Event Structures

Seminar "Concurrency Theory WS 2011/2012"

Markus Hoffmann

Event Structures are a mathematical model that describes computational processes and their progress. In their general version, they were formalized by Glynn Winskel [W87] to establish a connection between Petri nets and concepts from domain theory. Event structures can be considered a model of true concurrency, unlike interleaving models.

1 Introduction

The basic idea of event structures is to establish a causal dependency between a set of events. There are few restrictions on the nature of these events. Some examples for events are transitions of an automaton, synchronization actions of processes or observable behaviour of an electric circuit. A computational process can thus be described by the set of events it generates as the computation goes on. A corresponding event structure expresses how these events are related to each other.

Interleaving models express concurrency indirectly by giving all possible orders in which concurrent events can occur. Event structures do not impose a causal dependency on concurrent events at all, there is no order on the occurrence of concurrent events. Since the causal dependency of events is expressed in a direct way, concurrent events are unrelated. This is why event structures can be considered a model of true concurrency.

This paper will give an overview of event structures, their formal definitions and properties. It should give an insight on the roles of non-determinism and concurrency as well as how progress of a computational process can be modelled with event structures.

2 Event Structures

In this section - following [W89] - three event structure definitions will be introduced successively: elementary event structures, prime event structures, and general event structures. It will be shown why modelling non-determinism and concurrency make the more general model necessary.

2.1 Elementary Event Structures

Elementary event structures were introduced in an early attempt of Nielsen, Plotkin and Winskel [NPW79] at combining the models of Petri nets and Scott domains. A partial order on a set of events is used to model their causal dependency.

Definition 1. A **partial order** $\leq$ on a set S is a binary relation that satisfies the following properties:

- reflexivity $\forall a \in S \,.\, a \leq a$
- transitivity $\forall a, b, c \in S \,.\, a \leq b \wedge b \leq c \Rightarrow a \leq c$
- antisymmetry $\forall a, b \in S \,.\, a \leq b \wedge b \leq a \Rightarrow a = b$

The tuple $(S, \leq)$ is called a **partially ordered set**.

Definition 2 ([NPW79])**.** An **elementary event structure** $(E, \leq)$ is a partially ordered set where

- E is a set of events and
- $\leq$ is the causal dependency relation, a partial order on E.

Intuition: For two events $e_1, e_2 \in E$ the fact that $e_1 \leq e_2$ means that the occurrence of e_2 depends on e_1 having occurred before.

In the following examples an intuitive graphical representation of transition systems will be used to model their corresponding event structures.

Definition 3. A transition system $(S, I, A, \rightarrow)$ is a tuple where

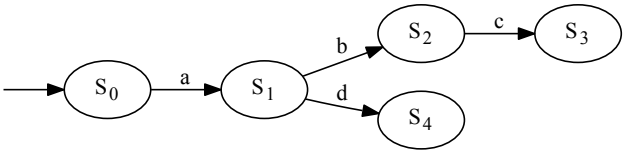
- S is a set of states,
- $I \subseteq S$ is the set of initial states,
- A is a set of actions and
- $\rightarrow \subseteq S \times A \times S$ is the transition relation.

For $s, t \in S, a \in A$ the expression $s \xrightarrow{a} t$ means that there is a transition from s to t by taking the action a.

To model a transition system with an event structure, one can use the transitions as observable events.

Example 1. A transition system and a possible representation as an elementary event structure.

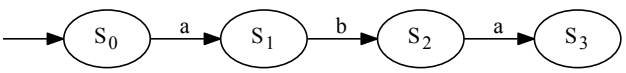
$$S = \{S_0, S_1, S_2, S_3, S_4\} \qquad I = \{S_0\} \qquad A = \{a, b, c, d\}.$$



This transition system can be represented by an elementary even structure $(E, \leq)$ where

$$E = \{a, b, c, d\} \text{ and } \leq = \{(a,a), (b,b), (c,c), (d,d), (a,b), (a,d), (a,c), (b,c)\}.$$

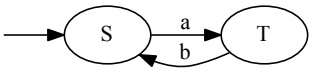
Example 2. If an action occurs multiple times in a transition system it will also have multiple corresponding events. Events can have different labels even if the transitions they correspond to have identical labels. In this example the action a occurs multiple times which results in it being modelled by multiple events. The transition system



can be represented by an elementary event structure $(E, \leq)$ where

$$E = \{a_1, a_2, b\} \text{ and } \leq = \{(a_1, a_1), (a_2, a_2), (b, b), (a_1, b), (b, a_2), (a_1, a_2)\}.$$

Example 3. Transition systems containing loops require unfolding to translate them to event structures. The transition system



can be unfolded to

and an elementary event structure $(E, \leq)$ representing it is given by

$$E = \{a_1, b_1, a_2, b_2, ...\} \text{ and } \leq = \{(a_1, b_1), (b_1, a_2), (a_2, b_2), ...\} \cup \{(e, e) | e \in E\}.$$

In this example the set of events as well as the causal dependency relation are infinite.

The behaviour in these two examples results from the fact that events need to have a fixed place and time in a computational process to establish a causal dependency between them. Properly transforming a model to a corresponding event structure representation might not always be straightforward.

Definition 4. Let $(E, \leq)$ be an elementary event structure. A subset $X \subseteq E$ is **left-closed** if for all events $e_1, e_2 \in E$

$$e_2 \in X \wedge e_1 \leq e_2 \Rightarrow e_1 \in X.$$

For any event e in it, a left-closed set of events must also contain all events that e causally depends on.

Definition 5. Let $(E, \leq)$ and be an elementary event structure. For any $e \in E$ we define the set $\lceil e \rceil := \{e' \mid e' \leq e\}$. Intuitively, $\lceil e \rceil$ is the set of events that must have occurred for event e to happen.

Proposition 1. Let $(E, \leq)$ be an elementary event structure. Then $\lceil e \rceil$ is left-closed for any $e \in E$.

Proof. Let $e' \in \lceil e \rceil$, then $e' \leq e$ by definition of $\lceil e \rceil$. For any $e'' \leq e'$ it follows from the transitivity of $\leq$ that $e'' \leq e$ and hence $e'' \in \lceil e \rceil$. □

Example 4. Since an elementary event structure is just defined as a partial order, one can view $(\mathbb{N}, \leq)$ as an event structure. For any $n \in \mathbb{N}$, the set $\lceil n \rceil$ is the set of numbers up to n with 0 being the smallest element.

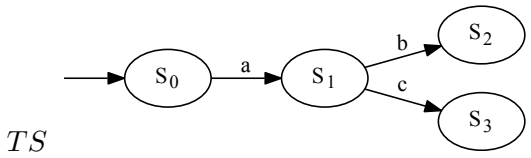
Example 5. $(\mathbb{R}^+ \backslash \{0\}, \leq)$ is an event structure. For any $r \in \mathbb{R}^+ \backslash \{0\}$, the set $\lceil r \rceil$ is infinite and only has an infimum, not a minimum.

The above example illustrates that the definition of an elementary event structure as a partial order allows that an event may require an infinite set of events to have occurred before. This of course contradicts the intuitive understanding of a computational process and leads to the following definition.

Definition 6. An event structure $(E, \leq)$ satisfies the **axiom of finite causes** if and only if for all $e \in E$ the set $\lceil e \rceil$ is finite.

Limitations of elementary event structures Non-determinism and branching in a computational process introduce conflicting behaviours that cannot be properly modelled by an elementary event structure. The following example will illustrate one of their problems.

Example 6. This type of conflict can already be observed in Example 1.

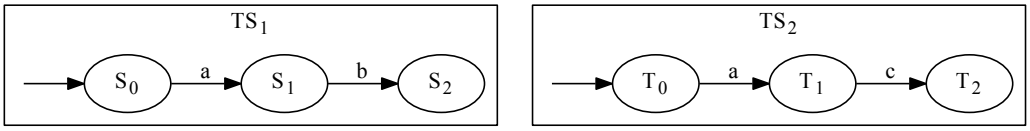


TS

In this transition system the actions b and c are conflicting in the sense that they can not appear in a single execution of a computational process together. They exclude each other and hence are unrelated. An elementary event structure $\mathcal{E} = (E, \leq)$ representing this transition system is given by

$$E = \{a, b, c\} \qquad \leq = \{(a, b), (a, c), (a, a), (b, b), (c, c)\}$$

But two unrelated events could also be concurrent. For two transition systems TS_1 and TS_2, let their parallel composition $TS_1 \parallel_{\{a\}} TS_2$ synchronizes on action a. This intuitively means transitions labelled a can only be taken by TS_1 and TS_2 at the same time, while the other transitions are not restricted in this way.



The event structure representation of the parallel composition $TS_1 \parallel_{\{a\}} TS_2$ is isomorphic to $\mathcal{E}$, the event representation of TS. There is no way to distinguish those two reasons for events being unrelated with elementary event structures. To overcome this problem Winskel defines prime event structures. Another limitation, which applies to elementary and prime event structures, is shown in Examples 9 and 10.

2.2 Prime Event Structures

Prime even structures as defined in [W89] extend elementary event structures with a way to express conflicting events. By adding a relation that identifies conflicting events, prime event structures can distinguish between conflicting and concurrent events.

Definition 7 ([W89])**.** A **prime event structure** is a tuple $(E, \leq, \#)$ consisting of

- a set of events E
- the causal dependency relation $\leq$
- the conflict relation $\#$, a binary relation on $E \times E$

that satisfies the following properties :

- $(E, \leq)$ is a partial order
- Axiom of finite causes, i.e. $\lceil e \rceil$ is finite for any $e \in E$
- $\#$ is irreflexive and symmetric

- Consistency of $\#$, i.e. $\forall e_1, e_2, e_3 \in E \,.\, e_1 \# e_2 \wedge e_2 \leq e_3 \;\Rightarrow\; e_1 \# e_3$

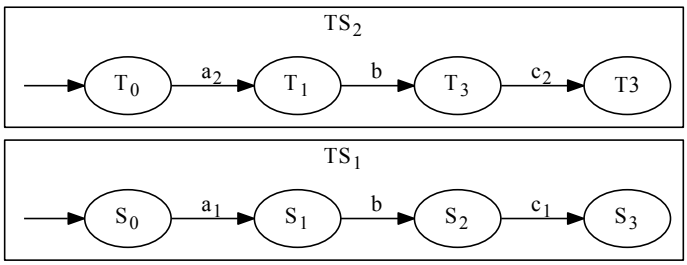
For any $e_1, e_2 \in E$ the fact that $e_1 \# e_2$ means that the events e_1 and e_2 are in conflict and cannot appear in the same computational history. The consistency of $\#$ ensures that events depending on an event e are in conflict with the events that e is in conflict with.

The conflict relation enables the distinction between concurrency and conflict, which is not possible with elementary event structures. Conflicting events are unrelated by $\leq$ and related by $\#$ while concurrent events are unrelated by both $\leq$ and $\#$. The consistency of $\#$ ensures that the distinction between conflict and concurrency does properly propagate to all events depending on conflicting events.

Example 7. A corresponding prime event structure $(E, \leq, \#)$ for the transition system in Example 6 is given by

$$E = \{a, b, c\} \;,\; \leq = \{(a,b), (a,c), (a,a), (b,b), (c,c)\} \;\text{ and }\; \# = \{(b,c), (c,b)\}.$$

Example 8. This example shows how concurrent processes with a synchronization action can be modelled with prime event structures. It shows two transition systems TS_1 and TS_2. Their parallel composition $TS_1 \parallel_{\{b\}} TS_2$ synchronizes on action b, which means transitions labelled b can only be taken by TS_1 and TS_2 at the same time while the other transitions are not restricted in this way.



A prime event structure representing the parallel composition $TS_1 \parallel_{\{b\}} TS_2$ is as follows:

$$E = \{a_1, a_2, b, c_1, c_2\} \;,\; \# = \emptyset \;\text{ and}$$

$$\leq = \{(a_1,b), (a_2,b), (b,c1), (b,c_2), (a_1,c_1), (a_2,c_2), (a_1,c_2), (a_2,c_1)\} \cup \{(e,e) \mid e \in E\}.$$

Note that the pair a_1 and a_2 as well as the pair c_1 and c_2 are unrelated and not in conflict. There is no order required for them since it does matter which one happens first. They are concurrent. The synchronization on event b only requires that both events a_1 and a_2 happen before events c_1 and c_2 happen. This example illustrates how event structures are a model of true of concurrency where interleaving is not required.

To model computational progress with event structures, Winskel defines in [W89] the configurations of a prime event structure.

Definition 8. A **configuration** of a prime event structure $(E, \leq, \#)$ is a subset $X \subseteq E$ such that

- X is conflict-free : $(X \times X) \cap \# = \emptyset$ and
- X is left-closed.

Intuitively, a configuration can be thought of as the computational history of a single run of a process up to a certain point. It cannot contain conflicting events and it has to contain all the events that lead to the current computational state.

Proposition 2. For any $e \in E$ the set $\lceil e \rceil$ is a configuration.

Proof. The left-closedness of $\lceil e \rceil$ was shown in Proposition 1. It remains to show the conflict-freedom. Let $e', e'' \in \lceil e \rceil$, $e' \neq e''$ and assume $e' \# e''$. Since $e'' \in \lceil e \rceil$ implies $e'' \leq e$, by consistency of $\#$ we have

$$e' \# e'' \wedge e'' \leq e \;\Rightarrow\; e' \# e$$

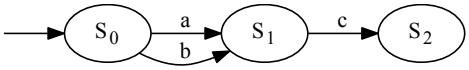
and by symmetry of $\#$ also $e \# e'$. Since $e' \in \lceil e \rceil$ implies $e' \leq e$ using the consistency of $\#$ yields

$$e \# e' \wedge e' \leq e \;\Rightarrow\; e \# e\,.$$

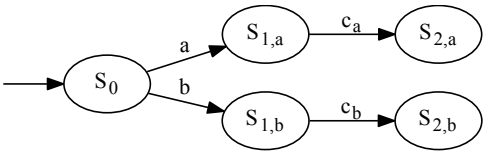
This is a contradiction to the irreflexivity of $\#$. □

Limitations of prime event structures Prime event structures can model the type of conflict in Example 6. The partial order that they are based on still causes problems in the case of an event which has no unique "predecessor" but can be "enabled" by any event in a set of events. The following examples will illustrate the problem.

Example 9. In this example there is a non-deterministic choice between event a and event b, a and b are in conflict. For c to happen it is only required that either a has happened or b has happened. There is no specific requirement for a, so $a \leq c$ does not hold. But there is no specific requirement for b either, so $b \leq c$ does not hold either.



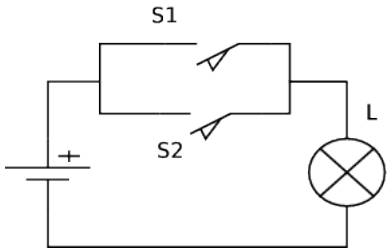
Since a and b are in conflict, a possible solution is to unfold the non-deterministic behaviour and to rename events.



Leaving out symmetry of $\#$ and reflexivity of $\leq$, the resulting prime event structure is given by:

$$E = \{a, b, c_a, c_b\}$$
$$a \# b \quad c_a \# c_b \quad a \# c_b \quad b \# c_a$$
$$a \leq c_a \quad b \leq c_b$$

Example 10. This example taken from [W87] shows an electric circuit with two open switches that can turn on a light.



Representing this as a set of events yields $E = \{S1, S2, L\}$ where S1 is "switch one is closed", S2 is "switch two is closed" and L is "the light turns on". S1 and S2 are obviously not conflicting, but still neither $S1 \leq L$ nor $S2 \leq L$.

The behaviour shown in these two examples can be modelled with the general event structures defined in the next section.

2.3 Event Structures - A more general approach

Winskel introduced a more general version of event structures in [W87]. His goals were to overcome the problems with events that are not enabled in a unique manner as well as simplifying constructions on event structures. The causal dependency relation, a partial order, is replaced by an enabling relation . This enabling relation relates an event e to a set of events that enables e. The binary conflict relation is replaced by the consistency predicate to provide a more general model of conflict. The consistency predicate contains all conflict-free finite subsets of a set of events.

Definition 9 ([W87])**.** A **event structure** is a tuple $(E, Con, \vdash)$ consisting of

- a set of events E,
- the consistency predicate $Con \subseteq 2^E$, the set of conflict-free finite subsets of E, and
- the enabling relation $\vdash \subseteq Con \times E$

which satisfies the following properties

- consistency of Con : $\forall X, Y \subseteq E \,.\, X \in Con \wedge Y \subseteq X \Rightarrow Y \in Con$ and
- $\forall e \in E. \forall X, Y \subseteq E \,.\, X \vdash e \wedge X \subseteq Y \in Con \Rightarrow Y \vdash e$, that is, if X enables e so does any conflict-free superset Y of X.

Note that Con is always non-empty since $\emptyset$ and $\{e\}$ for $e \in E$ are conflict-free and hence $\emptyset \in Con$ and $\forall e \in E. \{e\} \in Con$. The requirement that all sets in Con are finite is the general event structure counterpart to the axiom of finite causes for prime event structures. One can define a consistency predicate Con from a binary conflict relation $\#$:

$$Con = \{X \subseteq_{finite} E \mid \forall e_1, e_2 \in X. (e_1, e_2) \notin \#\}.$$

Intuitively $X \vdash e$ means that the events of X enable event e. While X needs to be conflict-free, X doesn't need to be a minimal set enabling e, it just needs to contain a subset of events that enables e.

Example 11. A general event structure representation $\mathcal{E} = (E, Con, \vdash)$ for the circuit in Example 10 is given by :

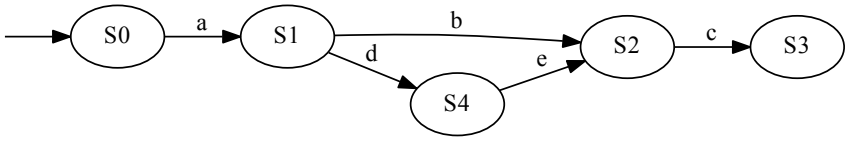
$$E = \{S1, S2, L\}$$

$$Con = \{\emptyset, \{S1\}, \{S2\}, \{L\}, \{S1, L\}, \{S2, L\}, \{S1, S2\}, \{S1, S2, L\}\}$$

$$\vdash: \quad \emptyset \vdash S1 \quad \emptyset \vdash S2 \quad \{S1\} \vdash L \quad \{S2\} \vdash L \quad \{S1, S2\} \vdash L$$

For simplicity, only a subset of the enabling relation is given. For any $X \vdash e$, any conflict free superset $Y \supseteq X$ also enables e.

Example 12. This example shows a transition system which has two conflicting branches $S_1 \xrightarrow{b} S_2$ and $S_1 \xrightarrow{d} S_4 \xrightarrow{e} S_2$.



A general event structure $(E, Con, \vdash)$ representing this transition system is given by :

$$E = \{a, b, c, d, e\}$$

$$Con = \{\ \emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{a, b\}, \{a, c\}, \{a, d\}, \{a, e\}, \{b, c\}, \{c, d\}, \{c, e\}, \{d, e\},$$
$$\{a, b, c\}, \{a, c, d\}, \{a, c, e\}, \{a, d, e\}, \{a, c, d, e\}\ \}$$

$$\vdash :\ \emptyset \vdash a\ ,\ \{a\} \vdash b\ ,\ \{a\} \vdash d\ ,\ \{a, d\} \vdash e\ ,\ \{a, b\} \vdash c\ ,\ \{a, d, e\} \vdash c$$

Again, for simplicity, only a subset of the enabling relation is given. For any $X \vdash e$, any conflict free superset $Y \supseteq X$ also enables e.

Definition 10. A **configuration** of an event structure $(E, Con, \vdash)$ is a subset $C \subseteq E$ that satisfies the following properties

- C is consistent, all finite subsets of C are conflict-free : $\forall X \subseteq_{finite} C.X \in Con$
- C is secured : $\forall e \in C. \exists e_1, ..., e_n \in C. e_n = e \wedge \forall 1 \leq i \leq n. \{e_1, ..., e_{i-1}\} \vdash e_i$

The secured-ness property means that for any event e in a configuration, the configuration has as subsets a sequence of configurations $\emptyset\ ,\ \{e_1\} \ldots \{e_1, .., e_n\}$, called a securing for e in C, such that one can build a "chain of enablings" [W89] that enables e just from $\emptyset$.

$$\emptyset \vdash e_1\ ,\ \{e_1\} \vdash e_2, ..., \{e_0, ..., e_{n-1}\} \vdash e_n = e$$

A configuration of an event structure can be understood as the history of a computation up to a certain computation state. This history can of course not contain conflicting events, which is why consistency is a requirement in the definition of a configuration. The secured-ness property ensures that it a chain of enablings without any conflicts.

Definition 11. For a general event structure $\mathcal{E} = (E, Con, \leq)$, Winskel denotes the set of all configurations of $\mathcal{E}$ as $\mathcal{F}(\mathcal{E})$.

Example 13. The complete set of configurations of the event structure in Example 12 is

$$\{\emptyset\ ,\ \{a\}\ ,\ \{a, b\}\ ,\ \{a, b, c\}\ ,\ \{a, d\}\ ,\ \{a, d, e\}\ ,\ \{a, c, d, e\}\}.$$

Example 14. The configurations of the circuit of Examples 10,11 are given by

$$\{\emptyset\ ,\ \{S1\}\ ,\ \{S2\}\ ,\ \{S1, S2\}\ ,\ \{S1, L\}\ ,\ \{S1, L\}\ ,\ \{S1, S2, L\}\}.$$

A partial order on configurations as computational progress Subset inclusion can be used as a partial order on the set of configurations $\mathcal{F}((E, Con, \vdash))$ of an event structure. This partial order on configurations can be understood as a model of computational progress of a process. A configuration C_1 being a subset of another configuration C_2 means that the state of computation in C_2 can be reached from the state of computation in C_1. C_1 is an earlier state of computation for C_2 and C_2 is a possible future state of computation of C_1. The computation starts in $\emptyset$.

Example 15. For the configurations in Example 13, this yields

$$\emptyset\ \subseteq\ \{a\}\ \subseteq\ \{a, b\}\ \subseteq\ \{a, b\}\ \text{and}$$

$$\emptyset\ \subseteq\ \{a\}\ \subseteq \{a, d\}\ \subseteq\ \{a, d, e\}\ \subseteq\ \{a, c, d, e\}.$$

From $\{a\}$ on, the configurations are not related any more by subset inclusion since the computations take a different branch.

2.4 Stable event structures

Stable event structures are subset of general event structures. They describe processes whose events are enabled in a unique way inside a single computational history. More precisely: If an event is enabled in several ways then those are conflicting. This notion of stable event structures allows branching as a reason for a non-unique enabling but not a "parallel" cause like the two switches in the electric circuit example.

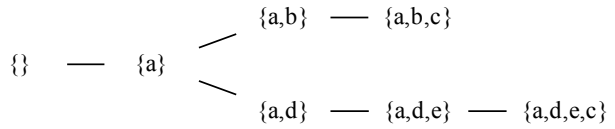
Definition 12. An event structure $(E, Con, \vdash)$ is a **stable event structure** if it satisfies the **axiom of stability**

$$\forall X, Y \in Con, e \in E \,.\, X \vdash e \wedge Y \vdash e \wedge X \cup Y \cup \{e\} \in Con \;\Rightarrow\; X \cap Y \vdash e.$$

If sets of events X and Y both enable event e, and their union including e is still conflict-free, then X and Y contain a common set of events that enables e. This implies that in a configuration of a stable event structure each event is enabled in a unique way.

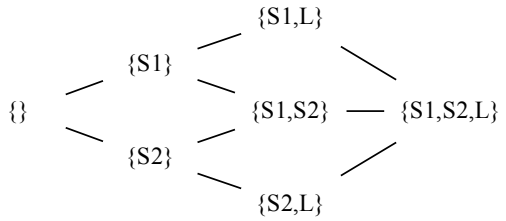
When checking if an event structure is stable it is helpful to consider the structure of its set of configurations. If the axiom of stability is not satisfied this will be witnessed by two configurations because a least set enabling an event will always be a configuration.

Example 16. The configurations of the transition system in Examples 12 and 13 have the form



The two configurations of interest are $\{a, b\}$ and $\{a, d, e\}$ which both enable c. The event structure is stable because $\{a, b\} \cup \{a, d, e\} \cup \{c\} \notin Con$. Hence it is not required that $\{a, b\} \cap \{a, d, e\} = \{a\}$ enables c.

Example 17. The configurations of the circuit in Examples 10,11, and 14 have the form



The configurations of interest in this example are $\{S1\}$ and $\{S2\}$ which both enable L. The event structure is not stable because $\{S1\} \cup \{S2\} \cup \{L\} \in Con$ yet $\{S1\} \cap \{S2\} = \emptyset \nvdash L$.

One can define a partial order on the events of a configuration of a stable event structure.

Definition 13. Let $\mathcal{E} = (E, Con, \vdash)$ be a stable event structure, let $C \in \mathcal{F}(\mathcal{E})$ be a configuration of $\mathcal{E}$. For $e_1, e_2 \in C$ define $\leq_C$ by

$$e_1 \leq_C e_2 \quad \Leftrightarrow \quad \forall B \in \mathcal{F}(\mathcal{E}) \,.\, e_2 \in B \wedge B \subseteq C \Rightarrow e_1 \in B$$

The relation $\leq_C$ imposes an order of causal dependency on the events of each configuration of a stable event structure. This is similar to the partial order that elementary and prime event structures impose on their sets of events.

Proposition 3. Let $\mathcal{E} = (E, Con, \vdash)$ be a stable event structure, let $C \in \mathcal{F}(\mathcal{E})$ be a configuration of $\mathcal{E}$. Then $\leq_C$ is a partial order on C.

Proof. For $\leq_C$ to be a partial order on C it needs to be reflexive, transitive and antisymmetric.

- reflexivity : $\forall e \in C. e \leq_C e$ is obviously valid.
- transitivity : Let $e_1, e_2, e_3 \in C$, let $e_1 \leq_C e_2$ and $e_2 \leq_C e_3$. The subset inclusion $\subseteq$ on the set of configurations $\mathcal{F}(\mathcal{E})$ is a partial order. Then by the transitivity of $\subseteq$ it follows that $e_1 \leq_C e_3$ and hence the transitivity of $\leq_C$.
- antisymmetry : Let $e, e' \in C$ and $e \leq_C e'$ and $e' \leq_C e$. The definition of $\leq_C$ now implies that for any configuration $C' \in \mathcal{F}(\mathcal{E})$ such that $C' \subseteq C$, we have $e \in C' \Leftrightarrow e' \in C'$.

  Since C is a configuration, there exists a securing $e_1, ..., e_n, e$ with $n \geq 0$ for e in C such that $B := \{e_1, ..., e_n\} \vdash e$, where $e \notin B$. B is a configuration since $B \subseteq C$ is conflict-free by the consistency of Con and B is secured by the chain of enabling it gives for e.

  Assuming $e \neq e'$, then either $e' \in B$, which implies that B is a configuration that does contain e' but not e, or $e' \notin B$, which implies that $B' := B \cup \{e\}$ is a configuration that contains e but not e'.

  In both cases this contradicts above implication that every configuration $C' \subseteq C$ either contains both e and e' or none of them. Hence $e = e'$ which implies that $\leq_C$ is antisymmetric.

□

Relation between stable and prime event structures As the reader might have noticed there is strong resemblance between stable event structures and prime event structures. They are similar in the sense that they require a unique enabling of an event.

There exists a variety of slightly different event structure definitions already in the references for this paper. For example one can define prime event structures with a consistency predicate instead of a binary conflict relation as $(E, Con, \leq)$. In the same way one can define general event structures with the less general binary conflict relation as $(E, \#, \vdash)$.

By doing one or the other it is possible to transform a stable event structure into a prime event structure. This requires "unfolding" non-determinism and relabelling events as seen in Example 9. This obviously changes the set of events and the set of configurations. Winskel managed to prove though that the configurations of a stable event structures and those of its corresponding prime event structure are isomorphic [W87]. While this is not as as good as equivalence of stable event structures and prime event structures, it still allowed him to perform many constructions on stable instead of prime event structures. While possible, he considered most constructions "clumsy" and too complicated when applied to prime event structures. Such constructions include parallel composition, non-deterministic sum and restriction of event structures.

3 Constructions on event structures

To model communicating processes with event structures, several constructions on event structures are required. The two most important are the sum of event structures, which expresses branching and the product of event structures which models concurrency. To define those, one needs a way to express synchronisation between events of event structures. Because of the types of processes and events usually represented by other models of concurrency, Winskel restricts all these constructions on stable event structures.

3.1 Synchronisation by morphisms on event structures

In many models of communicating processes, the calculus of communicating systems(CCS) for example, one distinguishes between the internal actions of process and actions that are synchronized between several (two in the case of CCS) processes. To express this behaviour with event structures, one has to distinguish between internal events of an event structure and "events of synchronization" [W87], i.e. events that are synchronized with events of another event structure. Winskel introduces a morphism on stable event structures to express this synchronization.

Since usually not all events of a process are synchronized with events of another process, such a morphism is usually only a partial function. It is only defined for the subset of events that is synchronized between processes.

Notation In the following, let $\mathcal{E}_0 = (E_0, Con_0, \vdash_0)$ and $\mathcal{E}_1 = (E_1, Con_1, \vdash_1)$ be stable event structures. $e \in E_0$ denotes an event of $\mathcal{E}_0$ and $X \subseteq E_0$ a set of events of $\mathcal{E}_0$.

- The expression $\Theta : E_0 \to_* E_1$ means that Θ is a partial function mapping events of $\mathcal{E}_0$ to events of $\mathcal{E}_1$. The function Θ is only defined for a subset of E_0, i.e. those events of $\mathcal{E}_0$ which are synchronization events and have a corresponding event in $\mathcal{E}_1$.
- For a set $X \subseteq E_0$ the set $\Theta X := \{\Theta(e) \,|\, e \in E_0 \wedge \Theta(e) \text{ is defined } \}$ is the image of X under Θ, i.e. the set of events of $\mathcal{E}_1$ that X is mapped to. Since Θ is only a partial function the cardinality of $|\Theta X|$ can be less than $|X|$ or possibly $\Theta X = \emptyset$.
- For the sake of clarity, in the following definitions the occurrence of $\Theta(e)$ for an event e in any statement implies that Θ is indeed defined for event e.

Morphisms between event structures are structure-preserving mappings from one event structure to another. They should preserve important properties of event structures.

Definition 14 ([W87])**.** Let $\mathcal{E}_0 = (E_0, Con_0, \vdash_0)$ and $\mathcal{E}_1 = (E_1, Con_1, \vdash_1)$ be stable event structures. A **partially synchronous morphism** from $\mathcal{E}_0$ to $\mathcal{E}_1$ is a partial function $\Theta : E_0 \to_* E_1$ on events which satisfies the following properties

(1) Θ preserves consistency: $X \in Con_0 \Rightarrow \Theta X \in Con_1$

(2) Θ preserves stability: $\{e, e'\} \in Con_0 \wedge \Theta(e) = \Theta(e') \Rightarrow e = e'$

(3) Θ preserves the enabling relation: $X \vdash_0 e \wedge \Theta(e)$ is defined $\Rightarrow \Theta X \vdash_1 \Theta(e)$

Property (1) ensures that the image ΘX of a consistent set X of events under a synchronous morphism Θ is consistent. Property (2) means only conflicting events of $\mathcal{E}_0$ can be mapped to the same event in $\mathcal{E}_1$. Property (3) ensures that for a set X enabling an

event e in $\mathcal{E}_0$, the image $\Theta(X)$ enables the event $\Theta(e)$ that e is mapped to in $\mathcal{E}_1$, provided that $\Theta(e)$ is defined. If $\Theta(e)$ is not defined, meaning that e is no synchronization event, then there exists no corresponding enabling in $\mathcal{E}_1$.

Θ is called a **synchronous morphism** if it is a total function, i.e. if Θ is defined for all events $e \in E_0$. Then all events of event structure $\mathcal{E}_0$ are synchronization events. Note that $\mathcal{E}_1$ can have still unsynchronized events since Θ being a global function does not imply that it is surjective.

Proposition 4. Morphisms on stable event structures preserve configurations. For stable event structures $\mathcal{E}_0 = (E_0, Con_0, \vdash_0)$ and $\mathcal{E}_1 = (E_1, Con_1, \vdash_1)$ and a partially synchronous morphism $\Theta : E_0 \to_* E_1$ it holds that

$$C \in \mathcal{F}(\mathcal{E}_0) \;\Rightarrow\; (\Theta C \in \mathcal{F}(\mathcal{E}_1) \wedge \forall e, e' \in C . \Theta(e) = \Theta(e') \Rightarrow e = e')$$

Proof. Let $C \in \mathcal{F}(\mathcal{E}_0)$ be a configuration. First, one needs to prove that ΘC is a configuration, i.e. that ΘC is consistent and secured.

- Since C is a configuration, C is consistent, i.e. all finite subsets X of C are conflict-free: $\forall X \subseteq_{finite} C . X \in Con_0$. Hence by property (1) in Definition 14 it follows that $\forall X \subseteq_{finite} C \,.\, \Theta X \in Con_1$. Since every finite subset of $Y \subseteq \Theta C$ is the image of a finite subset $X \subseteq C$, i.e. $\forall Y \subseteq_{finite} \Theta C \,.\, \exists X \subseteq_{finite} C \;.\; Y = \Theta X$, this implies that all finite subsets of ΘC are conflict-free: $\forall Y \subseteq_{finite} \Theta C . Y \in Con_1$. Hence ΘC is consistent, too.

- For any event $e' \in \Theta C$ there is a corresponding event $e \in C$ such that $e' = \Theta(e)$. Because C is a configuration, there exists a securing $X := \{e_1, ..., e_n, e\}$ with $n \geq 0$ for every e in C such that $\emptyset \vdash e_1 \,, \{e_1\} \vdash e_2 \,, \ldots \,, \{e_1, ..., e_n\} \vdash e$. Then by property (3) in Definition 14, $\Theta(X)$ is a securing for $\Theta(e)$ with $\emptyset \vdash e_1 \,, \Theta\{e_1\} \vdash e_2 \,, \ldots \,, \Theta\{e_1, ..., e_n\} \vdash e$, provided that Θ is defined all e_i. If this is not the case, the corresponding step in the chain of enablings is just skipped. This implies that ΘC is secured.

It remains to show the second part of the implication. Since C is a configuration $\{e, e'\} \in Con_0$ for any $e, e' \in C$. Then by property (2) in Definition 14 it follows that $\Theta(e) = \Theta(e') \Rightarrow e = e'$.

□

3.2 Product and sum of stable event structures

With these morphisms on event structures Winskel defines the product and sum of event structures. The morphisms provide the semantics for the constructions by a mapping between the construction and its components. These mappings are (partially) synchronous morphisms synchronizing events of the construction with corresponding events of the components.

Notation For the product of event structures we introduce a special unsynchronized event $*$. For a function Θ and an event e the expression $\Theta(e) = *$ means that the function Θ is not defined for the the event e because e has no corresponding synchronized event. Occurrence of $*$ in an event tuple $(*, e)$ or $(e, *)$ will indicate that $(*, e)$ or $(e, *)$ is only synchronized with one of the component systems, while tuples of type (e_0, e_1) indicate synchronization with both components.

Definition 15. Let $\mathcal{E}_0 = (E_0, Con_0, \vdash_0)$ and $\mathcal{E}_1 = (E_1, Con_1, \vdash_1)$ be stable event structures. The **partially synchronous product** $\mathcal{E}_0 \times \mathcal{E}_1 := (E, Con, \vdash)$ where E, Con and $\vdash$ are defined as follows:

- The set of events E is given by

$$E = \{(e_0, *) \mid e_0 \in E_0\} \cup \{(*, e_1) \mid e_1 \in E_1\} \cup \{(e_0, e_1) \mid e_0 \in E_0 \wedge e_1 \in E_1\}$$

- The projections π_0 and π_1 are morphisms from $\mathcal{E}_0 \times \mathcal{E}_1$ to $\mathcal{E}_0$ and $\mathcal{E}_1$ such that
  - $\pi_0 : E \to_* E_0$ is defined by $\pi_0(e_0, e_1) = e_0$ and
  - $\pi_1 : E \to_* E_1$ is defined by $\pi_1(e_0, e_1) = e_1$
- The consistency predicate is given by : $X \in Con$ if and only if
  - $X \subseteq_{finite} E$ and
  - $\pi_0 X \in Con_0 \ \wedge \ \pi_1 X \in Con_1$ and
  - $\forall e, e' \in X \,.\, \pi_0(e) = \pi_0(e') \vee \pi_1(e) = \pi_1(e') \Rightarrow e = e'$
- The enabling relation $\vdash$ is given by : $X \vdash e$ if and only if
  - $X \in Con, e \in E$ and
  - $\pi_0(e)$ is defined $\Rightarrow \pi_0 X \vdash_0 \pi_0(e)$ and
  - $\pi_1(e)$ is defined $\Rightarrow \pi_1 X \vdash_0 \pi_1(e)$

The morphisms π_1, π_2 are partially synchronous morphisms mapping events of the product event structure to its components. Events of type (e_0, e_1) are synchronized with e_0 by π_0 and with e_1 by π_1. Events of type $(e_0, *)$ are only synchronized with e_0 by π_0 and events of type $(*, e_1)$ are only synchronized with e_1 by π_1.

A **synchronous product** can be defined in a similar way by restricting the set of events to the synchronized ones $\{(e_0, e_1) \mid e_0 \in E_0 \wedge e_1 \in E_1\}$ and defining consistency predicate and enabling relation only on this restricted set of events. The projections π_i are then total functions and thus synchronous morphisms.

Example 18. Let $\mathcal{E}_0 = (E_0, Con_0, \vdash_0)$ and $\mathcal{E}_1 = (E_1, Con_1, \vdash_1)$ be two stable event structures with events and consistency predicate given by

$$E_0 = \{a_0, b_0\} \quad Con_0 = 2^{E_0} \qquad E_1 = \{a_1, b_1\} \quad Con_1 = 2^{E_1}$$

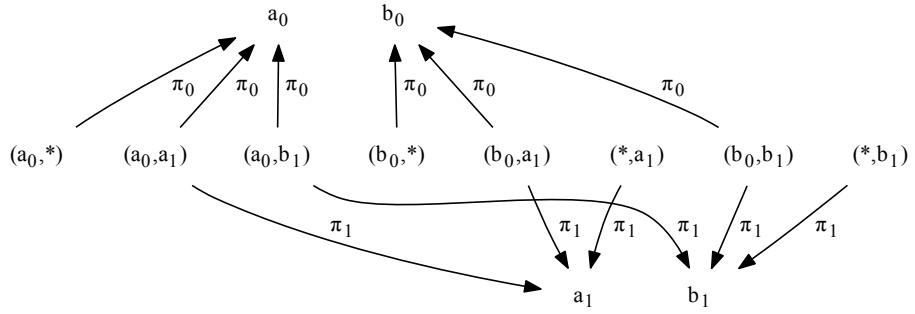
and enabling relation the least one that contains

$$\emptyset \vdash_0 a_0 \quad \{a_0\} \vdash_0 b_0 \qquad \emptyset \vdash_1 a_1 \quad \{a_1\} \vdash_1 b_1$$

Then for the partially synchronous product $\mathcal{E}_0 \times \mathcal{E}_1 = (E, Con, \vdash)$, the set of events is given by :

$$E = \{(a_0, *), (a_0, a_1), (a_0, b_1), (b_0, *), (b_0, a_1), (b_0, b_1), (*, a_1), (*, b_1)\}$$

The partially synchronous morphisms π_0, π_1 are given by the following mappings :



The consistency predicate is $Con = 2^E$ and the enabling relation is the least one that contains

$$\emptyset \vdash (a_0, *) \qquad \emptyset \vdash (*, a_1) \qquad \emptyset \vdash (a_0, a_1)$$
$$\{(a_0, *)\} \vdash (b_0, *) \qquad \{(a_0, a_1)\} \vdash (b_0, *)$$
$$\{(*, a_1)\} \vdash (*, b_1) \qquad \{(a_0, a_1)\} \vdash (*, b_1)$$
$$\{(a_0, a_1)\} \vdash (b_0, b_1) \qquad \{(a_0, *), (*, a_1)\} \vdash (b_0, b_1)$$

Note that the following enablings are not in $\vdash$:

$$\{(a_0, *)\} \nvdash (*, b_1) \qquad \{(*, a_1)\} \nvdash (b_0, *) \qquad \{(a_0, *)\} \nvdash (b_0, b_1) \qquad \{(*, a_1)\} \nvdash (b_0, b_1)$$

This is because π_0 is undefined for events of type $(*, e)$. Hence the image $\pi_0\{(*, a_1)\} = \emptyset$ and $\emptyset$ enables only the event a_0 in $\mathcal{E}_0$. The same argument holds for events of type $(e, *)$ and π_1.

Definition 16. Let $\mathcal{E}_0 = (E_0, Con_0, \vdash_0)$ and $\mathcal{E}_1 = (E_1, Con_1, \vdash_1)$ be stable event structures. The **sum** $\mathcal{E}_0 + \mathcal{E}_1$ of $\mathcal{E}_0$ and $\mathcal{E}_1$ is the event structure $(E, Con, \vdash)$ where

- the set of events E is the disjoint union of E_0 and E_1

$$E = \{(0, e_0) \mid e_0 \in E_0\} \cup \{(1, e_1) \mid e_1 \in E_1\}$$

- injections (injective morphisms) $\imath_0, \imath_1$ are given by
  - $\imath_0 : E_0 \to E$ such that $\imath_0(e) = (0, e)$ and
  - $\imath_1 : E_1 \to E$ such that $\imath_1(e) = (1, e)$
- the consistency predicate includes all sets X such that
  - $\exists X_0 \in Con_0 . X = \imath_0 X_0$ or
  - $\exists X_1 \in Con_1 . X = \imath_1 X_1$
- the enabling relation is such that for any $X \in Con, e \in E$ $X \vdash e$ if and only if
  - $\exists X_0 \in Con_0, e_0 \in E_0 . X = \imath_0 X_0 \wedge e = \imath_0(e_0) \wedge X_0 \vdash_0 e_0$ or
  - $\exists X_1 \in Con_1, e_1 \in E_1 . X = \imath_1 X_1 \wedge e = \imath_1(e_1) \wedge X_1 \vdash_1 e_1$

The injections $\imath_0, \imath_1$ are synchronous morphisms that map the events of $\mathcal{E}_0, \mathcal{E}_1$ to their corresponding events in the sum event structure. Note that there are no sets of type $\{(0, e_0), (1, e_1)\}$ in the consistency predicate of the sum event structure since every set $Y \in Con$ is the image of a set $X_0 \in Con_0$ or $X_1 \in Con_1$. The sum of event structures models conflicting branches of a computation.

Example 19. Using the event structures $\mathcal{E}_0$ and $\mathcal{E}_1$ from Example 18 the sum $\mathcal{E}_0 + \mathcal{E}_1 = (E, Con, \vdash)$ is given by the set of events

$$E = \{(0, a_0), (0, b_0), (1, a_1), (1, b_1)\},$$

the injections

$$\imath_0 \ : \ a_0 \mapsto (0, a_0) \ , \ b_0 \mapsto (0, b_0) \quad \text{and} \quad \imath_1 \ : \ a_1 \mapsto (1, a_1) \ , \ b_1 \mapsto (1, b_1) \, ,$$

the consistency predicate

$$Con = \{\emptyset \, , \, \{(0, a_0)\} \, , \, \{(0, b_0)\} \, , \, \{(1, a_1)\} \, , \, \{(1, b_1)\} \, , \, \{(0, a_0), (0, b_0)\} \, , \, \{(1, a_1), (1, b_1)\}\} \, ,$$

and the enabling relation the least one that contains the enablings

$$\emptyset \vdash (0, a_0) \quad \emptyset \vdash (1, a_1) \quad \{(0, a_0\} \vdash (0, b_0) \quad \{(1, a_1)\} \vdash (0, b_1) \, .$$

4 Connection to other concepts

As already mentioned in the abstract, Winskel introduced event structures to establish a connection between Petri nets and results from domain theory, specifically Scott Domains. In [W87] he relates event structures to other models of concurrency. These include

- Petri nets. Winskel shows that 1-safe Petri nets, a subset of Petri nets, can be modelled with event structures. He also shows that prime event structures, which are a subset of event structures, can be modelled with Petri nets. Both constructions use occurrence nets, the unfolding of Petri nets, as an intermediate step.
- The CCS and CSP languages. Winskel shows how synchronization between processes can be expressed by morphisms on stable event structures. By defining the operations possible in CCS and CSP for event structures, one can give non-interleaving semantics to CCS and CSP.

5 Conclusion

This paper presents event structures as a mathematical model for concurrent processes. It is shown why a partial order on events is not enough to model all types of process behaviour and why the more complex general event structures model is necessary.

Simple examples for event structures and their configurations offer some intuitive insight about causality in computational processes. Still they already show the difficulties of the model. The examples in this paper illustrate that compact representations of processes like transition systems can quickly blow up in size when modelled by event structures. A single loop already leads to an infinite representation for the simple elementary event structure model. For the general event structures, consistency predicate and enabling relation grow quickly as soon as the process being modelled contains any branching.

While event structures provide a way to relate other models of concurrency, the necessary constructions for general results are far from intuitive and require advanced knowledge in domain theory.

References

[NPW79] Mogens Nielsen, Gordon Plotkin, Glynn Winskel. Petri nets, event structures and domains. In: Lecture Notes in Computer Science, 1979

[W87] Glynn Winskel. Event structures. In Petri Nets: Applications and Relationships to other models of concurrency, volume 255 of Lecture Notes in Computer Science, pages 325 to 392. Springer, 1987.

[W89] Glynn Winskel. An introduction to event structures. In: Proceeding Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop. Springer, 1989

[W09] Glynn Winskel. Events, causality and symmetry. Computer Journal, 54(1): pages 42 to 57, 2009.