

Event Structures for Interrupt Process Algebras

Harald Fecher

*Universität Mannheim, Fakultät für Mathematik und Informatik
D7, 27, 68131 Mannheim, Germany*

Abstract

Interruption is a useful feature in programming and specification languages. Therefore, process algebras has been extended with an additional interrupt operator. We invent a class of event structures, called interrupt event structures, to give a true concurrent semantic to process algebras containing interruption. Interrupt event structures are more expressive than other event structures with respect to event trace execution. Furthermore, interrupt event structures can also distinguish simultaneous event executions from event interleaving. We show consistency, based on bisimulation, between the operational and the denotational semantics of a process algebra that contains an interrupt operator.

Key words: process algebra, event structures, interruption, consistency, true concurrency

1 Introduction

A useful feature in programming and specification languages is the ability to describe the interruption of a normal execution of the system. Interrupt mechanisms are, for example, increasingly used to perform memory management. It is already the case that some modern, general-purpose CPUs have sophisticated ways of handling interrupts. See for example [26], where an interrupt calculus is invented to analyze such CPU-based interrupts. See also [24,29], where precise interrupts are discussed. Since it is also essential to have interrupt mechanisms in reactive, concurrent systems (for example the downloading from the internet can be interrupted and continued by the user), various interrupt mechanisms are added to process algebras, see e.g. [3,4,11,12]. A common approach is to use an interrupt operator (\gg) where process $B_1 \gg B_2$ behaves like process B_1 , except that it may be interrupted by B_2 at any time before the termination of B_1 . Furthermore, B_1 continues its execution after the termination of B_2 .

True concurrency models are important to describe the independence of events (actions). *Event structures* (e.g. [9,21,30]) are typical true concurrency models. They are especially used to give truly concurrent denotational semantics to process algebras. Standard event structures still lack the possibility to describe interrupts: an event that gets interrupted has to become disabled, and later it has to become

*This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs*

enabled again (when the interrupting process has terminated). But standard event structures cannot undo the disabling¹. Exceptions are the *inhibitor event structures* [5]. Nevertheless, in general they are not sufficient to give denotational semantics to process algebras that contains an interrupt operator (more precisely, to the process algebra presented here), since the disabling of these event structures is too restrictive.

The cancellation of the disabling is modelled in our event structures, which we call *interrupt event structures*, as follows: We collect a set X of events that disables event e , i.e. e is disabled if all events of X are executed. Then we connect another set Y of events to the tuple (X, e) . This Y denotes the cancellation of the disabling, i.e. the disabling is removed (or will not happen anymore) if an event from Y is executed. In other words, when an event becomes disabled and when it becomes enabled is encoded in a single relation.

It turns out that some event structures are special cases of interrupt event structures. Furthermore, the other standard event structures have less expressive power than interrupt event structures with respect to event traces. Interrupt event structures can also distinguish event interleaving from simultaneous event executions. More precisely, it is possible that two events may interleave but they cannot be executed simultaneously. We classify the sets of step event traces that can be described by interrupt event structures. A process algebra that contains an interrupt operator is also introduced. We give an operational semantics in terms of (labelled) transition systems and a denotational semantics in terms of interrupt event structures. A consistency result between the two semantics is given.

The paper is organized as follows: In Section 2 we introduce the interrupt event structures. Their expressive power is examined in Section 3. Section 4 introduces our process algebra together with the two semantics and the consistency result. Related works are discussed in Section 5.

2 Interrupt Event Structures

In the following, $\mathcal{P}(M)$ denotes the powerset of M , and $\mathcal{P}_f(M)$ denotes the set of all finite subsets of M . Furthermore, we assume a fixed countable set of events \mathcal{U} such that $\bullet \in \mathcal{U}$ and $\star \notin \mathcal{U}$ and for all $e, e' \in \mathcal{U}$ we have $(e, e'), (\star, e), (e, \star) \in \mathcal{U}$. The constraints on \mathcal{U} result from technical reasons (they guarantee the well-definedness of the operators which are defined in Subsection 4.2).

Definition 2.1 An *interrupt event structure* (IES), denoted by $\mathcal{I} = (E, \rightsquigarrow)$, is an element of $\mathcal{P}(\mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{P}(\mathcal{U}) \times \mathcal{U})$ such that $\rightsquigarrow \subseteq \mathcal{P}_f(E) \times \mathcal{P}(E) \times E$. Let $\mathcal{M}^{\mathcal{I}}$ denote the set of all interrupt event structures.

E is called *set of events* and \rightsquigarrow is called *interrupt relation*. In the following, we will write $X \xrightarrow{Y} e$ rather than $(X, Y, e) \in \rightsquigarrow$.

¹ Disabling is modelled in standard event structure by a binary relation between (sets of) events and events, hence once an event is disabled it remains disabled.

The intuitive meaning of an IES is the following: Set X in $X \overset{Y}{\rightsquigarrow} e$ states when event e gets disabled and set Y indicates when this disabling condition gets removed. More precisely, if all events from X have been executed, then e is disabled unless an event from Y has already been executed. In other words, the first component of \rightsquigarrow is universally quantified (Winskel’s approach [30]) and the second component is existentially quantified (bundle approach [21]). It is possible to use the same kind of quantification (universal or existential) in both components, but this would complicate the definition of event execution. Furthermore, with our definition (*pre-*)inhibitor event structures² [5] become a special case of interrupt event structures. Additionally, we have the feeling that the mixed quantification approach gives simpler opportunities to extend interrupt event structures by performance issues, like time.

Some IES are shown in Figure 1. Events, here named a, b, c, d , are depicted as dots. The first component of an interrupt relation is illustrated by wavy lines and the second component by straight lines. More precisely, $X \overset{Y}{\rightsquigarrow} e$ is depicted by a wavy arrow from the elements of X to e together with a straight arrow from the elements of Y to this wavy line. Furthermore, we use only one wavy line for each (X, e) tuple. For example, the upper wavy line in \mathcal{I}_2 encodes $\{b\} \overset{\{b\}}{\rightsquigarrow} a$ and $\{b\} \overset{\{c\}}{\rightsquigarrow} a$. The straight line points directly to e if X is the empty set, as it is the case for the arrow from b in \mathcal{I}_1 . The straight lines are omitted if Y is the empty set unless X is also the empty set, as it is the case in \mathcal{I}_3 . Process algebra terms to which the depicted interrupt event structures correspond are also given in Figure 1. The process algebra terms will be introduced in Section 4.

Remark 2.2 Interrupt event structures have to be extended with an additional constraint (called approximation closedness) in order to obtain a complete partial order³ (if the standard approach is used to define the order) [14,13]. This technical constraint is omitted here in order to increase the readability.

Hereafter, we consider \mathcal{I} to be (E, \rightsquigarrow) . Furthermore, $\text{init}(\mathcal{I})$ denotes the set of the initial events of \mathcal{I} , i.e. those events in \mathcal{I} which are ready to be executed.

$$\text{init}(\mathcal{I}) = \{e \in E \mid \neg(\exists Y : \emptyset \overset{Y}{\rightsquigarrow} e)\}$$

The remainder of an IES with respect to an initial event e describes the system after the execution of e . Therefore, we remove e in E and also from the first components of the interrupt relation. Furthermore, we only consider those tuples $X \overset{Y}{\rightsquigarrow} e'$ where $e \notin Y$. Formally:

Definition 2.3 Let $\mathcal{I} \in \mathcal{M}^{\mathcal{I}}$ and $e \in \text{init}(\mathcal{I})$. The *remainder* of \mathcal{I} with respect to e is the interrupt event structure $\mathcal{I}_{[e]} = (E \setminus \{e\}, \rightsquigarrow')$ where $\rightsquigarrow' = \{(X \setminus \{e\}, Y, e') \mid e' \neq e \wedge e \notin Y \wedge X \overset{Y}{\rightsquigarrow} e\}$.

² (Pre-)inhibitor event structures have a similar interrupt relation (the second component is existentially quantified) except that only single events are allowed in the first component. Hence, their disabling is more restrictive than in interrupt event structures.

³ Complete partial orders [1] are used to give meanings to recursive processes.

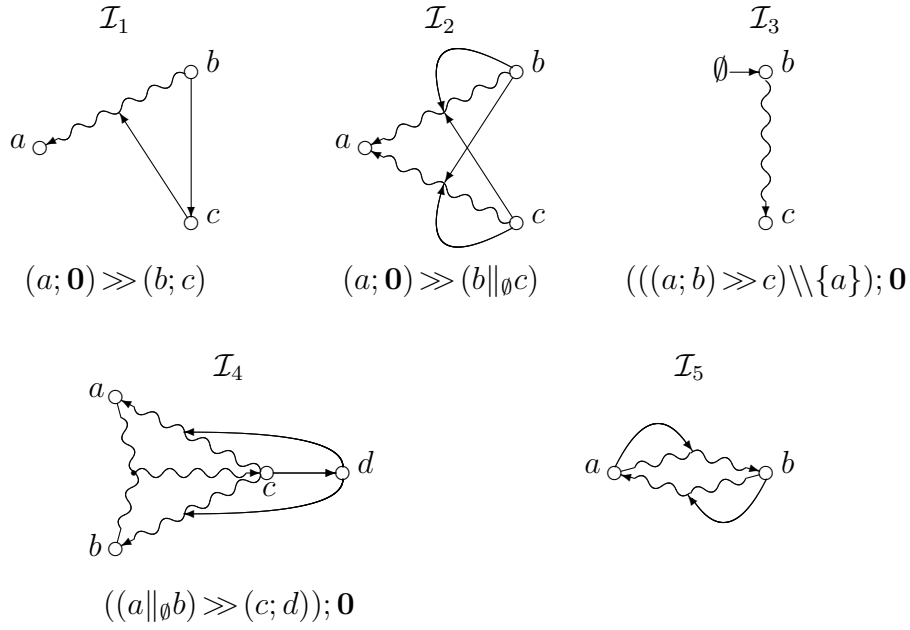


Fig. 1. Some Interrupt Event Structures

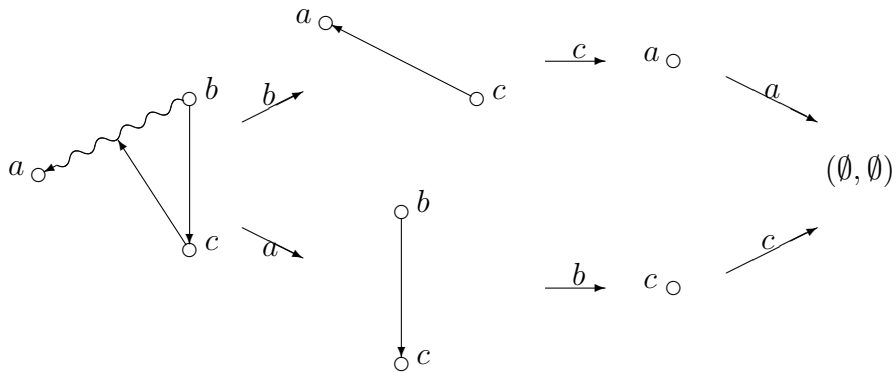


Fig. 2. Event Execution of \mathcal{I}_1

The event execution of \mathcal{I}_1 shown in Figure 1 is presented in Figure 2. The execution of event b in \mathcal{I}_2 shown in Figure 1 yields the IES presented in Figure 3. There it can be easily seen that c is the only initial event of $\mathcal{I}_{2[b]}$, since $\emptyset \xrightarrow{\{c\}} a$. Event c of \mathcal{I}_4 shown in Figure 1 is disabled if both events a and b are executed, but not if only one event a or b is executed. This can be seen in Figure 3, where the execution of event a in \mathcal{I}_4 is shown.

3 Expressive Power

3.1 Event Traces

In this subsection, we compare the expressive power of interrupt event structures with other event structures with respect to the event traces they can execute.


 Fig. 3. Partly Event Execution of \mathcal{I}_2 and \mathcal{I}_4

Definition 3.1 An *event trace*, denoted by \vec{v} , is a finite sequence of distinct elements of \mathcal{U} . Let \mathcal{V} denote the set of all event traces.

If $\vec{v} = (e_1, \dots, e_n) \in \mathcal{V}$, then $\bar{\vec{v}} = \{e_1, \dots, e_n\}$.

An event trace (e_1, \dots, e_n) is an execution of $\mathcal{I} \in \mathcal{M}^{\mathcal{I}}$ if and only if $e_{i+1} \in \text{init}(\mathcal{I}_{[e_1] \dots [e_i]})$ for every $i < n$. Let $\text{EvTr}(\mathcal{I})$ denote the set of all event traces that are executions of \mathcal{I} .

The set of event traces corresponding to \mathcal{I}_1 is $\text{EvTr}(\mathcal{I}_1) = \{\emptyset, a, ab, abc, b, bc, bca\}$.

Dual event structures [19] and consequently (*extended*) *bundle* [20,21], *prime* [25] and (*pre*-)*asymmetric event structures*⁴ [6], are special cases of interrupt event structures if the termination and the action labelling of the events are neglected⁵: The binary conflict relation, for example between e and e' , is modelled by $\{e\} \overset{\emptyset}{\rightsquigarrow} e'$, and the bundle causality relation ($Y \mapsto e$) is modelled by $\emptyset \overset{Y}{\rightsquigarrow} e$. Fortunately, in these cases the depiction of the dual event structures coincides with the depiction of the corresponding interrupt event structures (see for example the first two IESs shown in Figure 5 on page 10).

(*Pre*-)*inhibitor event structures* [5] are interrupt event structures where the first component of \rightsquigarrow has to be either the empty set or a single event set. *Termination bundle* or *termination precursor event structures*⁶ [13] can, similarly to dual event structures, be encoded in interrupt event structures such that they have the same set of event traces. Hence, interrupt event structures can also describe the sets of event traces of *flow* [9] and Winskel's (*stable*) *event structures* [30], since they can be described by termination event structures [13].

On the other hand, the set of event traces corresponding to the interrupt event structure \mathcal{I}_4 shown in Figure 1 cannot be described by any of the other class of event structures mentioned. These facts are summarized in the following theorem:

Theorem 3.2 *Every set of event traces that is described by a prime [25], flow [9], Winskel's [30], bundle [21], (pre)-asymmetric [6], extended bundle [20], dual [19], termination bundle [13], termination precursor [13] or (pre)-inhibitor event structure [5] can also be described by an interrupt event structure, but not vice*

⁴ (Pre)-asymmetric event structures are prime event structures where the conflict relation is not necessarily symmetric.

⁵ Interrupt event structures are extended with termination and action labelling in Subsection 4.2.

⁶ Termination bundle or termination precursor event structures have a more general disabling relation (set of events disables an event).

versa.

Proof. Can be easily checked by the results given in [13] and by the above described embedding of dual event structures in interrupt event structures. \square

3.2 Simultaneous Event Execution

In this subsection, we introduce step executions of interrupt event structures.

Definition 3.3 A *step event trace*, denoted by \vec{s} , is a finite sequence of pairwise disjoint elements of $\mathcal{P}_f(\mathcal{U}) \setminus \{\emptyset\}$. Let \mathcal{S} denote the set of all step event traces.

A set Y of initial events are independent if Y does not trigger a disabling of an event of Y . This is formalized in the following definition, where step execution on interrupt event structures is also introduced.

Definition 3.4 [Step Execution] Let $\mathcal{I} \in \mathcal{M}^{\mathcal{I}}$. Define the set of all sets of initial, independent events by

$$\widehat{\text{init}}(\mathcal{I}) = \{Y \in \mathcal{P}_f(E) \setminus \{\emptyset\} \mid \forall (X, Y', e) \in \rightsquigarrow: e \in Y \Rightarrow \neg(X \subseteq Y' \setminus \{e\})\}.$$

A step event trace (Y_1, \dots, Y_n) is an execution of $\mathcal{I} \in \mathcal{M}^{\mathcal{I}}$ if $\forall i < n : Y_{i+1} \in \widehat{\text{init}}(\mathcal{I}_{[Y_1] \dots [Y_i]})$, where $\mathcal{I}_{[\{e_1, \dots, e_m\}]} = \mathcal{I}_{[e_1] \dots [e_m]}$. Let $\text{StEvTr}(\mathcal{I})$ denote the set of all step event traces that are executions of \mathcal{I} .

The step event execution in Definition 3.4 is well defined, since $\mathcal{I}_{[e_1] \dots [e_m]}$ is defined and it is independent from the order of these events whenever $\{e_1, \dots, e_m\} \in \widehat{\text{init}}(\mathcal{I})$. The set $\{\{a\}, \{a\}\{b\}, \{b\}, \{b\}\{a\}\}$ consists of all step event trace executions obtained from \mathcal{I}_5 shown in Figure 1. From this example, it follows that two events do not have to be independent ($\{a, b\}$ is not in $\widehat{\text{init}}(\mathcal{I}_5)$) if they can be executed in any order. In other words, the interleaving of events does not imply independence of these events in $\mathcal{M}^{\mathcal{I}}$. But this is the case for nearly all other event structures. Hence, the interrupt event structures are of special interest with respect to true concurrency. Please note that we have considered interleaving with respect to event execution and not with respect to action execution (where the action denoted to the event, see Subsection 4.2, is only observable and not the event itself). Of course it is possible to distinguish independence and interleaving with respect to action execution in the standard event structures (to some amount).

3.3 Classification of the Set of Step Event Traces from $\mathcal{M}^{\mathcal{I}}$

Step event trace execution (and also event trace execution) in $\mathcal{M}^{\mathcal{I}}$, is independent of the execution order. Furthermore, simultaneous event executions imply event interleaving. This is formalized as follows:

Definition 3.5 Suppose \mathbf{S} is a set of step event traces, i.e. $\mathbf{S} \subseteq \mathcal{S}$. Then

- \mathbf{S} is *prefix closed* if $\forall (Y_1, \dots, Y_n) \in \mathbf{S} : \forall m : m \leq n \Rightarrow (Y_1, \dots, Y_m) \in \mathbf{S}$,
- \mathbf{S} is *interleaving closed* if $\forall (Y_1, \dots, Y_{n+1}) \in \mathbf{S} : \forall Y \in \mathcal{P}_f(Y_{n+1}) \setminus \{\emptyset, Y_{n+1}\} : (Y_1, \dots, Y_n, Y, Y_{n+1} \setminus Y) \in \mathbf{S}$,

- \mathbf{S} is *history-order independent* if

$$\forall (Y_1, \dots, Y_{n+1}), (Y'_1, \dots, Y'_m) \in \mathbf{S} : \bigcup_{i=1}^n Y_i = \bigcup_{i=1}^m Y'_i \Rightarrow (Y'_1, \dots, Y'_m, Y_{n+1}) \in \mathbf{S}.$$

It can be easily checked that the prefix closed, history-order independent sets of event traces (event traces correspond to step event traces where only single sets of events are allowed) and *event automata*⁷ [27] where every state can be reached are equivalent approaches. Furthermore, prefix closed, interleaving closed, history-order independent sets of step event traces and *local event structures*⁸ [17] are equivalent approaches.

Theorem 3.6 *Let $\mathbf{S} \subseteq \mathcal{S}$. Then \mathbf{S} is prefix closed, interleaving closed and history-order independent if and only if $\exists \mathcal{I} \in \mathcal{M}^{\mathcal{I}} : \mathbf{S} = \text{StEvTr}(\mathcal{I})$.*

Proof. [Sketch]

\Rightarrow : It can be checked that the interrupt event structure given by $(\mathcal{U}, \{(X \cup \tilde{X}, \mathcal{U} \setminus X, e) \mid |X \cup \tilde{X}| < \infty \wedge X \cap \tilde{X} = \emptyset \wedge e \notin X \cup \tilde{X} \wedge \forall (Y_1, \dots, Y_n) \in \mathcal{S} : X = \bigcup_{i=1}^n Y_i \Rightarrow (Y_1, \dots, Y_n, \tilde{X} \cup \{e\}) \notin \mathbf{S}\})$ satisfies the requirement.

\Leftarrow : Easily checked. □

An immediately consequence of Theorem 3.6 is that for every set \mathbf{S} of event traces, i.e. $\mathbf{S} \subseteq \mathcal{V}$, we have \mathbf{S} is prefix closed and history-order independent if and only if $\exists \mathcal{I} \in \mathcal{M}^{\mathcal{I}} : \mathbf{S} = \text{EvTr}(\mathcal{I})$.

4 Interrupt Process Algebra

Our process algebra is based on the interrupt process algebra of [3]⁹. Consequently, we follow the philosophy that the ‘final’ executed action of a process terminates the process [8].

Let τ denote the *internal action*. Furthermore, let Obs be a set such that $\tau \notin \text{Obs}$. We call Obs the set of *observable actions*. The *set of all actions* \mathcal{Act} is defined by $\mathcal{Act} = \{\tau\} \cup \text{Obs}$. Furthermore, we assume a fixed countable set of *process variables* Var which is disjoint from \mathcal{Act} . The process algebra expressions EXP are defined by the following BNF-grammar.

$$B ::= \mathbf{0} \mid a \mid B + B \mid B; B \mid B \triangleright B \mid B \gg B \mid B \parallel_A B \mid B \setminus A \mid x$$

where $x \in \text{Var}$, $a \in \mathcal{Act}$ and $A \subseteq \text{Obs}$. A *process* with respect to EXP is a pair $\langle \text{decl}, B \rangle$ consisting of a declaration $\text{decl} : \text{Var} \rightarrow \text{EXP}$ and an expression $B \in \text{EXP}$. Let PA denote the set of all processes. We sometimes call an expression $B \in \text{EXP}$ a process if decl is clear from the context.

⁷ An event automaton is a set of configurations together with a relation on configurations, which indicates the events that can be executed in the corresponding configurations.

⁸ Local event structures can be considered as event automata where step executions are also described.

⁹ We use different operator symbols.

The expressions have the following intuitive meaning: $\mathbf{0}$ is the inactive process, i.e. it cannot execute any action. a is the process that executes a and terminates. $B_1 + B_2$ is a choice between the behaviors described by B_1 and B_2 . The choice is determined by the first action that occurs. $B_1; B_2$ is the sequential composition, i.e. B_1 proceeds until it terminates, after which B_2 takes over. $B_1 [> B_2$ is the disruption of B_1 by B_2 , i.e. any action from B_2 disables B_1 as long as B_1 has not terminated. On the other hand, the termination of B_1 disables B_2 . The interrupt process $B_1 \gg B_2$ behaves like B_1 , except that it may be interrupted by B_2 at any time before the termination of B_1 . Furthermore, B_1 continues its execution after the termination of B_2 . $B_1 \parallel_A B_2$ describes the parallel execution of B_1 and B_2 where both processes have to synchronize on actions from A . The process terminates if both sides terminate in the case of synchronization or if one of them terminates and the other one has already terminated. The restriction process $B \setminus A$ executes action a if B executes action a , provided a is not contained in A . The behavior of x is given by the declaration, i.e. it allows the description of recursion.

4.1 Operational Semantics

The operational semantics of processes is given in terms of (labelled) transition systems. As stated before, we adopt the philosophy that the ‘final’ action terminates the process. Therefore, we have to distinguish between ‘final’ and ‘non-final’ actions. This is modelled by using a predicate for every action a , indicating that the process can terminate by performing action a [8,3]. This predicate is typically denoted by $\xrightarrow{a} \surd$, more precisely $B \xrightarrow{a} \surd$ indicates that B can terminate by executing action a . The transition rules of $\longrightarrow_{\text{decl}} \subseteq \text{EXP} \times \text{Act} \times (\text{EXP} \cup \{\surd\})$ with respect to $\text{decl} : \text{Var} \rightarrow \text{EXP}$ are presented in Figure 4.

4.2 Denotational Semantics

We extend the interrupt event structures by termination and action labelling information in order to give a denotational semantics to PA. We use an additional set of event sets (T) to model termination. A system run is terminated if every element of T contains an event of the system run. Furthermore, an event structure can only terminate by executing an event, i.e. we demand that $\text{T} \neq \emptyset$. However, T might consist of the empty set only.

Before we present the definition, we introduce the following notation: $M_1 \rightarrow M_2$ denotes the set of all partial functions from M_1 to M_2 (we sometimes consider a partial function from M_1 to M_2 as a subset of $M_1 \times M_2$). Furthermore, the domain of a partial function f is the set $\{m \mid f(m) \text{ is defined}\}$ and is denoted by $\text{dom}(f)$. Function $f \upharpoonright M'_1$, where $M'_1 \subseteq M_1$, denotes the restriction of function f to domain M'_1 . We write $f(m_1) \simeq f'(m'_1)$ to denote that $(f(m_1) \text{ is defined} \Leftrightarrow f'(m'_1) \text{ is defined}) \wedge (f(m_1) \text{ is defined} \Rightarrow f(m_1) = f'(m'_1))$.

Definition 4.1 A labelled interrupt event structure $\mathcal{E} = (E, \rightsquigarrow, \text{T}, \mathbf{1})$ is an element

$\frac{}{a \xrightarrow{a} \checkmark}$	$\frac{B_1 \xrightarrow{a} B'}{B_1 + B_2 \xrightarrow{a} B'}$	$\frac{B_1 \xrightarrow{a} \checkmark}{B_1 + B_2 \xrightarrow{a} \checkmark}$
	$B_2 + B_1 \xrightarrow{a} B'$	$B_2 + B_1 \xrightarrow{a} \checkmark$
$\frac{B_1 \xrightarrow{a} B'_1}{B_1; B_2 \xrightarrow{a} B'_1; B_2}$	$\frac{B_1 \xrightarrow{a} \checkmark}{B_1; B_2 \xrightarrow{a} B_2}$	
$\frac{B_1 \xrightarrow{a} B'_1}{B_1 [> B_2 \xrightarrow{a} B'_1 [> B_2}$	$\frac{B_1 \xrightarrow{a} \checkmark}{B_1 [> B_2 \xrightarrow{a} \checkmark}$	
$B_2 [> B_1 \xrightarrow{a} B'_1$	$B_2 [> B_1 \xrightarrow{a} \checkmark$	
$\frac{B_1 \xrightarrow{a} B'_1}{B_1 \gg B_2 \xrightarrow{a} B'_1 \gg B_2}$	$\frac{B_1 \xrightarrow{a} \checkmark}{B_1 \gg B_2 \xrightarrow{a} \checkmark}$	
$B_2 \gg B_1 \xrightarrow{a} B'_1; B_2$	$B_2 \gg B_1 \xrightarrow{a} B_2$	
$\frac{B_1 \xrightarrow{a} B'_1 \quad a \notin A}{B_1 \ _A B_2 \xrightarrow{a} B'_1 \ _A B_2}$	$\frac{B_1 \xrightarrow{a} B'_1 \quad B_2 \xrightarrow{a} B'_2 \quad a \in A}{B_1 \ _A B_2 \xrightarrow{a} B'_1 \ _A B'_2}$	
$B_2 \ _A B_1 \xrightarrow{a} B_2 \ _A B'_1$		
$\frac{B_1 \xrightarrow{a} \checkmark \quad a \notin A}{B_1 \ _A B_2 \xrightarrow{a} B_2 \setminus \setminus A}$	$\frac{B_1 \xrightarrow{a} \checkmark \quad B_2 \xrightarrow{a} B'_2 \quad a \in A}{B_1 \ _A B_2 \xrightarrow{a} B'_2 \setminus \setminus A}$	
$B_2 \ _A B_1 \xrightarrow{a} B_2 \setminus \setminus A$	$B_2 \ _A B_1 \xrightarrow{a} B'_2 \setminus \setminus A$	
$\frac{B_1 \xrightarrow{a} \checkmark \quad B_2 \xrightarrow{a} \checkmark \quad a \in A}{B_1 \ _A B_2 \xrightarrow{a} \checkmark}$		$\frac{\text{decl}(x) \xrightarrow{a} B'}{x \xrightarrow{a} B'}$
$\frac{B \xrightarrow{a} B' \quad a \notin A}{B \setminus \setminus A \xrightarrow{a} B' \setminus \setminus A}$	$\frac{B \xrightarrow{a} \checkmark \quad a \notin A}{B \setminus \setminus A \xrightarrow{a} \checkmark}$	$\frac{\text{decl}(x) \xrightarrow{a} \checkmark}{x \xrightarrow{a} \checkmark}$

 Fig. 4. Transition Rules for $\xrightarrow{\text{decl}}$

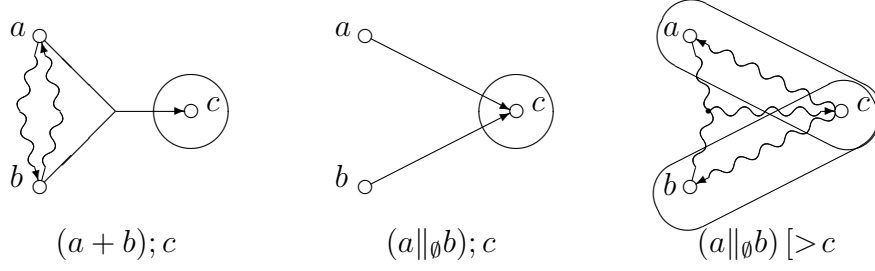


Fig. 5. Some Labelled Interrupt Event Structures

of $\mathcal{P}(\mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U})) \times (\mathcal{U} \rightarrow \mathcal{Act})$ such that

- $(E, \rightsquigarrow) \in \mathcal{M}^{\mathcal{I}}$
- $T \subseteq \mathcal{P}(E)$ and $T \neq \emptyset$
- $\text{dom}(l) = E$ and
- $\forall \vec{v} \in \text{EvTr}((E, \rightsquigarrow)) :$
 $(\forall Z \in T : \vec{v} \cap Z \neq \emptyset) \Rightarrow (\forall e \in E \setminus \vec{v} : \exists X \subseteq \vec{v}, Y \subseteq E \setminus \vec{v} : X \rightsquigarrow^Y e)$

Let $\mathcal{M}^{\mathcal{E}}$ denote the set of all labelled interrupt event structures.

T is called the *termination set* and l the *action-labelling* function. The last constraint on labelled interrupt event structures guarantees that after termination ($\forall Z \in T : \vec{v} \cap Z \neq \emptyset$) no further event can be executed. Some labelled interrupt event structures are shown in Figure 5. The events and the interrupt relation are depicted as in Figure 1. The action names are shown close to the dots (we do not name the events explicitly and identify them with the action names if no confusion arises). We mark a termination set $Z \in T$ by surrounding its events with a closed line.

Hereafter, we consider \mathcal{E} to be $(E, \rightsquigarrow, T, l)$ and \mathcal{E}_i to be $(E_i, \rightsquigarrow_i, T_i, l_i)$. The definition of the initial events is extended to labelled interrupt event structures by $\text{init}(\mathcal{E}) = \text{init}((E, \rightsquigarrow))$. Furthermore, the predicate $\Upsilon(T, e)$ holds if and only if e is a *termination event* with respect to T , i.e. \mathcal{E} terminates by executing e . Formally:

$$\Upsilon(T, e) \iff \forall Z \in T : e \in Z$$

We present the operators on $\mathcal{M}^{\mathcal{E}}$ that are used to define the denotational semantics. In the following, π_i denotes the projection to the i^{th} component.

Definition 4.2 [Operators on $\mathcal{M}^{\mathcal{E}}$] Let $A \subseteq \text{Obs}$ and $i \in \{1, 2\}$. Then define $\tilde{+} : \mathcal{M}^{\mathcal{E}} \times \mathcal{M}^{\mathcal{E}} \rightarrow \mathcal{M}^{\mathcal{E}}$ with $\mathcal{E}_1 \tilde{+} \mathcal{E}_2 \simeq (E_1 \cup E_2, \tilde{\rightsquigarrow}, \tilde{T}, l_1 \cup l_2)$ where

$$\begin{aligned} \tilde{\rightsquigarrow} &= \rightsquigarrow_1 \cup \{(\{e_2\}, \emptyset, e_1) \mid e_1 \in \text{init}(\mathcal{E}_1) \wedge e_2 \in \text{init}(\mathcal{E}_2)\} \cup \\ &\quad \rightsquigarrow_2 \cup \{(\{e_1\}, \emptyset, e_2) \mid e_1 \in \text{init}(\mathcal{E}_1) \wedge e_2 \in \text{init}(\mathcal{E}_2)\} \\ \tilde{T} &= \{Z_1 \cup Z_2 \mid Z_1 \in T_1 \wedge Z_2 \in T_2\} \end{aligned}$$

$\tilde{;} : \mathcal{M}^{\mathcal{E}} \times \mathcal{M}^{\mathcal{E}} \rightarrow \mathcal{M}^{\mathcal{E}}$ with $\mathcal{E}_1 \tilde{;} \mathcal{E}_2 \simeq (E_1 \cup E_2, \tilde{\rightsquigarrow}, T_2, l_1 \cup l_2)$ where

$$\tilde{\rightsquigarrow} = \rightsquigarrow_1 \cup \rightsquigarrow_2 \cup \{(\emptyset, Z_1, e_2) \mid e_2 \in \text{init}(\mathcal{E}_2) \wedge Z_1 \in T_1\}$$

$$\begin{aligned}
 \widetilde{>} : \mathcal{M}^\varepsilon \times \mathcal{M}^\varepsilon &\rightarrow \mathcal{M}^\varepsilon \text{ with } \mathcal{E}_1 \widetilde{>} \mathcal{E}_2 \simeq (E_1 \cup E_2, \widetilde{\rightarrow}, \widetilde{T}, l_1 \cup l_2) \text{ where} \\
 \widetilde{\rightarrow} &= \rightsquigarrow_1 \cup \rightsquigarrow_2 \cup \{(\{e_2\}, \emptyset, e_1) \mid e_1 \in E_1 \wedge e_2 \in \text{init}(\mathcal{E}_2)\} \cup \\
 &\quad \{(X_1, \emptyset, e_2) \mid e_2 \in \text{init}(\mathcal{E}_2) \wedge X_1 \in \mathcal{P}_f(\bigcup T_1) \wedge \forall Z_1 \in T_1 : X_1 \cap Z_1 \neq \emptyset\} \\
 \widetilde{T} &= \{Z_1 \cup Z_2 \mid Z_1 \in T_1 \wedge Z_2 \in T_2\} \\
 \\
 \widetilde{\gg} : \mathcal{M}^\varepsilon \times \mathcal{M}^\varepsilon &\rightarrow \mathcal{M}^\varepsilon \text{ with } \mathcal{E}_1 \widetilde{\gg} \mathcal{E}_2 \simeq (E_1 \cup E_2, \widetilde{\rightarrow}, T_1, l_1 \cup l_2) \text{ where} \\
 \widetilde{\rightarrow} &= \rightsquigarrow_1 \cup \rightsquigarrow_2 \cup \{(\{e_2\}, Z_2, e_1) \mid e_1 \in E_1 \wedge e_2 \in \text{init}(\mathcal{E}_2) \wedge Z_2 \in T_2\} \cup \\
 &\quad \{(X_1, \emptyset, e_2) \mid e_2 \in \text{init}(\mathcal{E}_2) \wedge X_1 \in \mathcal{P}_f(\bigcup T_1) \wedge \forall Z_1 \in T_1 : X_1 \cap Z_1 \neq \emptyset\} \\
 \\
 \widetilde{\parallel}_A : \mathcal{M}^\varepsilon \times \mathcal{M}^\varepsilon &\rightarrow \mathcal{M}^\varepsilon \text{ with } \mathcal{E}_1 \widetilde{\parallel}_A \mathcal{E}_2 = (\widetilde{E}, \widetilde{\rightarrow}, \widetilde{T}, \widetilde{l}) \text{ where} \\
 \widetilde{E} &= (E_1^f \times \{\star\}) \cup (\{\star\} \times E_2^f) \cup E^s \\
 E_i^f &= \{e \in E_i \mid l_i(e) \notin A\} \\
 E^s &= \{(e_1, e_2) \in E_1 \times E_2 \mid l_1(e_1) = l_2(e_2) \in A\} \\
 \widetilde{\rightarrow} &= \{(\{\{e'_1, e'_2\}\}, \emptyset, (e_1, e_2)) \mid (e'_1, e'_2) \neq (e_1, e_2) \wedge \exists i : e'_i = e_i\} \cup \\
 &\quad \{(\widetilde{X}, \widetilde{Y}, (e_1, e_2)) \mid \exists i : \exists Y_i : \pi_i(\widetilde{X}) \xrightarrow{Y_i} e_i \wedge \widetilde{Y} = \{(e'_1, e'_2) \in \widetilde{E} \mid e'_i \in Y_i\}\} \\
 \widetilde{T} &= \{\widetilde{Z} \mid \exists i : \exists Z_i \in T_i : \widetilde{Z} = \{(e_1, e_2) \in \widetilde{E} \mid e_i \in Z_i\}\} \\
 \widetilde{l}(e_1, e_2) &= \begin{cases} l_1(e_1) & \text{if } e_2 = \star \\ l_2(e_2) & \text{otherwise} \end{cases} \\
 \widetilde{\parallel\!\!\! \setminus} A : \mathcal{M}^\varepsilon &\rightarrow \mathcal{M}^\varepsilon \text{ with } \mathcal{E} \widetilde{\parallel\!\!\! \setminus} A = \mathcal{E} \upharpoonright \{e \in E \mid l(e) \notin A\}.
 \end{aligned}$$

It is clear that all operators applied to \mathcal{E}_1 and \mathcal{E}_2 are defined if \mathcal{E}_1 and \mathcal{E}_2 have disjoint sets of events. We are only interested in such cases (out of readability, we define the operators without modelling the disjoint union explicitly).

The denotational meaning of processes $\mathbf{0}$ and a are $\llbracket \langle \text{decl}, \mathbf{0} \rangle \rrbracket = (\emptyset, \emptyset, \{\emptyset\}, \emptyset)$ and $\llbracket \langle \text{decl}, a \rangle \rrbracket = (\{\bullet\}, \emptyset, \{\{\bullet\}\}, \{(\bullet, a)\})$. The meaning of the other processes of PA can be derived from the operators in the standard way [13] (including recursion), where the disjointness of the sets of events can be ensured by using event renaming operators. Examples of the denotational semantics of processes can be found in Figure 1 and Figure 5, where some processes are shown with their corresponding labelled interrupt event structures.

4.3 Consistency

Our causality result is based on bisimulation [22]. Please note that bisimulation is a congruence for our process algebra, which follows from the format of the transition rules [2]. Before we present our consistency result, we introduce action execution (including termination information) for labelled interrupt event structures.

Definition 4.3 The transition relation $\hookrightarrow \subseteq \mathcal{M}^\mathcal{E} \times \mathcal{Act} \times (\mathcal{M}^\mathcal{E} \cup \{\sqrt{\ }\})$ is defined by $\mathcal{E} \xrightarrow{a} \mathcal{E}'$ if and only if there is $e \in \text{init}(\mathcal{E})$ such that $a = l(e)$ and

- $(\neg \Upsilon(\mathbb{T}, e)) \Rightarrow ((E', \sim') = \mathcal{I}_{[e]} \wedge l' = 1 \upharpoonright E' \wedge \mathbb{T} = \{Z \in \mathbb{T} \mid e \notin Z\})$
- $\Upsilon(\mathbb{T}, e) \Rightarrow \mathcal{E}' = \sqrt{\ }.$

Theorem 4.4 (Consistency) *Suppose $\langle \text{decl}, B \rangle \in \text{PA}$. Then the transition systems $(\text{EXP}, \mathcal{Act}, \longrightarrow_{\text{decl}}, B)$ and $(\mathcal{M}^\mathcal{E}, \mathcal{Act}, \hookrightarrow, \llbracket \langle \text{decl}, B \rangle \rrbracket)$ are bisimilar.*

Proof. [Sketch] The proof uses the techniques presented in [13]: An event based transition relation is defined such that the events correspond to the events of the denotational semantics. This technique is used especially to handle unguarded recursion. Then we show that its corresponding transition system is bisimilar to $(\text{EXP}, \mathcal{Act}, \longrightarrow_{\text{decl}}, B)$ and in addition bisimilar to $(\mathcal{M}^\mathcal{E}, \mathcal{Act}, \hookrightarrow, \llbracket \langle \text{decl}, B \rangle \rrbracket)$. Hence, the statement follows by the transitivity of the bisimilarity [22]. \square

5 Related Work

How interrupt event structures are related to other event structures is intensively examined in Section 3. There it is also shown that *local event structures* [17] (and *event automata* [27]) have the same expressive power as interrupt event structures with respect to step event traces (respectively event traces). Event automata and local event structures represent configurations explicitly, i.e. conflicts are not described intensionally as it is done in interrupt event structures. The intentional representation has, for example, the advantage of making the extension with time aspects less expensive.

(1-safe) *Petri nets* [28] can model some kinds of interruption. For example, the process $(a; 0) \gg (b; c)$ is modelled by the Petri net pictured on the left hand side in Figure 6. But 1-safe Petri nets (also if they are extended with read and inhibitor arcs [23]) cannot model the behavior of \mathcal{I}_4 shown in Figure 1, where sets of events $(\{a, b\})$ disable an event (c) . On the other hand, 1-safe Petri nets that also contain read and inhibitor arcs can be expressed in inhibitor event structures [5] and consequently in interrupt event structures.

General Petri nets (with possibly multiple tokens on places) can model the behavior of \mathcal{I}_4 shown in Figure 1, but till now it is not clear how an interrupt operator can be defined. On the other hand, interrupt event structure may help to give a simpler semantics to general Petri nets.

Petri nets can also model event interleaving without implying simultaneous executions. See for example the Petri net pictured on the right hand side in Figure 6. This Petri net has the same behavior with respect to step execution as the interrupt event structure \mathcal{I}_5 shown in Figure 1.

Configuration structures [15] lack the possibility of modelling interruptions, nor do they distinguish between event interleaving and simultaneous executions. *Comtraces* [18], which are a generalization of step based *Mazurkiewicz traces*, and *chu-spaces* [16] distinguish between event interleaving and simultaneous execu-

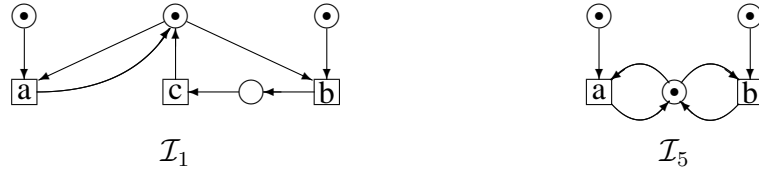


Fig. 6. Corresponding Petri Nets

tions. Contrary to our approach, event interleaving implies simultaneous executions in these models but not vice versa.

References

- [1] Samson Abramsky and Achim Jung. Domain theory. In Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994.
- [2] Luca Aceto, Wan Fokkink, and Chris Verhoef. Structural operational semantics. In Bergstra et al. [7], pages 197–292.
- [3] J. C. M. Baeten and J. A. Bergstra. Mode transfer in process algebra. Report CSR 00-01, Vakgroep Informatica, Technische Universiteit Eindhoven, 2000.
- [4] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, 9:127–168, 1986.
- [5] Paolo Baldan, Nadia Busi, Andrea Corradini, and G. Michele Pinna. Functional concurrent semantics for petri nets with read and inhibitor arcs. In C. Palamidessi, editor, *CONCUR 2000 – Concurrency Theory*, volume 1877 of *LNCS*, pages 442–457. Springer-Verlag, 2000.
- [6] Paolo Baldan, Andrea Corradini, and Ugo Montanari. Contextual petri nets, asymmetric event structures, and processes. *Information and Computation*, 171:1–49, 2001.
- [7] J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra*. North-Holland, 2001.
- [8] Jan A. Bergstra, Wan Fokkink, and Alban Ponse. Process algebra with recursive operations. In Bergstra et al. [7], pages 333–389.
- [9] Gérard Boudol and Ilaria Castellani. Permutation of transitions: an event structure semantics for CCS and SCCS. In de Bakker et al. [10], pages 411–427.
- [10] J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors. *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*. Springer-Verlag, 1989.
- [11] B. Dierens. New features in PSFI – interrupts, disrupts, and priorities. Report P9417, Programming Research Group - University of Amsterdam, 1994.

- [12] A. Engels and Th. Cobben. Interrupt and disrupt in MSC: Possibilities and problems. In Y. Lahav, A. Wolisz, J. Fischer, and E. Holz, editors, *Proceedings fo the 1st Workshop of the SDL Forum Society on SDL and MSC*, number 104 in Informatikberichte. Humboldt-Universitt zu Berlin, 1998.
- [13] Harald Fecher. *Action Refinement in End-Based Choice Settings*. PhD thesis, Universität Mannheim, 2003.
- [14] Harald Fecher, Mila Majster-Cederbaum, and Jinzhao Wu. Bundle event structures: A revised cpo approach. *Information Processing Letters*, 83:7–12, 2002.
- [15] R. J. van Glabbeek and G. D. Plotkin. Configuration structures. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, pages 199–209. IEEE Computer Society Press, 1995.
- [16] Vineet Gupta. *Chu Spaces: A Model of Concurrency*. PhD thesis, Stanford University, 1994.
- [17] P. W. Hoogers, H. C. M. Kleijn, and P. S. Thiagarajan. An event structure semantics for general Petri nets. *Theoretical Computer Science*, 153:129–170, 1996.
- [18] Ryszard Janicki and Maciej Koutny. Semantics of inhibitor nets. *Information and Computation*, 123:1–16, 1995.
- [19] Joost-Pieter Katoen. *Quantitative and Qualitative Extension of Event Structures*. PhD thesis, Enschede: Centre for Telematics and Information Technology, P.O. Box 217 - 7500 AE Enschede - The Netherlands, 1996.
- [20] Rom Langerak. *Transformations and Semantics for LOTOS*. PhD thesis, Department of Computer Science, University of Twente, 1992.
- [21] Rom Langerak. Bundle event structures: A non-interleaving semantics for LOTOS. In M. Diaz and R. Groz, editors, *Formal Description Techniques, V*, pages 331–346. Elsevier, 1993.
- [22] Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.
- [23] Ugo Montanari and Francesca Rossi. Contextual nets. *Acta Informatica*, 32:545–596, 1995.
- [24] Mayan Moudgill and Stamatis Vassiliadis. Precise interrupts. *IEEE Micro*, 16(1):58–67, 1996.
- [25] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.
- [26] Jens Palsberg and Di Ma. A typed interrupt calculus. In W. Damm and E.-R. Olderog, editors, *FTRFT 2002*, volume 2469 of *LNCS*, pages 291–310. Springer-Verlag, 2002.
- [27] G. Michele Pinna and Axel Poigné. On the nature of events: another perspective in concurrency. *Theoretical Computer Science*, 138(2):425–454, 1995.

- [28] Wolfgang Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.
- [29] Wade Walker and Harvey G. Cragon. Interrupt processing in concurrent processors. *Computer*, 28(6):36–46, 1995.
- [30] Glynn Winskel. An introduction to event structures. In de Bakker et al. [10], pages 364–397.