. Ronchi Della Rocca, editors: Proceedings 16th International Colloquium on *Automata, Languages and Programming* (ICALP '89), Stresa, Italy, July 1989, LNCS 372, Springer, pp. 234–248.

[DD-b] Ph. Darondeau & P. Degano (1991): *About semantic action refinement.* *Fundamenta Informaticae* XIV, pp. 221–234.

[DD-c] Ph. Darondeau & P. Degano (1993): *Refinement of actions in event structures and causal trees.* *Theoretical Computer Science* 118, pp. 21–48.

[DDM-a] P. Degano, R. De Nicola & U. Montanari (1987): *Observational equivalences for concurrency models.* In M. Wirsing, editor: *Formal Description of Programming Concepts – III, Proceedings of the 3^{th} IFIP WG 2.2 working conference,* Ebberup 1986, North-Holland, pp. 105–129.

[DDM-b] P. Degano, R. De Nicola & U. Montanari (1988): *On the consistency of 'truly concurrent' operational and denotational semantics.* In *Proceedings third Annual Symposium on Logic in Computer Science,* Edinburgh, IEEE Computer Society Press, pp. 133–141.

[DDM-c] P. Degano, R. De Nicola & U. Montanari (1989): *Partial orderings descriptions and observations of nondeterministic concurrent processes.* In J.W. de Bakker, W.P. de Roever & G. Rozenberg, editors: REX School and Workshop on *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency,* Noordwijkerhout, The Netherlands, May/June 1988, LNCS 354, Springer, pp. 438–466.

[DG-a] P. Degano & R. Gorrieri (1991): *Atomic refinement for process description languages.* In A. Tarlecki, editor: *Proceedings Mathematical Foundations of Computer Science 1991 (MFCS),* LNCS 520, Springer, pp. 121–130.

[DG-b] P. Degano & R. Gorrieri (1995): *A causal operational semantics of action refinement. Information and Computation* 122(1), pp. 97–119.

[DGR] P. Degano, R. Gorrieri & G. Rosolini (1993): *A categorical view of process refinement.* In J.W. de Bakker, W.P. de Roever & G. Rozenberg, editors: Proceedings REX Workshop on *Semantics: Foundations and Applications,* Beekbergen, The Netherlands, June 1992, LNCS 666, Springer, pp. 138–153.

[DF] R. De Nicola & G.L. Ferrari (1990): *Observational logics and concurrency models.* In Proceedings 10th Conference on *Foundations of Software Technology and Theoretical Computer Science,* Bangalore, India, December 1990, LNCS 472, Springer, pp. 301–315.

[DH] R. De Nicola & M. Hennessy (1984): *Testing equivalences for processes. Theoretical Computer Science* 34, pp. 83–133.

[Devillers-a] R. Devillers (1992): *Maximality preserving bisimulation. Theoretical Computer Science* 102, pp. 165–183.

[Devillers-b] R. Devillers (1992): *Maximality preservation and the ST-idea for action refinement.* In G. Rozenberg, editor: *Advances in Petri Nets 1992,* LNCS 609, Springer, pp. 108–151.

[DK] R. Devillers & H. Klaudel (1995): *Refinement and recursion in a high level Petri box calculus.* In J. Desel, editor: Proceedings of the International workshop on *Structures in Concurrency Theory* (STRICT), Berlin, May 1995, Workshops in Computing, Springer, pp. 144–159.

[Gischer] J.L. Gischer (1988): *The equational theory of pomsets. Theoretical Computer Science* 61, pp. 199–224.

[vG-a] R.J. van Glabbeek (1990): *The refinement theorem for ST-bisimulation semantics.* In M. Broy & C.B. Jones, editors: Proceedings IFIP TC2 Working Conference on *Programming Concepts and Methods,* Sea of Gallilee, Israel, North-Holland, pp. 27–52.

[vG-b] R.J. van Glabbeek (1990): *Comparative Concurrency Semantics and Refinement of Actions.* PhD thesis, Free University, Amsterdam. Second edition available as *CWI tract* 109, CWI, Amsterdam 1996.

[vG-c] R.J. van Glabbeek (1990): *The linear time – branching time spectrum.* In J.C.M. Baeten & J.W. Klop, editors: *Proceedings CONCUR 90,* Amsterdam, LNCS 458, Springer, pp. 278–297.

[GG-a] R.J. van Glabbeek & U. Goltz (1989): *Equivalence notions for concurrent systems and refinement of actions.* Arbeitspapiere der GMD 366, Gesellschaft für Mathematik und Datenverarbeitung, Sankt Augustin. An extended abstract appeared in A. Kreczmar & G. Mirkowska, editors: Proceedings 14^{th} Symposium on *Mathematical Foundations of Computer Science* (MFCS'89), Porąbka-Kozubnik, Poland, August/September 1989, LNCS 379, Springer, pp. 237–248.

[GG-b] R.J. van Glabbeek & U. Goltz (1989): *Partial order semantics for refinement of actions – neither necessary nor always sufficient but appropriate when used with care. Bulletin of the European Association for Theoretical Computer Science* 38, pp. 154–163.

[GG-c] R.J. van Glabbeek & U. Goltz (1990): *Refinement of actions in causality based models.* In [BRR], pp. 267–300.

[GG-d] R.J. van Glabbeek & U. Goltz (1990): *Equivalences and refinement.* In I. Guessarian, editor: Proceedings *Semantics of Systems of Concurrent Processes,* LITP Spring School on Theoretical Computer Science, La Roche Posay, France, April 1990, LNCS 469, Springer, pp. 309–333.

[GG-e] R.J. van Glabbeek & U. Goltz (1990): *A deadlock-sensitive congruence for action refinement.* SFB-Bericht Nr. 342/23/90 A, Institut für Informatik, Technische Universität München.

[GP] R.J. van Glabbeek & G.D. Plotkin (1995): *Configuration structures (extended abstract).* In D. Kozen, editor: Proceedings 10^{th} Annual IEEE Symposium on *Logic in Computer Science* (LICS95), San Diego, USA, IEEE Computer Society Press, pp. 199–209.

[GV-a] R.J. van Glabbeek & F.W. Vaandrager (1987): *Petri net models for algebraic theories of concurrency.* In J.W. de Bakker, A.J. Nijman & P.C. Treleaven, editors: *Proceedings PARLE conference,* Eindhoven, *Vol. II (Parallel Languages),* LNCS 259, Springer, pp. 224–242.

[GV-b] R.J. van Glabbeek & F.W. Vaandrager (1997): *The difference between splitting in n and $n+1$.* Information and Computation 136(2), pp. 109–142.

[GW] R.J. van Glabbeek & W.P. Weijland (1989): *Refinement in branching time semantics.* Report CS-R8922, CWI, Amsterdam. In J.W. de Bakker, 25 jaar semantiek, liber amicorum, CWI, Amsterdam 1989, pp. 247-252. Also in Proceedings AMAST Conference, Iowa, USA, May 1989, pp. 197-201.

[GGR] U. Goltz, R. Gorrieri & A. Rensink (1996): *Comparing syntactic and semantic action refinement.* Information and Computation 125(2), pp. 118–143.

[GKP] U. Goltz, R. Kuiper & W. Penczek (1992): *Propositional temporal logics and equivalences.* In W.R. Cleaveland, editor: Proceedings CONCUR '92, Stony Brook, NY, USA, August 1992, LNCS 630, Springer, pp. 222–236.

[GW-a] U. Goltz & H. Wehrheim (1996): *Causal testing.* In W. Penczek & A. Szalas, editors: Proceedings 21st International Symposium on Mathematical Foundations of Computer Science (MFCS'96), Cracow, Poland, September 1996, LNCS 1113, Springer, pp. 394–406.

[GW-b] U. Goltz & H. Wehrheim (1996): *Modelling causality via action dependencies in branching time semantics.* Information Processing Letters 59(4), pp. 179–184.

[Gorrieri-a] R. Gorrieri (1991): *Refinement, Atomicity and Transactions for Process Description Languages.* PhD thesis: TD-2/91, Dipartimento di Informatica, Università di Pisa.

[Gorrieri-b] R. Gorrieri (1992): *A hierarchy of system descriptions via atomic linear refinement.* Fundamenta Informaticae XVI, pp. 289–336.

[GL] R. Gorrieri & C. Laneve (1995): *Split and ST bisimulation semantics.* Information and Computation 118(2), pp. 272–288.

[GMM] R. Gorrieri, S. Marchetti & U. Montanari (1990): *A^2CCS: Atomic actions for CCS.* Theoretical Computer Science 72(2–3), pp. 203–223.

[GM] R. Gorrieri & U. Montanari (1990): *Towards hierarchical specification of systems: A proof system for strong prefixing.* International Journal of Foundations of Computer Science 1(3), pp. 277–293.

[Gupta] V. Gupta (1994): *Chu Spaces: A Model of Concurrency.* PhD thesis, Stanford University. Available at `ftp://boole.stanford.edu/pub/gupthes.ps.gz`.

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

[Hennessy-b] M. Hennessy (1995): *Concurrent testing of processes.* Acta Informatica 32(6), pp. 509–543.

[HM] M. Hennessy & R. Milner (1985): *Algebraic laws for nondeterminism and concurrency.* *Journal of the ACM* 32(1), pp. 137–161.

[Hoare] C.A.R. Hoare (1985): *Communicating Sequential Processes.* Prentice Hall, Englewood Cliffs.

[Huhn-a] M. Huhn (1996): *Action refinement and property inheritance in systems of sequential agents.* In U. Montanari & V. Sassone, editors: *Proceedings CONCUR '96: 7th International Conference on Concurrency Theory,* Pisa, Italy, August 1996, LNCS 1119, Springer, pp. 639–654.

[Huhn-b] M. Huhn (1997): *On the Hierarchical Design of Distributed Systems.* PhD thesis, University of Hildesheim, Germany.

[Janssen] W. Janssen (1994): *Layered Design of Parallel Systems.* PhD thesis, Department of Computer Science, University of Twente.

[JPZ] W. Janssen, M. Poel & J. Zwiers (1991): *Action systems and action refinement in the development of parallel systems.* In J. C. M. Baeten & J. F. Groote, editors: *Proceedings CONCUR 91,* Amsterdam, LNCS 527, Springer, pp. 298–316.

[JZ-a] W. Janssen & J. Zwiers (1992): *Protocol design by layered decomposition: A compositional approach.* In J. Vytopil, editor: Proceedings Second International Symposium on *Formal Techniques in Real-Time and Fault-Tolerant Systems,* Nijmegen, The Netherlands, January 1992, LNCS 571, Springer, pp. 307–326.

[JZ-b] W. Janssen & J. Zwiers (1992): *From sequential layers to distributed processes, deriving a distributed minimum weight spanning tree algorithm (extended abstract).* In Proceedings 11th ACM Symposium on *Principles of Distributed Computing*, ACM, pp. 215–227.

[JM] L. Jategaonkar & A. Meyer (1992): *Testing equivalence for Petri nets with action refinement.* In W.R. Cleaveland, editor: *Proceedings CONCUR '92,* Stony Brook, NY, USA, August 1992, LNCS 630, Springer, pp. 17–31.

[JNW] A. Joyal, M. Nielsen & G. Winskel (1996): *Bisimulation from open maps.* *Information and Computation* 127(2), pp. 164–185.

[Kiehn-a] A. Kiehn (1989): *A Structuring Mechanism for Petri Nets.* PhD thesis, Report TUM-I8902, Technische Universität München, Germany.

[Kiehn-b] A. Kiehn (1990): *Petri net systems and their closure properties.* In G. Rozenberg, editor: *Advances in Petri Nets 1989*, LNCS 424, Springer, pp. 306–328.

[Lamport] L. Lamport (1986): *On interprocess communication.* *Distributed Computing* 1(2), pp. 77–101.

[Langerak] R. LANGERAK (1992): *Transformations and Semantics for LOTOS.* PhD thesis, Department of Computer Science, University of Twente.

[LG] R. LOOGEN & U. GOLTZ (1991): *Modelling nondeterministic concurrent processes with event structures.* Fundamenta Informaticae XIV(1), pp. 39–74.

[Mazurkiewicz] A. MAZURKIEWICZ (1987): *Trace theory.* In W. Brauer, W. Reisig & G. Rozenberg, editors: *Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II;* Proceedings of an Advanced Course, Bad Honnef, September 1986, LNCS 255, Springer, pp. 279–324.

[Milner-a] R. MILNER (1980): *A Calculus of Communicating Systems*, LNCS 92. Springer.

[Milner-b] R. MILNER (1989): *Communication and Concurrency.* Prentice Hall, Englewood Cliffs.

[Moller] F. MOLLER (1989): *Axioms for concurrency.* PhD thesis, Report CST-59-89, Department of Computer Science, University of Edinburgh.

[NEL] M. NIELSEN, U. ENGBERG & K.S. LARSEN (1989): *Fully abstract models for a process language with refinement.* In J.W. de Bakker, W.P. de Roever & G. Rozenberg, editors: REX School and Workshop on *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, Noordwijkerhout, The Netherlands, May/June 1988, LNCS 354, Springer, pp. 523–548.

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

[OH] E.-R. OLDEROG & C.A.R. HOARE (1986): *Specification-oriented semantics for communicating processes.* Acta Informatica 23, pp. 9–66.

[Pomello] L. POMELLO (1986): *Some equivalence notions for concurrent systems – An overview.* In G. Rozenberg, editor: *Advances in Petri Nets 1985*, LNCS 222, Springer, pp. 381–400.

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

[Pratt-b] V.R. PRATT (1995): *Chu spaces and their interpretation as concurrent objects.* In J. van Leeuwen, editor: *Computer Science Today: Recent Trends and Developments*, LNCS 1000, Springer, pp. 392–405.

[RT] A. RABINOVICH & B.A. TRAKHTENBROT (1988): *Behavior structures and nets.* Fundamenta Informaticae 11(4), pp. 357–404.

[Reisig] W. REISIG (1985): *Petri nets – an introduction.* EATCS Monographs on Theoretical Computer Science, Volume 4. Springer.

[Rensink-a] A. Rensink (1993): *Models and Methods for Action Refinement.* PhD thesis, University of Twente, The Netherlands.

[Rensink-b] A. Rensink (1994): *Methodological aspects of action refinement.* In E.-R. Olderog, editor: Proceedings IFIP TC2 Working Conference on *Programming Concepts, Methods and Calculi,* San Miniato, Italy, June 1994, IFIP Transactions A-56, North-Holland, pp. 227–246.

[Rensink-c] A. Rensink (1995): *An event-based SOS for a language with refinement.* In J. Desel, editor: Proceedings of the International workshop on *Structures in Concurrency Theory* (STRICT), Berlin, May 1995, Workshops in Computing, Springer, pp. 294–309.

[RG] A. Rensink & R. Gorrieri (1997): *Action refinement as an implementation relation.* In Bidoit & Dauchet, editors: *Proceedings TAPSOFT '97: Theory and Practice of Software Development*, LNCS 1214, Springer, pp. 772–786.

[RW] A. Rensink & H. Wehrheim (1997): *Dependency-based action refinement.* In I. Prívara & P. Ružička, editors: Proceedings 22nd International Symposium on *Mathematical Foundations of Computer Science* (MFCS'97), Bratislava, Slovakia, August 1997, LNCS 1295, Springer, pp. 468–477.

[Schreiber] S. Schreiber (1991): *Flow event structures as a semantic model for a language with sequential composition.* Diplomarbeit, University of Bonn.

[SM] I. Suzuki & T. Murata (1983): *A method for stepwise refinement and abstraction of Petri nets.* *Journal of Computer and System Sciences* 27(1), pp. 51–76.

[TV] D.A. Taubner & W. Vogler (1989): *Step failure semantics and a complete proof system.* *Acta Informatica* 27(2), pp. 125–156.

[Vaandrager-a] F.W. Vaandrager (1989): *A simple definition for parallel composition of prime event structures.* Report CS-R8903, CWI, Amsterdam.

[Vaandrager-b] F.W. Vaandrager (1989): *An explicit representation of equivalence classes of the history preserving bisimulation.* Unpublished manuscript, quoted in [DD-c].

[Valette] R. Valette (1979): *Analysis of Petri nets by stepwise refinements.* *Journal of Computer and System Sciences* 18, pp. 35–46.

[Vogler-a] W. Vogler (1987): *Behaviour preserving refinements of Petri nets.* In G. Tinhofer & G. Schmidt, editors: 12^{th} *International Workshop on Graph-Theoretic Concepts in Computer Science,* Bernried, 1986, LNCS 246, Springer, pp. 82–93.

[Vogler-b] W. Vogler (1991): *Failures semantics based on interval semiwords is a congruence for refinement.* *Distributed Computing* 4, pp. 139–162.

[Vogler-c] W. VOGLER (1991): *Deciding history preserving bisimilarity.* In J. Leach Albert, B. Monien & M. Rodríguez-Artalejo, editors: Proceedings 18th International Colloquium on *Automata, Languages and Programming* (ICALP '91), Madrid, Spain, July 1991, LNCS 510, Springer, pp. 495–505.

[Vogler-d] W. VOGLER (1992): *Modular construction and partial order semantics of Petri nets.* LNCS 625. Springer.

[Vogler-e] W. VOGLER (1993): *Bisimulation and action refinement. Theoretical Computer Science* 114, pp. 173–200.

[Vogler-f] W. VOGLER (1995): *Generalized OM-bisimulation. Information and Computation* 118(1), pp. 38–47.

[Vogler-g] W. VOGLER (1996): *The limit of* $\mathrm{Split}_\mathrm{n}$*-language equivalence. Information and Computation* 127(1), pp. 41–61.

[Vrancken] J.L.M. VRANCKEN (1986): *The algebra of communicating processes with empty process.* Report FVI 86-01, Dept. of Computer Science, University of Amsterdam.

[Wehrheim-a] H. WEHRHEIM (1994): *Parametric action refinement.* In E.-R. Olderog, editor: Proceedings IFIP TC2 Working Conference on *Programming Concepts, Methods and Calculi,* San Miniato, Italy, June 1994, IFIP Transactions A-56, North-Holland, pp. 247–266.

[Wehrheim-b] H. WEHRHEIM (1996): *Specifying Reactive Systems with Action Dependencies — Modelling and Hierarchical Design.* PhD thesis, University of Hildesheim, Germany.

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

[Wirth] N. WIRTH (1971): *Program development by stepwise refinement. Communications of the ACM* 14(4), pp. 221–227.

[Zwiers] J. ZWIERS (1992): *Layering and action refinement for typed systems.* In J.W. de Bakker, C. Huizing, W.P. de Roever & G. Rozenberg, editors: Proceedings REX Workshop on *Real-Time: Theory in Practice,* Mook, The Netherlands, June 1991, LNCS 600, Springer, pp. 687–723.

Index