

Resource Structures as a Model of Concurrency

Jane Hopkins

Honours Project, 1996

Abstract

In this report, we introduce a model of concurrency based on event structures which we call resource structures. We show how these structures can model concepts relating to resource management such as resource contention, generation and consumption. Various constructors are also presented which are useful for specifying complex resource structures as the composition of several smaller substructures. Like event structures, resource structures have an underlying transition system which can be used to define a notion of equivalence. Using this equivalence, we then present several properties satisfied by the constructors such as commutative, distributive and unit laws.

Contents

- 1 Introduction** **1**

- 2 Preliminaries** **3**
 - 2.1 Event Structures 3
 - 2.1.1 Concurrency 5
 - 2.1.2 Configurations 5
 - 2.2 Transition Systems 6
 - 2.3 Examples 7

- 3 Introducing Resource Structures** **10**
 - 3.1 Formal Definition 10
 - 3.2 Examples 12

- 4 Combining Resource Structures** **15**
 - 4.1 Sequencing 15
 - 4.2 Choice 18
 - 4.3 Parallel Composition 19
 - 4.4 Synchronous Composition 20
 - 4.5 Composite Resource Structures 22

- 5 Properties** **23**
 - 5.1 Properties of Resource Structures 25
 - 5.2 Comparing Resource Structures and Event Structures 31

- 6 Conclusions and Further Work** **33**

Chapter 1

Introduction

As concurrent systems become increasingly popular, so too is the need for formal models to help manage the extra complexity that these systems have over traditional sequential systems. Models of concurrency can be used for the specification of systems, supporting the design phase of a project. They can also be used in the verification of various properties of a concurrent system such as deadlock and reachability of states which leads to an increase in reliability of systems.

A wide variety of models for concurrent systems have been proposed and include Petri nets[Rei85], transition systems[WN94] and event structures[Win86, Win88]. A comparison of different methods of modelling concurrency is presented in [WN94, Che88].

We are interested in developing models for concurrent systems with respect to resource management. Resources in a system may be processors, time, physical memory, messages, files, in fact, anything that a process requires in order to be executed.

Resources are either consumable or reusable. Consumable resources are used once and then become unavailable, such as time. That is, they cannot be reused by another process. Limited resources which can be shared by processes in an interleaving manner are reusable, such as processors or common databases.

In managing these resources, three conditions may arise.

Resource generation: A process, $p1$ is dependent on process $p2$ if $p2$ generates a resource that $p1$ requires. The process $p1$ must wait for $p2$ to occur before it can be executed. Hence resource generation may restrict the order that processes occur.

Resource consumption: Resource consumption occurs when a process requires a consumable resource. Once the process has consumed the required resource, the resource becomes unavailable.

Resource contention: Resource contention arises when multiple processes require the use of a common resource. If the resource is not reusable then resource consumption occurs, otherwise the assignment of the resource to one process would temporarily block the others and so the processes cannot occur together in parallel. Instead one must wait for the other to finish and release the required resource before being able to proceed. Hence resource contention forces processes to be executed in an interleaved manner.

Contention of a consumable resource can be specified by resource consumption, so the resource contention referred to in the remainder of this report refers to contention of a common, reusable resource.

Concurrency can be specified in terms of resources. Two processes can be executed concurrently only if they are not dependent on one another and do not require any common resource. In this report, we present a model of concurrency which can specify concepts related to resource management in terms of concurrency.

Event Structures. Event structures were introduced by Winskel in his thesis[Win80] and are a well known model of concurrency. The behaviour of a system is specified by the possible occurrences of atomic actions, or *events*, where the key focus is on causality and conflict. The order that these events occur is defined by an enabling relation, and nondeterministic choice between event occurrences is represented by a conflict relation.

In the context of resources, the enabling relation can be seen as a method of resource generation where the required resources are generated for the enabled event. The conflict relation forces a choice to be made between two events, where only one of the two events may occur. This can be thought of as resource consumption with the two events both requiring a single, consumable resource. Only a limited form of resource contention can be specified by event structures, in terms of sequencing and choice.

Resource Structures. We observe that the conflict relation of the event structure is not required if we modify the enabling relation.

In this report, we look at how resource generation, consumption and contention can be specified using such resource structures as models of concurrent systems. We will also show how simple resource structures may be combined in various ways to produce more complex resource structures.

The remainder of this paper is organised as follows. Chapter 2 introduces event structures and presents several simple examples. In Chapter 3, we define resource structures and present several examples of how resource structures model resource management as well as concurrency. Chapter 4 presents the constructors used to combine resource structures. In Chapter 5, we define the notion of equivalence on resource structure and present several properties which are satisfied by resource structures. The report is concluded with Chapter 6, in which conclusions and further work are discussed.

Chapter 2

Preliminaries

2.1 Event Structures

Event structures [Win86, Win88] are a model of concurrent processes or systems, where a process is considered to be performing various events over time. An event structure consists of a set of event occurrences together with relations which constrain the order and compatibility of events. What an event is depends on the required level of abstraction. An event may be a complicated process or an indivisible action. However, for modelling purposes, an event is considered to be instantaneous and indivisible—either a system has exhibited an event or it has not. Each event may occur at most once in any run of the process because events of an event structure correspond to event *occurrences* and even if a similar event occurs many times, each occurrence is unique. We define a *computation* of a process as a possible run of the process.

In general the occurrence of one event may depend on the previous occurrence of other events. For example a process may receive an instruction after which it performs an appropriate action. The event of performing the action depends on the process first receiving a corresponding instruction. This causal dependency between events is specified by an *enabling relation* where an event is enabled by a set of events. For general event structures, events may be enabled by more than way. A *prime event structure* is a restricted form of general event structure in which an event can only be enabled in one way. That is, each event has a unique cause. We consider only general event structures in this report.

It is expected that processes may have different behaviour under different circumstances. This notion of non-determinism is reflected by a *conflict relation*, which determines which two events are incompatible and hence exclude each other.

We will use $\mathcal{P}_f(X)$ to denote the set of finite subsets of a set X .

Definition 2.1 A general event structure is a triple $(E, \#, \vdash)$ where

E is a set of event occurrences.

$\# \subseteq E \times E$ is a binary, symmetric, and irreflexive relation called the **conflict relation**. For $(e, f) \in \#$, we write $e \# f$.

$\vdash \subseteq \text{Con} \times E$ is the **enabling relation** where Con is the collection of conflict free subsets of E , i.e.

$$\text{Con} \subseteq \mathcal{P}_f(E) \text{ such that } \forall C \in \text{Con}, \forall e, f \in C, \neg(e \# f)$$

This enabling relation satisfies

$$\text{if } X \vdash e \text{ and } X \subseteq Y \in \text{Con then } Y \vdash e$$

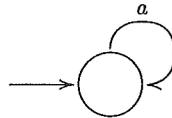
That is, once an event is enabled, it can be disabled only by conflict.

If events e and e' are in conflict then only one of them may occur in any computation. That is, once e has occurred, e' can never occur, and vice versa. Hence, the conflict relation places a constraint on which set of events may occur in any computation of a process. These consistent set of events are in the set Con .

The enabling relation constrains the order in which events occur. An event can only occur if it has been enabled by the occurrence of necessary events. An event may be enabled in one of several ways. For example, if $\{a, b\} \vdash c$ and $\{d, e\} \vdash c$ then the event c may occur only after events a and b or events c and d have been exhibited. Once an event has been enabled, it can always be exhibited after a number of events, provided there is no conflict between the enabled event and the exhibited events.

The behaviour of a process is specified in terms of conflict and enabling of events.

Example. Consider the process which is only capable of exhibiting the action a as shown in the automaton below.



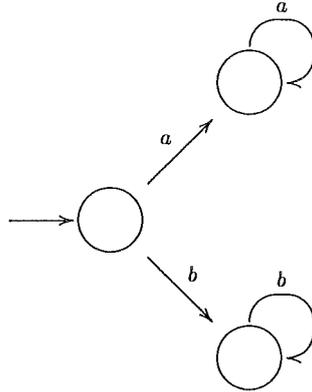
This process can be represented by an event structure with events $E = \{(a, i) \mid i \in \mathbb{N}\}$ where the event (a, i) represents the i th occurrence of a . The occurrence of the event (a, i) depends on the previous $i - 1$ occurrences of a . For example the third occurrence of a corresponds to the event $(a, 3)$ and can only occur after the first and second occurrences of a . This causal dependency between events is specified by the enabling relation

$$\{(a, i) \mid i < n\} \vdash (a, n).$$

We say that (a, i) is enabled by the set of previous occurrences of a . There is no conflict between events because we can have any number of occurrences of a , and therefore the conflict relation is empty.

▽

Example. Let us now consider a similar process which is capable of exhibiting either a 's or b 's as shown in the following automaton.



It is nondeterministic as to whether a sequence of a 's or b 's will be executed. This process can be represented by a similar event structure to the previous example, with events $E = \{(a, i) \mid i \in \mathbb{N}\} \cup \{(b, i) \mid i \in \mathbb{N}\}$. Events are enabled in the same way as the previous event structure. The i th occurrence of an a or b is enabled after the previous $i - 1$ occurrences of an a or b respectively.

$$\{(a, i) \mid i < n\} \vdash (a, n) \text{ and } \{(b, i) \mid i < n\} \vdash (b, n).$$

However, once an a occurs, a b can never occur, and vice versa. This nondeterminism is represented by the conflict relation

$$\forall i, j, (a, i) \# (b, j).$$

Any occurrence of a is in conflict with any occurrence of b .

▽

2.1.1 Concurrency

Concurrency is a derived notion. Two events are considered to be concurrent if they can both occur in a computation in any order. That is to say, neither event causally depends on the other. This situation will arise when both events have been enabled and there is no conflict between them.

Say two events $e, e' \in E$ are *concurrent*, iff

$$\exists C \subset E \text{ such that } C \vdash e, C \vdash e' \text{ and } \neg(e \# e').$$

If events e and e' are concurrent, we write $(e \text{ co } e')$ where the relation $\text{co} \subseteq E \times E$.

2.1.2 Configurations

Given an event structure, we now define a notion of computation state. This is called a *configuration* and is the set of events which have occurred up to some stage in a possible run of the system. Any configuration must be reachable. That is, we must be able to get to the configuration from the initial configuration, \emptyset , where no event has occurred. In order to be reachable, every event in a configuration should have been enabled by the occurrence of previous events, which will also be present in the configuration. We assume that this chain of enablings is finite for any event and so will terminate eventually with events which are enabled by the null set, and so do not require the occurrence of any previous events. It is also required that no two events in conflict may both be present in a valid configuration.

Definition 2.2 Let $E = (E, \#, \vdash)$ be an event structure. Define a configuration of E to be a subset of events $C \subseteq E$ which is

reachable: (referred to as “secured” by Winskel)

$$\forall e \in C, \exists e_0, e_1, \dots, e_n \in C \text{ such that } \emptyset \vdash e_0, \{e_0\} \vdash e_1, \dots, \{e_0, \dots, e_n\} \vdash e$$

conflict-free:

$$\forall e, e' \in C, \neg(e \# e')$$

The set of all configurations of the event structure E is written as $\mathcal{C}(E)$.

2.2 Transition Systems

Transition systems are useful as they can explain the behaviour of a system and have an underlying operational intuition which can be represented graphically. A labelled transition system consists of a collection of states and transitions between states which are labelled with actions.

Definition 2.3 A transition system is a triple $(S, \Sigma, \longrightarrow)$ where

S is a set of states,

Σ is a set of actions, and

$\longrightarrow \subseteq S \times \Sigma \times S$ is the transition relation.

Notation. If $(s, a, s') \in \longrightarrow$, we write $s \xrightarrow{a} s'$ meaning at state s , the action a can be performed to enter the state s' .

The set of configurations of an event structure can be seen as the set of states of a labelled transition system [LPRT95]. At any time a system is in a particular configuration. The system may effect a transition from one configuration to another by performing one or more concurrent events. These transitions between configurations are labelled with the set of events which are concurrently exhibited in the transition. The labels of transitions are sets of events instead of single events because we want to explicitly represent the concurrent exhibition of one or more events in a single step.

Definition 2.4 An event structure, $E = (E, \#, \vdash)$ induces a transition system, $TS = (S, \Sigma, \longrightarrow)$ where

$$S = \mathcal{C}(E)$$

$$\Sigma = \mathcal{P}_f(E)$$

$\longrightarrow \subseteq S \times \Sigma \times S$ is the transition relation. $C \xrightarrow{S} C'$ iff

1. $\forall e \in S, C \vdash e$,
2. $C' = C \cup S$,
3. $C \cap S = \emptyset$ and

4. $S \neq \emptyset$

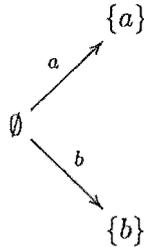
We can exhibit the set of concurrent events S at configuration C if all events in S are enabled from C (requirement 1), no events in S are in conflict and no event in S is in conflict with any event in C . These last two conditions are satisfied because the transition relation requires C' to be a configuration, and hence conflict-free. The configuration obtained after the transition will consist of all the events which occurred in the transition (S), as well as all the events which had previously occurred, (C). This is defined in 2. Requirement 3 ensures that events occur no more than once in any computation and requirement 4 means that we are only considering non-empty transitions.

Transition systems only consider configurations and therefore any unreachable states or transitions are not included. We are only interested in the relevant behaviour of event systems and so will use transition systems to graphically represent such behaviour.

Notation. We use symbols a, b, c, \dots to represent events, C, C_1, C_2, C', \dots to represent configurations and S, S_1, S_2, S', \dots to represent sets of events. As a notational convenience we write $C \xrightarrow{a,b} C'$ instead of $C \xrightarrow{\{a,b\}} C'$. We also give a “core” relation instead of an enabling relation. The enabling relation is the least extension of the core that satisfies the monotonicity requirements. For example the core relation $\emptyset \vdash a, \emptyset \vdash b$ corresponds to the enabling relation $\emptyset \vdash a, \emptyset \vdash b, \{a\} \vdash b, \{b\} \vdash a$.

2.3 Examples

Resource consumption. Consider the event structure, E , with events $E = \{a, b\}$ where $\emptyset \vdash \{a\}, \emptyset \vdash \{b\}$ and $a \# b$. $\mathcal{C}(E) = \{\emptyset, \{a\}, \{b\}\}$. $\{a, b\}$ is not a valid configuration because it is not conflict free. The behaviour of the system is shown in the transition system displayed graphically below. Either an a or a b is exhibited and then the system terminates.



Nondeterminism appears as “branching” in a transition system. In any run of the system, only one branch can be followed. Because the events a and b are in conflict, once a occurs, b can never occur, and vice versa. This can be thought of as resource consumption, where the events a and b both require a common, consumable resource. Hence the conflict relation models resource consumption.

▽

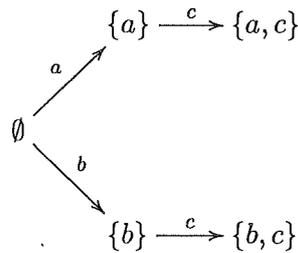
Resource generation. Consider the event structure with events $E = \{a, b, c\}$ where $\emptyset \vdash a, \{a\} \vdash b, \{b\} \vdash c$ and $\# = \emptyset$.

$$\emptyset \xrightarrow{a} \{a\} \xrightarrow{b} \{a, b\} \xrightarrow{c} \{a, b, c\}$$

The empty set enables a which means that a has no prerequisites and can occur at any time in the computation, providing there is no conflict with events already occurred. The event b is enabled by a and so cannot be exhibited until after a has occurred. Similarly c is dependent on b . This forces an ordering on the occurrence of events. This dependency can be thought of as the previous events generating resources that the enabled event requires and hence resource generation can be specified by the enabling relation.

▽

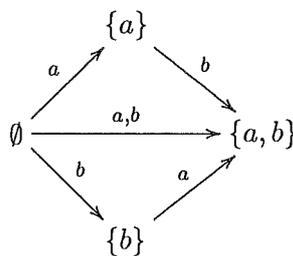
An event can be enabled in more than one way. For example, consider the event structure with events $E = \{a, b, c\}$ where $\emptyset \vdash a, \emptyset \vdash b, \{a\} \vdash c, \{b\} \vdash c$ and $a \# b$.



The event c can either be enabled from a or from b . This example can be viewed as a suspended process where any key must be hit to continue. Events a and b are possible keys which may be hit and c is the continuation.

▽

Concurrency. Consider the event structure, E , with events $E = \{a, b\}$ where $\emptyset \vdash \{a\}, \emptyset \vdash \{b\}$ and $\# = \emptyset$. $C(E) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$.

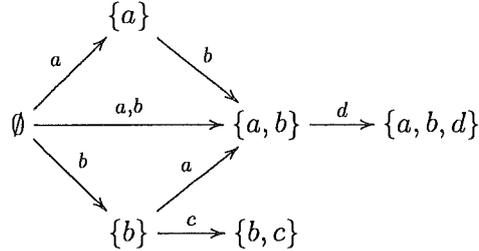


The events a and b are neither causally dependent nor in conflict. Therefore, their behaviour is not restricted in any way and so the two events may occur concurrently in a single step, or their occurrences may be interleaved in a nondeterministic manner. Concurrency of two events appears as a “square” in corresponding transition system. Similarly, three concurrent events will be represented as a “cube”.

Concurrency implies interleaving, hence the concurrency modelled by event structures is “possible” concurrency. That is to say “step” concurrency, the simultaneous execution of concurrent events in one step without interleaving is not directly modelled.

▽

Example. Consider an event structure consisting of the events, $E = \{a, b, c, d\}$ where $\emptyset \vdash a, \emptyset \vdash b, \{b\} \vdash c, \{a, b\} \vdash d$ and $a \# c$. The corresponding transition system is shown below.



In this case, a single run of the system will produce one of the four following possible sequences of configurations: $[\emptyset, \{a\}, \{a, b\}, \{a, b, d\}]$, $[\emptyset, \{b\}, \{a, b\}, \{a, b, d\}]$, $[\emptyset, \{a, b\}, \{a, b, d\}]$ or $[\emptyset, \{b\}, \{b, c\}]$.

Events a and b are concurrent and so it is possible for both events to occur in a single step. However, “step” concurrency is not demanded and so interleaving of a and b is also possible. Because events a and c are in conflict, once a has occurred, c can never occur and vice versa. The event d is enabled by $\{a, b\}$ so depends on the occurrence of both a and b before it can be exhibited. We observe that conflict is inherited through the enabling relation. Because a is in conflict with c and a enables d , then c and d are also in conflict.

▽

Chapter 3

Introducing Resource Structures

In this chapter, we introduce a different form of event structures called resource structures and look at how one can model the generation, consumption and sharing of resources in this framework.

3.1 Formal Definition

Definition 3.1 *A resource structure is a pair (E, \vdash) where*

E is a set of event occurrences

$\vdash \subseteq \mathcal{P}_f(E) \times \mathcal{P}_f(E)$ represents the enabling relation.

Resource structures are based on event structures with a modified enabling relation, allowing sets of events to be enabled instead of single events at a time. Also, the requirement on the enabling relation that $X \vdash e \ \& \ X \subseteq Y \in \text{Con} \Rightarrow Y \vdash e$ no longer holds. If an event (or set of events) has been enabled, then they do not automatically remain enabled throughout the rest of the computation. Because monotonicity is not assumed, all possible configurations at which an event may occur must be included in the enabling relation. Conflict is represented by the absence of enablings and so the conflict relation is no longer required.

We now define a configuration of a resource structure.

Definition 3.2 *Let $R = (E, \vdash)$ be a resource structure. Define a configuration of R to be a subset of events $C \subseteq E$ where*

\emptyset is a configuration.

if C is a configuration, $C \vdash S$ and $C \cap S = \emptyset$ then $C \cup S$ is also a configuration.

The set of all configurations of a resource structure, R , is written as $\mathcal{C}(R)$.

The first requirement states that \emptyset is always a configuration of a resource structure. This corresponds to the initial state of a computation, when no events have occurred. The second condition requires that any other configuration must have been enabled from a previous configuration, which was enabled by a previous configuration, and so on. This series of enablings must eventually terminate with the initial configuration, ensuring that any configuration is reachable.

As with event structures, a given resource structure induces a transition system. The states, or nodes of the diagram correspond to configurations and the arcs which connect a pair of nodes correspond to transitions between configurations and are labelled with the set of events which are executed concurrently in the transition.

Definition 3.3 A resource structure, $R = (E, \vdash)$ induces a transition system, $TS = (S, \Sigma, \longrightarrow)$ where

$$S = \mathcal{C}(R)$$

$$\Sigma = \mathcal{P}_f(E)$$

$\longrightarrow \subseteq S \times \Sigma \times S$ is the transition relation. $C \xrightarrow{S} C'$ iff

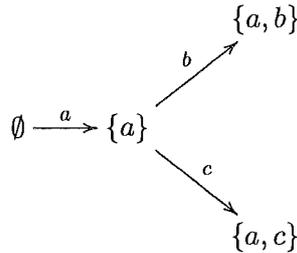
1. $C \vdash S$,
2. $C' = C \cup S$,
3. $C \cap S = \emptyset$ and
4. $S \neq \emptyset$

If a configuration enables a set of events, then it is possible to concurrently exhibit those events, entering a new configuration. Requirement 1 ensures that enabling is respected. The new configuration will be the union of the old configuration and the events just seen, (requirement 2). Only transitions from configurations are considered, and so any unreachable or irrelevant behaviour of resource structures is ignored. As with the transition system induced by event structures, requirements 3 and 4 ensure that each event occurs at most once in any computation and only non-empty transitions are considered.

We shall use transition systems again to represent the relevant behaviour of resource structures.

Notation. As before, we use symbols a, b, c, \dots to represent events, S, S_1, S_2, S', \dots to represent sets of events and C, C_1, C_2, C', \dots to represent configurations. As a notational convenience we write $a \vdash b$ instead of $\{a\} \vdash \{b\}$. We also write $C \xrightarrow{a,b} C'$ instead of $C \xrightarrow{\{a,b\}} C'$.

Example. The resource structure, R , with $E = \{a, b, c, d\}$ and $\emptyset \vdash a, a \vdash b, a \vdash c, c \vdash d$ has configurations $\mathcal{C}(R) = \{\emptyset, \{a\}, \{a, b\}, \{a, c\}\}$. The sets of events $\{c\}$ and $\{c, d\}$ are not reachable and are therefore not configurations of R . The behaviour of R can be represented by the following transition system.

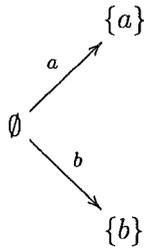


▽

3.2 Examples

Resource consumption. The consumption of a non-reusable resource provides conflict and forces a choice to be made between events. This is shown by the resource structure where

- events $E = \{a, b\}$
- enabling relation $\emptyset \vdash a, \emptyset \vdash b$



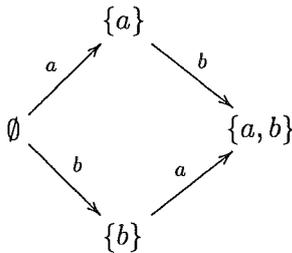
Both events a and b are enabled from the initial configuration, \emptyset , and so either of these events may occur at this point. However, once a occurs, b can no longer occur because the configuration $\{a\}$ does not enable b . In a similar manner, a cannot occur once b occurs. Events a and b are in conflict because the occurrence of one event rules out the occurrence of the other, and a choice must be made between them. We see that conflict is modelled by the absence of enablings.

▽

Resource contention. As seen in the previous example, the enabling relation directly enables events from a given configuration and the events do not remain enabled. Events which are sharing a reusable resource are not in conflict because both events can occur one after the other in any order. To model this, we must consider all possible configurations of the system, and which events can occur from these configurations. These events must be explicitly enabled.

The following resource structure shows how resource contention between events a and b can be modelled using resource structures.

- events $E = \{a, b\}$
- enabling relation $\emptyset \vdash a, \emptyset \vdash b, b \vdash a, a \vdash b$



Both events a and b are enabled from the initial configuration and so either event may initially occur. If an a occurs, b is now enabled and so may occur. Similarly, once a b occurs, a is enabled. Hence the events a and b must be executed in an interleaving manner. They can be considered independent because the order in which they occur is not constrained.

▽

Step concurrency. Step concurrency is the concurrent execution of two or more events in a single step.

The concurrent execution of two events a and b is shown by the resource structure with

- events $E = \{a, b\}$
- enabling relation $\emptyset \vdash \{a, b\}$

$$\emptyset \xrightarrow{a,b} \{a, b\}$$

Both events a and b are enabled from the initial configuration and so these events can be executed in a single step. No other events are enabled from \emptyset and so no other actions are possible. The notion of step concurrency is useful when modelling synchronous systems where multiple components are performed together in a lockstep manner. For example, timed systems require each step of a process to synchronise with the “tick” of a clock.

▽

With resource structures, step concurrency is illustrated by enabling a set of two or more events from a given configuration. If $C \vdash S$ and $|S| > 1$, all events in S must be exhibited in a single step and can be considered truly concurrent. Unlike event structures and distributed transition systems [LPRT95], step concurrency can be demanded and does not imply that interleaving is possible. If $C \vdash S$, this does not automatically imply that $C \vdash S'$ where $S' \subseteq S$.

Possible concurrency is modelled by resource structures enabling the concurrent set of events as well as all subsets of this set.

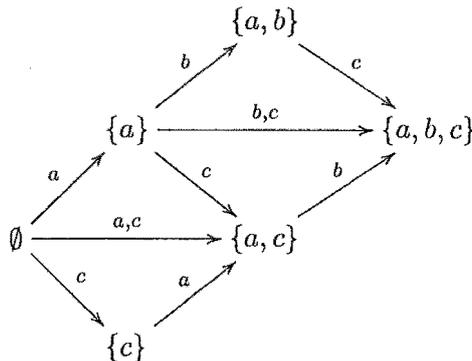
Event structures versus resource structures. In this example we illustrate the difference between event structures and resource structures.

Consider the enabling relation

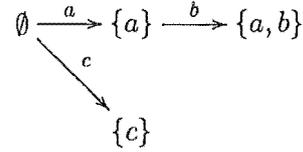
$$a \vdash b, \emptyset \vdash a, \emptyset \vdash c$$

If this is the enabling relation for an event structure with an empty conflict relation, then the only requirement on the corresponding system is that event a must be performed before event b . There are no events in conflict. Events a and c are concurrent from the initial configuration, and events b and c are concurrent after an a .

By monotonicity, rules $a \vdash c$ and $b \vdash c$ are implied by the above set of rules and we obtain the following transition system.



However for a resource structure, $a \vdash c$ is not in the enabling relation so c cannot occur after the a . Events a and c are in conflict because once a occurs, c can never occur and vice versa.



To obtain a resource structure which is the same as the original event structure the enabling relation must be augmented by adding all valid configurations and the events which are possible from these configurations.

$$\emptyset \vdash a, \emptyset \vdash c, \emptyset \vdash \{a, c\}, c \vdash a, a \vdash b, a \vdash c, a \vdash \{b, c\}, \{a, b\} \vdash c, \{a, c\} \vdash b$$

▽

Non-Terminating resource structures Resource structures may be infinite which means that it is possible for them have an infinite run of event occurrences. For example, consider the resource structure with events $E = \{a_i \mid i \in \{0, 1, \dots\}\}$ and enabling relation $\{a_i \mid i < j\} \vdash a_j$.

$$\emptyset \xrightarrow{a_0} \{a_0\} \xrightarrow{a_1} \{a_0, a_1\} \xrightarrow{a_2} \dots$$

This resource structure produces the infinite sequence of events a_0, a_1, a_2, \dots

▽

This concludes the introduction to resource structures. In the next chapter we will look at ways of combining resource structures.

Chapter 4

Combining Resource Structures

Complex structures are often created from the combination of smaller substructures in various ways. This is also possible for resource structures. Resource structures are a model for concurrent systems, which like any computer system, should be designed and implemented in a modular manner. In this section, we will introduce and define the constructors used to combine resource structures, namely sequencing, choice, parallel composition and synchronous composition.

Notation. The empty resource structure (\emptyset, \emptyset) is also denoted by \emptyset . This corresponds to the terminated resource structure.

Two resource structures are said to be *disjoint* when their sets of events are also disjoint. For this section, we assume we are combining disjoint resource structures. If two resource structures $R_1 = (E_1, \vdash_1)$ and $R_2 = (E_2, \vdash_2)$ are not disjoint, that is $E_1 \cap E_2 \neq \emptyset$ then they can be made disjoint by relabelling their elements in a unique way. For example, by letting $E_1' = E_1 \times \{0\}$ and $E_2' = E_2 \times \{1\}$.

Let $(E, \vdash) \times \{i\}$ be the resource structure (E', \vdash') such that

$$E' = E \times \{i\} \text{ and}$$

$$C \vdash S \Rightarrow \{(e, i) \mid e \in C\} \vdash' \{(f, i) \mid f \in S\}$$

4.1 Sequencing

The sequential composition of two resource structures $(R_1 ; R_2)$ is a resource structure which initially behaves like R_1 until R_1 reaches a terminating state and then continues by behaving like R_2 . This operator is useful for modelling a sequence of event occurrences from one process followed by another sequence from another process. Sequencing can be viewed as resource generation, where the first process generates resources that are required by the second process. The second process must wait for the first process to finish before it can proceed.

A terminating state of a resource structure is a final configuration from which no events can occur.

Definition 4.1 Let $R = (E, \vdash)$ be a resource structure. Define a final configuration of R to be a configuration, $C_f \in \mathcal{C}(R)$ where

$$\nexists S \text{ such that } C_f \vdash S$$

The set of all final configurations of a resource structure R is written as $\mathcal{F}(R)$.

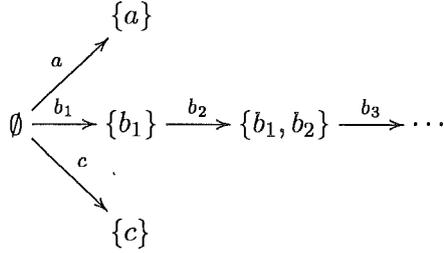
A set of events may only occur at a configuration if it has been directly enabled by that configuration. A final configuration does not enable any set of events, therefore no transition may occur from that configuration.

Example. The resource structure R , with

- events $E = \{a, c, b_i \mid i \in \{0, 1, \dots\}\}$
- enabling relation

$$\emptyset \vdash a, \emptyset \vdash c, \{b_i \mid i < j\} \vdash b_j$$

is an example of a resource structure with an infinite run as shown in the following transition diagram.



$\mathcal{F}(R) = \{\{a\}, \{c\}\}$. The sequence of b_i 's does not terminate and so is not included in the set of final configurations.

▽

Definition 4.2 Let $R_1 = (E_1, \vdash_1)$ and $R_2 = (E_2, \vdash_2)$ be resource structures. Define $(R_1; R_2) = (E, \vdash)$ where

$$E = E_1 \cup E_2$$

$$\vdash = \vdash_1 \cup \vdash'$$
 where

$$C \vdash_2 S, C_f \in \mathcal{F}(R_1) \Rightarrow C \cup C_f \vdash' S$$

The enabling relation of the sequential composition $R_1; R_2$ contains the enabling relation of R_1 . It also contains the enabling relation of R_2 with the extra constraint that events are only enabled if a final configuration of R_1 has occurred. This means that initially $R_1; R_2$ can only exhibit the behaviour of R_1 because all events from R_2 are blocked. Once a final configuration of R_1 has occurred, no more events from R_1 may occur and R_2 is free to proceed.

Example. Let A be the resource structure with events $\{a, b\}$ and enabling relation

$$\emptyset \vdash a, a \vdash b.$$

Let B be the resource structure with events $\{x, y\}$ and enabling relation

$$\emptyset \vdash x, \emptyset \vdash y.$$

The corresponding transition systems are shown in Figure 4.1.

$\mathcal{F}(A) = \{\{a, b\}\}$. The sequential composition $(A; B)$ results in a resource structure with

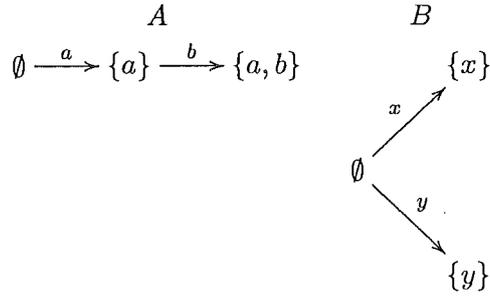
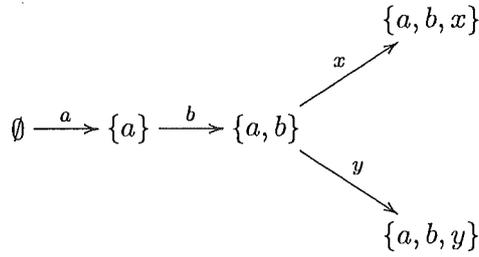


Figure 4.1: Transition systems for resource structures A and B .

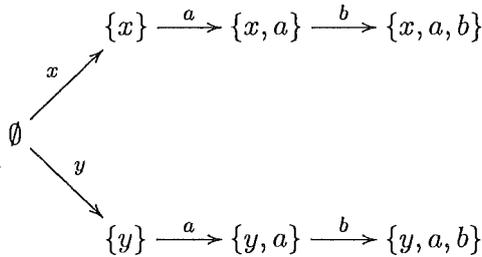
- events $E = \{a, b, x, y\}$
- enabling relation $\emptyset \vdash a, a \vdash b, \{a, b\} \vdash x, \{a, b\} \vdash y$

and can be represented by the following transition system



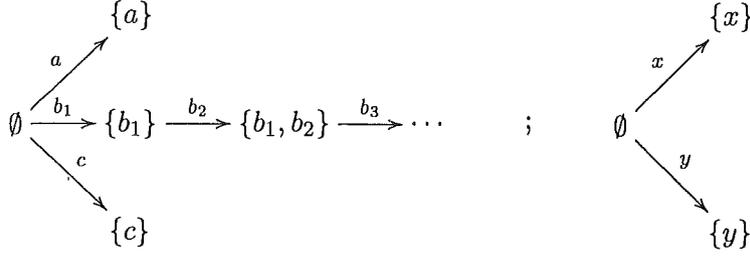
$\mathcal{F}(B) = \{\{x\}, \{y\}\}$. The sequential composition $(B ; A)$ results in the resource structure with

- events $E = \{a, b, x, y\}$
- enabling relation $\emptyset \vdash x, \emptyset \vdash y, x \vdash a, y \vdash a, \{x, a\} \vdash b, \{y, a\} \vdash b$

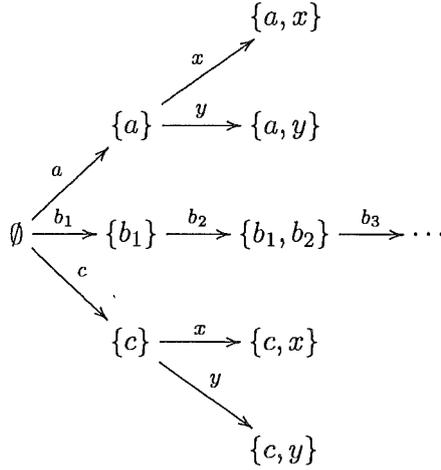


▽

Example. Consider the sequential composition of resource structures R and B defined in the previous two examples:



The sequential composition $R ; B$ corresponds to the resource structure



This composite resource structure behaves initially like R and can exhibit either an a , b_1 or c . R terminates after exhibiting an a or a c and so the final configurations of R are $\mathcal{F}(R) = \{\{a\}, \{c\}\}$. From these configurations, events in B may occur. However, if R initially exhibits b_1 then it will exhibit an infinite sequence of b_i 's and never reach a final configuration. Consequently, events from B are never enabled and cannot occur in this case. ∇

Example.

$$\begin{array}{c} P \\ \emptyset \xrightarrow{a_1} \{a_1\} \xrightarrow{a_2} \dots \end{array} ; \begin{array}{c} Q \\ \emptyset \xrightarrow{b} \{b\} \end{array} = \begin{array}{c} \emptyset \xrightarrow{a_1} \{a_1\} \xrightarrow{a_2} \dots \end{array}$$

The resource structure P consists of an infinite run of a_i events and never terminates. $\mathcal{F}(P) = \emptyset$ and so the event b from Q is never enabled. ∇

4.2 Choice

This operator represents branching in resource structures where each branch is a resource structure. The resultant resource structure behaves like either of its components. The exact behaviour depends on the chosen branch. This operator is useful in modelling the choice between two processes each requiring the use of a common, consumable resource. Hence, the choice operator can be seen as resource consumption.

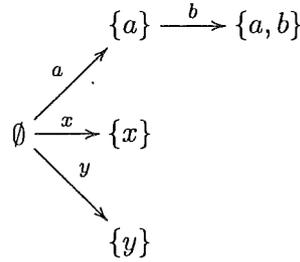
Definition 4.3 Let $R_1 = (E_1, \vdash_1)$ and $R_2 = (E_2, \vdash_2)$ be resource structures. Define $(R_1 \& R_2) = (E, \vdash)$ where

$$E = E_1 \cup E_2$$

$$\vdash = \vdash_1 \cup \vdash_2$$

The enabling relation of $(R_1 \& R_2)$ contains all the members of \vdash_1 and all the members of \vdash_2 . If $\emptyset \vdash_1 S_1$ and $\emptyset \vdash_2 S_2$ then $\emptyset \vdash S_1$ and $\emptyset \vdash S_2$. From the initial configuration, \emptyset , the resource structure $(R_1 \& R_2)$ can exhibit the set of events S_1 or S_2 . If the set of events S_1 are exhibited, the composite resource structure must continue to behave like R_1 . This is because we have assumed that the set of events E_1 and E_2 are disjoint and so once events in E_1 have occurred, they can only enable other events in E_1 . The opposite will occur for the initial exhibition of events S_2 . Hence the choice is resolved from the first move of $(R_1 \& R_2)$.

Example. Using resource structures A and B as defined previously, $A \& B$ gives



with events $\{a, b, x, y\}$.

▽

4.3 Parallel Composition

The parallel composition of two resource structures R_1 and R_2 is written $R_1 \mid R_2$ and behaves like R_1 and R_2 set in parallel and sharing a common resource. The transitions of R_1 and R_2 are interleaved in a nondeterministic manner. This operator is useful for modelling the interleaved execution of two processes.

Definition 4.4 Let $R_1 = (E_1, \vdash_1)$ and $R_2 = (E_2, \vdash_2)$ be resource structures. Define $(R_1 \mid R_2) = (E, \vdash)$ where

$$E = E_1 \cup E_2$$

The enabling relation is defined as

$$C_1 \vdash_1 S_1 \Rightarrow C_1 \cup C_2 \vdash S_1 \text{ where } C_2 \in \mathcal{C}(R_2)$$

$$C_2 \vdash_2 S_2 \Rightarrow C_2 \cup C_1 \vdash S_2 \text{ where } C_1 \in \mathcal{C}(R_1)$$

The two resource structures each progress independently in a nondeterministic manner. Therefore, if an event is enabled in R_1 , it should still be enabled after any number of event occurrences from R_2 and vice versa.

Example. Using the previous definitions of resource structures A and B , where the enabling relation of A is

$$\emptyset \vdash a, a \vdash b$$

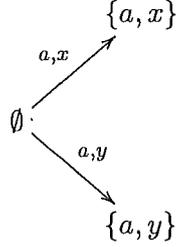
and the enabling relation of B is

$$\emptyset \vdash x, \emptyset \vdash y.$$

$A \otimes B$ is the resource structure over events $\{a, b, x, y\}$ with enabling relation

$$\emptyset \vdash \{a, x\}, \emptyset \vdash \{a, y\}, a \vdash \{b, x\}, a \vdash \{b, y\}.$$

Note that the enabling relation may contain unreachable states. $\{a\}$ is not a valid configuration because it is not reachable and so the last two enableings do not effect the behaviour of the resource structure which is shown in the transition system below.



$A \otimes B$ terminates when B terminates, on the occurrence of either an x or a y . $\mathcal{F}(R) = \{\{a, x\}, \{a, y\}\}$. The b event from resource structure A cannot occur because there is nothing to synchronise with from resource structure B .

▽

Example. The resource structure

$$\emptyset \xrightarrow{a_0} \{a_0\} \xrightarrow{a_1} \{a_0, a_1\} \xrightarrow{a_2} \dots$$

can be thought of as representing a clock, with a sequence of “ticks”, a_0, a_1, a_2, \dots

Any process which is in synchronous composition with such a clock process, will be forced to synchronise each action with a tick of the clock. For example:

$$\emptyset \xrightarrow{b_0} \{b_0\} \xrightarrow{b_1} \{b_0, b_1\} \xrightarrow{b_2} \{b_0, b_1, b_2\} \otimes \emptyset \xrightarrow{a_0} \{a_0\} \xrightarrow{a_1} \{a_0, a_1\} \xrightarrow{a_2} \dots$$

corresponds to the resource structure

$$\emptyset \xrightarrow{a_0, b_0} \{a_0, b_0\} \xrightarrow{a_1, b_1} \{a_0, b_0, a_1, b_1\} \xrightarrow{a_2, b_2} \{a_0, b_0, a_1, b_1, a_2, b_2\}$$

▽

4.5 Composite Resource Structures

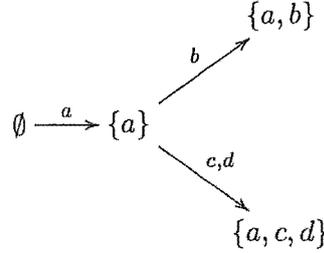
A resource structure may either be specified directly by means of a set of events and enabling relation, or by the combination of various smaller resource structures. Using the combinators previously defined to construct resource structures is not essential, as it is possible to formally represent any resource structure by a set of events and enabling relation. However, for large and complex systems, the resultant enabling relation may not be very intuitive, and it may be easier to define the system as a composition of smaller systems involving sequencing, choice, interleaving or synchronous composition of the subsystems.

Defining resource structures in this way can be thought of as a process language [BW90, Hen88] where the syntax is as follows:

$$RS ::= (E, \vdash) \mid (RS ; RS) \mid (RS \& RS) \mid (RS \mid RS) \mid (RS \otimes RS)$$

Example. Consider the composite resource structure $R_1 ; (R_2 \& (R_3 \otimes R_4))$, where R_1 , R_2 , R_3 and R_4 are also resource structures. The resulting resource structure would behave initially as R_1 , followed by either R_2 , or R_3 and R_4 together in lockstep.

For the simple situation where each component resource structure, R_1 , R_2 , R_3 and R_4 only exhibit a single event, a , b , c or d respectively, the composite resource structure has the behaviour shown in the following transition system.



Note that this resource structure can also be expressed by the enabling relation

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

over the set of events $E = \{a, b, c, d\}$.

This enabling relation for the composite resource structure can also be obtained from the formal definitions of the constructors.

Let $R_3 \otimes R_4$ be the resource structure (E', \vdash') . From Definition 4.5,

$$\emptyset \vdash_3 c, \emptyset \vdash_4 d \Rightarrow \emptyset \vdash' \{c, d\}$$

Let $R_2 \& (R_3 \otimes R_4)$ be the resource structure (E'', \vdash'') . From Definition 4.3,

$$\emptyset \vdash_2 b, \emptyset \vdash' \{c, d\} \Rightarrow \emptyset \vdash'' b, \emptyset \vdash'' \{c, d\}$$

Let $R_1 ; (R_2 \& (R_3 \otimes R_4))$ be the resource structure (E, \vdash) . From Definition 4.2,

$$\emptyset \vdash_1 a \Rightarrow \emptyset \vdash a$$

$$\mathcal{F}(R_1) = \{\{a\}\}, \emptyset \vdash'' \{c, d\}, \emptyset \vdash'' b \Rightarrow a \vdash b, a \vdash \{c, d\}$$

▽

Chapter 5

Properties

We can specify processes algebraically using the resource structure constructors as operators. In this chapter, we present properties satisfied by constructions on resource structures. This is based on a notion of equivalence which is defined below. We also show that resource structures can model event structures.

We would like to be able to determine if two processes specified in different ways are equal. For this, we need a notion of equivalence on resource structures.

We say two resource structures are equivalent if they are *trace equivalent*[Hoa85]. That is, if the two resource structures can exhibit the same observations, where an observation consists of one or more events occurring in one step.

Definition 5.1 Let $R_1 = (E_1, \vdash_1)$ and $R_2 = (E_2, \vdash_2)$ be resource structures. Define $R_1 \approx R_2$ iff

$$C \xrightarrow{S}_1 C' \Leftrightarrow C \xrightarrow{S}_2 C'$$

where \xrightarrow{S}_1 is the transition relation induced by R_1 and \xrightarrow{S}_2 the transition relation induced by R_2 .

The definition states that equivalent resource structures must have the same transition relation, and hence the same transition system. The transition system consists of only the relevant behaviour of a system because no unreachable states or transitions are considered. Therefore, two equivalent resource structures will have the same behaviour.

Event structures with identical sets of configurations will induce the same transition relation and so it is sufficient to compare configurations. However, two resource structures may have the same configurations but different transition relations. An example follows.



For this reason it is necessary to compare transitions of resource structures as well as configurations.

Example. Let $R_1 = (E_1, \vdash_1)$ and $R_2 = (E_2, \vdash_2)$ be resource structures where

$$\begin{array}{ll} E_1 = \{a, b\} & E_2 = \{a, b, c, d\} \\ \emptyset \vdash_1 a, a \vdash_1 b & \emptyset \vdash_2 a, a \vdash_2 b, c \vdash_2 d \end{array}$$

$\mathcal{C}(R_1) = \{\emptyset, \{a\}, \{a, b\}\} = \mathcal{C}(R_2)$ with transition relation $\emptyset \xrightarrow{a} \{a\} \xrightarrow{b} \{a, b\}$ for both R_1 and R_2 . Because both resource structures have the same transition relation, $R_1 \approx R_2$. The events c and d do not have any influence on the behaviour of the resource structure R_2 as no configuration containing a c is reachable, and so the events c and d can never occur. ∇

Proposition 5.1 \approx is an equivalence relation. That is, it is reflexive, symmetric and transitive.

We also define equivalence on resource structures with respect to a relabelling function, allowing us to have structural equivalence over different events. This is useful for comparing two processes which perform the same computations, but use a different naming system for the events.

Definition 5.2 Let $R_1 = (E_1, \vdash_1)$ and $R_2 = (E_2, \vdash_2)$ be resource structures and let f be a function mapping events in R_1 to events in R_2 or vice versa, where $f(S)$ is a pointwise application of f to the set of events S . Define $R_1 \approx_f R_2$ iff

$f : E_1 \rightarrow E_2$ such that

$$\begin{array}{ll} C_1 \xrightarrow{S_1}_1 C'_1 & \Rightarrow f(C_1) \xrightarrow{f(S_1)}_2 f(C'_1) \\ C_2 \xrightarrow{S_2}_2 C'_2 & \Rightarrow \exists C \xrightarrow{S}_1 C' \text{ such that } f(C) = C_2 \text{ and } f(S) = S_2. \end{array}$$

or $f : E_2 \rightarrow E_1$ such that

$$\begin{array}{ll} C_2 \xrightarrow{S_2}_1 C'_2 & \Rightarrow f(C_2) \xrightarrow{f(S_2)}_1 f(C'_2) \\ C_1 \xrightarrow{S_1}_2 C'_1 & \Rightarrow \exists C \xrightarrow{S}_2 C' \text{ such that } f(C) = C_1 \text{ and } f(S) = S_1. \end{array}$$

where \rightarrow_1 is the transition relation induced by R_1 and \rightarrow_2 is the transition relation induced by R_2 .

If it is possible to rename events in R_1 to correspond to events in R_2 , giving equivalent transition relations, then we say that R_1 and R_2 are equivalent over the renaming function, f .

Note that this equivalence over a relabelling function is not a strict equivalence as it is neither reflexive nor transitive. However, a pseudo-transitivity does hold.

Proposition 5.2 If $R_1 \approx_f R_2$ and $R_2 \approx_g R_3$ then $\exists h$ such that $R_1 \approx_h R_3$

Proof Consider the case where $f : E_1 \rightarrow E_2$ and $g : E_3 \rightarrow E_2$.

$h(e_1) = e_3$ such that $g(e_3) = f(e_1)$, where $e_1 \in E_1, e_3 \in E_3$. \square

Without loss of generality, we assume that $R_1 \approx_f R_2 \Rightarrow f : E_1 \rightarrow E_2$. Both directions are included in the definition for symmetry.

5.1 Properties of Resource Structures

We now use this notion of equivalence on resource structures to present several properties which are satisfied by the constructors of resource structures.

To show that two resource structures, R_1 and R_2 , are equivalent, we show

1. If C is a configuration of R_1 then C is a configuration of R_2 , and
2. If $C \vdash_{R_1} S$ in R_1 then $C \vdash_{R_2} S$ in R_2 .

If 1 and 2 can be shown then it follows that

$$C \xrightarrow{S}_{R_1} C' \text{ in } R_1 \Rightarrow C \xrightarrow{S}_{R_2} C' \text{ in } R_2.$$

The operators $\&$, $|$ and \otimes are commutative, however the order of components are important for sequential composition.

Proposition 5.3 Commutative laws

1. $R_1 \& R_2 \approx R_2 \& R_1$
2. $R_1 | R_2 \approx R_2 | R_1$
3. $R_1 \otimes R_2 \approx R_2 \otimes R_1$

Proof Directly from the definitions. □

Lemma 5.1 C is a configuration of $R_1 ; R_2 \Leftrightarrow$

C is a configuration of R_1 or

$\exists C_1, C_2$ such that

$$C_1 \cup C_2 = C,$$

C_1 is a final configuration of R_1 and

C_2 is a configuration of R_2 .

Proof Let $R_1 = (E_1, \vdash_1)$, $R_2 = (E_2, \vdash_2)$ and $(R_1 ; R_2) = (E, \vdash)$.

\Rightarrow

Let C be a configuration of $(R_1 ; R_2)$.

$C \subseteq E_1 \cup E_2$ and $\exists S_1, S_2, \dots, S_n$ such that

$$\emptyset \vdash S_1,$$

$$\forall j \leq n, \bigcup_{k < j} S_k \vdash S_j \text{ and}$$

$$\bigcup_{i \leq n} S_i = C.$$

$$\exists C_1, C_2 \text{ such that } C_1 \cup C_2 = C, C_1 \subseteq E_1, C_2 \subseteq E_2.$$

- If $C_2 = \emptyset$ then using \vdash_1 instead of \vdash ,
 $\emptyset \vdash_1 S_1, S_1 \vdash_1 S_2, S_1 \cup S_2 \vdash_1 S_3, \dots, \bigcup_{i < n} S_i \vdash_1 S_n$
 $\therefore C$ is a configuration of R_1 .
 - If $C_2 \neq \emptyset$ then $\exists S_i$ such that $\bigcup_{j < i} S_j \vdash S_i$ and $S_i \cap E_2 \neq \emptyset$.
Choose the least such i and call it i' . I.e., the first time an event in E_2 occurs.
 $\bigcup_{j < i'} S_j \cap E_2 = \emptyset$ and $S_{i'} \cap E_2 \neq \emptyset$.
From Definition 4.2, $\bigcup_{j < i'} S_j \vdash S_{i'} \Rightarrow$
 $\bigcup_{j < i'} S_j \not\vdash_1 S_{i'}$ because $\vdash_1 \subseteq \mathcal{P}_f(E_1) \times \mathcal{P}_f(E_1)$.
 $\exists X, Y$ such that
 $X \cup Y = \bigcup_{j \leq i'} S_j$,
 X is a final configuration of R_1 , and
 $Y \vdash_2 S_{i'}$.
 $\bigcup_{j < i'} S_j \subseteq E_1$ therefore $X = \bigcup_{j < i'} S_j \in \mathcal{F}(R_1)$
 $\emptyset \vdash_2 S_{i'}$
- Let $\bigcup_{j < i'} S_j = C_f$. $\forall k \geq i', \bigcup_{j \leq k} S_j \supseteq C_f$.
Applying the previous step for all such $S_k \Rightarrow \bigcup_{j < k} S_j \vdash_2 S_k$. Therefore,
 $C_1 = C_f$, a final configuration of R_1 and
 C_2 is a configuration of R_1 .

\Leftarrow

Follows from the definition. □

Lemma 5.2 C is a configuration of R_1 & $R_2 \Leftrightarrow$

C is a configuration of R_1 or

C is a configuration of R_2 .

Proof Let $R_1 = (E_1, \vdash_1)$, $R_2 = (E_2, \vdash_2)$ and $(R_1 \& R_2) = (E, \vdash)$.

\Rightarrow

Let C be a configuration of $(R_1 \& R_2)$.

$C \subseteq E_1 \cup E_2$ and $\exists S_1, S_2, \dots, S_n$ such that

$$\emptyset \vdash S_1,$$

$$\forall j \leq n, \bigcup_{k < j} S_k \vdash S_j \text{ and}$$

$$\bigcup_{i \leq n} S_i = C$$

From Definition 4.3, $\vdash = \vdash_1 \cup \vdash_2$.

- If $C \subseteq E_1$ and $C \cap E_2 = \emptyset$,
using \vdash_1 for $\vdash \Rightarrow C$ is a configuration of R_1 .

- Similarly for $C \subseteq E_2$ and $C \cap E_1 = \emptyset$.
- If $C \cap E_1 \neq \emptyset$ and $C \cap E_2 \neq \emptyset$ then
 - $\exists S_i$ such that $\bigcup_{j < i} S_j \vdash S_i$ and
 - $(\bigcup_{j < i} S_j \cap E_1 \neq \emptyset \text{ and } S_i \cap E_2 \neq \emptyset)$ or
 - $(\bigcup_{j < i} S_j \cap E_2 \neq \emptyset \text{ and } S_i \cap E_1 \neq \emptyset)$

But $\bigcup_{j < i} S_j \vdash S_i \notin \vdash_1$ because $\vdash_1 \subseteq \mathcal{P}_f(E_1) \times \mathcal{P}_f(E_1)$ and E_1 and E_2 are disjoint. Therefore C is not a configuration of R_1 .

Similarly, C is not a configuration of R_2 .

\Leftarrow

Obvious as $\vdash = \vdash_1 \cup \vdash_2$.

□

Lemma 5.3 C is a configuration of $R_1 \mid R_2 \Leftrightarrow \exists C_1, C_2$ such that

$$C_1 \cup C_2 = C,$$

C_1 is a configuration of R_1 and

C_2 is a configuration of R_2 .

Proof Let $R_1 = (E_1, \vdash_1)$, $R_2 = (E_2, \vdash_2)$ and $(R_1 \mid R_2) = (E, \vdash)$.

\Rightarrow

Let C be a configuration of $(R_1 \mid R_2)$.

$C \subseteq E_1 \cup E_2$ and $\exists S_1, S_2, \dots, S_n$ such that

$$\emptyset \vdash S_1,$$

$$\forall j \leq n, \bigcup_{k < j} S_k \vdash S_j \text{ and}$$

$$\bigcup_{i \leq n} S_i = C$$

$$\exists C_1, C_2 \text{ such that } C_1 \cup C_2 = C, C_1 \subseteq E_1, C_2 \subseteq E_2.$$

- If $C_2 = \emptyset$ then obviously C_2 is a configuration of R_2 ,

$\forall S_i$ such that $\bigcup_{j < i} S_j \vdash S_i$, the following conditions hold:

$$\bigcup_{j < i} S_j \subseteq E_1 \text{ and}$$

$$S_i \subseteq E_1.$$

$\therefore \bigcup_{j < i} S_j \vdash_1 S_i$, by Definition 4.4.

C is a configuration of R_1 and $C = C_1$,

$\therefore C_1$ is a configuration of R_1 .

- Similarly, if $C_1 = \emptyset$ then C_2 is a configuration of R_2 .

- If $C_2 \neq \emptyset$ and $C_1 \neq \emptyset$ then

$\exists S_i$ such that the following conditions hold:

$$\bigcup_{j < i} S_j \vdash S_i,$$

$$\bigcup_{j < i} S_j \cap E_2 = \emptyset \text{ and}$$

$$S_i \cap E_2 \neq \emptyset$$

where S_i is the first occurrence of events in E_2 .

From Definition 4.4, $\bigcup_{j < i} S_j \vdash S_i \Rightarrow$

$\bigcup_{j < i} S_j$ is a configuration of R_1 and

$$\emptyset \vdash_2 S_i$$

Similarly for $S_{i'}$, the first occurrence of events in E_1 .

By iterating this step over all S_i , C_1 is a configuration of R_1 and C_2 is a configuration of R_2 .

□

The components of the parallel composition of resource structures can behave independently, without demanding fairness.

Proposition 5.4 *If C is a configuration of R_1 , then C is also a configuration of $(R_1 \mid R_2)$*

Proof This follows from Lemma 5.3. Let C be a configuration of R_1 . \emptyset is a configuration of R_2 . Therefore C is a configuration of $R_1 \mid R_2$.

□

This means that it is possible for the components to occur in sequence.

Inconsistency between events is preserved in the parallel composition of resource structures.

Proposition 5.5 *If $\exists S \subseteq E_1$ such that $\forall C \in \mathcal{C}(R_1), S \not\subseteq C$ then $\forall C' \in \mathcal{C}(R_1 \mid R_2), S \not\subseteq C'$.*

If it is not possible for a set of events, S to occur in any computation of a component, then it is not possible for that set of events to occur in any computation of the composite resource structure. This means that conflict, and therefore choice between events is preserved.

Lemma 5.4 *C is a configuration of $R_1 \otimes R_2 \Rightarrow \exists C_1, C_2$ such that*

$$C_1 \cup C_2 = C,$$

C_1 is a configuration of R_1 and

C_2 is a configuration of R_2 .

Proof The proof is similar to the previous proofs presented, using Definition 4.5. □

The other direction of the lemma will only hold if the two configurations C_1 and C_2 can be reached after the same number of steps from the initial configuration. All other configurations will result in unreachable states of the synchronous composition.

The empty resource structure acts as a unit for $;$, $\&$ and $|$, and as an annihilator for \otimes .

Lemma 5.5 Unit and annihilator laws

1. $\emptyset ; R \approx R ; \emptyset \approx R$
2. $\emptyset \& R \approx R$
3. $\emptyset | R \approx R$
4. $\emptyset \otimes R \approx \emptyset$

Proof

1. We prove (a) $\emptyset ; R \approx R$ and (b) $R ; \emptyset \approx R$. By transitivity, $\emptyset ; R \approx R ; \emptyset \approx R$.

- (a) Let the enabling relation of $(\emptyset ; R)$ be \vdash and the enabling relation of R be \vdash_R .

\Rightarrow

Let C be a configuration of $\emptyset ; R$.

If $C = \emptyset$ then C is also a configuration of R .

If $C \neq \emptyset$ then by Lemma 5.1,

$\exists C_1, C_2$ such that

$$C_1 \cup C_2 = C,$$

C_1 is a final configuration of \emptyset and

C_2 is a configuration of R .

Clearly $C_1 = \emptyset$, as it is the only configuration of \emptyset .

$$\therefore C_2 = C$$

C is a configuration of R .

Assume $C \vdash S$. By Definition 4.2, $\exists C_1, C_2$ as before where $C_1 = \emptyset$ and $C_2 \vdash_R S$.

$$C_1 \cup C_2 = C$$

$$\therefore C_2 = C$$

$$\therefore C_2 \vdash_R C$$

$$C \vdash S \Rightarrow C \vdash_R S'$$

$$\therefore C \xrightarrow{S} C' \Rightarrow C \xrightarrow{S}_R C'$$

\Leftarrow

Let C be a configuration of R . \emptyset is a final configuration of R therefore, by Lemma 5.1, C is a configuration of $\emptyset ; R$.

From Definition 4.2, $C \vdash_R S \Rightarrow C \vdash S$ (as $\emptyset \in \mathcal{F}(\emptyset)$).

- (b) Similarly, let the enabling relations of $(R ; \emptyset)$ and R be \vdash and \vdash_R respectively.

\Rightarrow

Let C be a configuration of $R; \emptyset$.

From Lemma 5.1,

- C is a configuration of R , or
 - $\exists C_1, C_2$ such that
 - $C_1 \cup C_2 = C$,
 - C_1 is a final configuration of R and
 - C_2 is a configuration of \emptyset .
- Clearly $C_2 = \emptyset$, as it is the only configuration of \emptyset .
 $\therefore C_1 = C$
 C is a configuration of R .

From Definition 4.2, $C \vdash S \Rightarrow C \vdash_R S$ (as $\vdash_\emptyset = \emptyset$).

\Leftarrow

Similar to previous proof.

2. This is similar to the previous argument, using Lemma 5.2
3. Also similar to the argument in 1, using Lemma 5.3
4. Let the enabling relation of $(\emptyset \otimes R)$ be \vdash , the enabling relation of R be \vdash_R and the enabling relation of \emptyset be \vdash_\emptyset .

From Definition 4.5,

$$\emptyset \vdash S_1 \Rightarrow \exists X, Y \text{ such that } \emptyset \vdash_R X \text{ and } \emptyset \vdash_\emptyset Y$$

$$\text{But } \vdash_\emptyset = \emptyset, \text{ therefore } \vdash = \emptyset$$

$$\therefore \longrightarrow = \longrightarrow_\emptyset = \emptyset$$

□

Lemma 5.6 Distributive laws

1. $(R_1 \times \{0\}; R_2) \& (R_1 \times \{1\}; R_3) \approx_f R_1; (R_2 \& R_3)$
2. $(R_1 \times \{0\} \mid R_2) \& (R_1 \times \{1\} \mid R_3) \approx_f R_1 \mid (R_2 \& R_3)$
3. $(R_1 \times \{0\} \otimes R_2) \& (R_1 \times \{1\} \otimes R_3) \approx_f R_1 \otimes (R_2 \& R_3)$

where

$f : E_1 \times \{0\} \cup E_1 \times \{1\} \cup E_2 \cup E_3 \longrightarrow E_1 \cup E_2 \cup E_3$ such that

$$\forall e \in E_1, f(e, i) = e \text{ where } i \in \{0, 1\}$$

$$\forall e' \in E_2 \cup E_3, f(e') = e'$$

Proof

1. From Lemma 5.2, C is a configuration in $(R_1 \times \{0\}; R_2) \& (R_1 \times \{1\}; R_3) \Leftrightarrow$

- C is a configuration of $(R_1 \times \{0\}; R_2)$
- or C is a configuration of $(R_1 \times \{1\}; R_3)$.

From Lemma 5.1, C is a configuration of $(R_1 \times \{0\}; R_2) \Leftrightarrow$

- C is a configuration of $(R_1 \times \{0\})$
 C under f is a configuration of R_1
 $\therefore f(C)$ is a configuration of $R_1; (R_2 \& R_3)$ (from Lemma 5.1).
- or $\exists C_1, C_2$ such that
 $C_1 \cup C_2 = C$,
 C_1 is a final configuration of $R_1 \times \{0\}$ and
 C_2 is a configuration of R_2 .
 $f(C_1)$ is a configuration of R_1 and C_2 is a configuration of $(R_2 \& R_3)$
 $\therefore f(C_1) \cup C_2$ is a configuration of $R_1; (R_2 \& R_3)$ (from Lemma 5.1).
 $C_2 = f(C_2), \therefore f(C_1) \cup C_2 = f(C_1 \cup C_2)$
 $\therefore f(C)$ is a configuration of $R_1; (R_2 \& R_3)$

Following the same pattern as the proofs for the lemmas, it can be shown that $C \vdash_1 S \Leftrightarrow f(C) \vdash_2 f(S)$ and therefore $C \xrightarrow{S}_1 C' \Leftrightarrow f(C) \xrightarrow{f(X)}_2 f(C')$ where \vdash_1 and \rightarrow_1 are the enabling and transition relations of $(R_1 \times \{0\}; R_2) \& (R_1 \times \{1\}; R_3)$ and \vdash_2 and \rightarrow_2 are the enabling and transition relations of $R_1; (R_2 \& R_3)$.

2. This proof is similar to the previous proof, using Lemma 5.3 instead of Lemma 5.1.

3. Follows a similar argument to 1.

□

This concludes the section on properties for the sequencing, choice, parallel composition and synchronous composition operators.

5.2 Comparing Resource Structures and Event Structures

We now show that for every event structure, there is an equivalent resource structure.

Proposition 5.6 *Given an event structure, $E = (E_E, \#, \vdash_E)$ there exists a resource structure, $R = (E_R, \vdash_R)$ such that $E \approx R$ where*

$$E_R = E_E$$

$$C \xrightarrow{S}_E C' \Rightarrow C \vdash_R S$$

where \rightarrow_E is the transition relation induced by E , (defined in Definition 2.4).

Proof We write $C \rightarrow C'$ instead of $C \xrightarrow{C'-C} C'$ for convenience.

If $C \rightarrow_E C'$, then C is a configuration of E .

$\therefore \exists C_1, C_2, \dots$ such that $\emptyset \rightarrow_E C_1 \rightarrow_E C_2 \rightarrow_E \dots \rightarrow_E C$

$$\emptyset \rightarrow_E C_1 \Rightarrow \emptyset \vdash_R C_1$$

$$C_1 \rightarrow_E C_2 \Rightarrow C_1 \vdash_R (C_2 - C_1)$$

...

Therefore C is a configuration of R .

$$C \vdash_R S \Rightarrow C \xrightarrow{S} C'$$

□

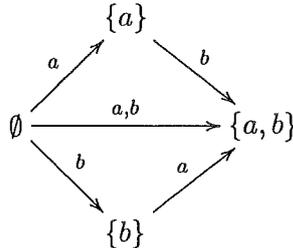
For resource structures, events have to be explicitly enabled at each point at which they may occur. Also, concurrent events must be enabled as a set of events.

If we allow homomorphisms on events then we can obtain an equivalent event structure for a given resource structure. A homomorphism is required because step concurrency cannot be demanded by event structures. The concurrent exhibition of a set of non-interleaving events must be represented by a single event in an event structure.

Example. Consider the following resource structure,

$$\emptyset \xrightarrow{a,b} \{a, b\}$$

The concurrent exhibition of events a and b is represented by the event structure where $\emptyset \vdash a$, $\emptyset \vdash b$ and $\# = \emptyset$. However, this is not equivalent to the resource structure above, as a and b can also be interleaved as shown in the following transition diagram.



It is possible to obtain an equivalent event structure if we relabel the set of events $\{a, b\}$ with a single event e and have $\emptyset \vdash e$ and $\# = \emptyset$. This relabelling is necessary because event structures cannot force the events a and b to occur in one step, which is what the resource structure is modelling.

▽

Therefore, event structures and resource structures are equivalent in expressive power.

Chapter 6

Conclusions and Further Work

In this report, we have introduced resource structures as a modification of event structures.

Other work on extensions to event structures includes that of Katoen[Kat96] who introduces various extensions to bundle event structures[Lan92] to model time constraints, timeouts and probabilistic occurrences of events.

We have shown that by modifying the enabling relation of event structures so that sets of events are enabled and also removing the monotonicity requirement that once enabled an event is to remain enabled, the conflict relation is no longer required. Resource structures are able to model conflict through the absence of enablings. This lack of monotonicity means that concepts such as timeouts and interrupts can be expressed very naturally by resource structures.

We are interested in concurrent systems which are resource constrained and have examined the possibility of modelling both concurrency and the management of resources in the same framework.

Resource structures are able to distinguish between interleaving of events (resource contention) and the concurrent execution of one or more events in a single step, which we call “step concurrency”. For event structures, these two concepts come hand in hand. “Possible concurrency” is modelled in which if events can occur concurrently, then they can also be non-deterministically interleaved and vice versa. Step concurrency or resource contention can only be directly modelled if the events are relabelled.

We have introduced several constructors which are useful for specifying complex resource structures in terms of several simpler components. Definitions for sequencing, choice, parallel composition and synchronous composition of resource structures have been given.

These constructors can be considered as the operators of a process language for resource structures which satisfy several interesting properties. For example, the sequencing, parallel composition and synchronous composition operators all distribute over the choice operator. These properties, along with their proofs have also been presented in this report.

We have shown that resource structures and event structures are equivalent in expressive power, however, resource structures provide a more natural way to express properties such as interrupts, timeouts and step concurrency. Further work could be done in the application of resource structures to real systems such as timed systems and electric circuits. Event structures can be used in the specification and verification of circuits and we believe that resource structures may provide a useful representation of synchronous transitions in such circuits.

Future work could also include the development of a logic for resource structures which would enable verification that a model satisfies the required specifications, expressed in terms of the

logic. Mukund and Thiagarajan [MT92] provide a logical characterisation of well branching event structures using temporal logic whereas Girard's linear logic[Gir87], which is a resource conscious logic, can be modelled by Petri nets[EW94]. An adaption to one of these approaches may provide a logic which characterises the behaviour of resource structures.

Bibliography

- [BW90] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, 1990.
- [Che88] L. A. Cherkasova. On models and algebras for concurrent processes. In *Mathematical Foundations of Computer Science*, LNCS 324, pages 27–43, 1988.
- [EW94] U.H. Engberg and G. Winskel. Linear logic on petri nets. In J. W. de Bakker et al., editor, *Proceedings of REX '93*, LNCS 803, 1994.
- [Gir87] J.-Y. Girard. Linear logic. *Theoretic Computer Science*, 50(1):1–102, 1987.
- [Hen88] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.
- [Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
- [Kat96] J.-P. Katoen. *Quantitative and Qualitative Extensions of Event Structures*. PhD thesis, University of Twente, The Netherlands, 1996.
- [Lan92] R. Langerak. Bundle Event Structures: A Non-interleaving Semantics for LOTOS. In M. Diaz and R. Groz, editors, *Formal Description Techniques, V*. North Holland, 1992.
- [LPRT95] K. Lodaya, R. Parikh, R. Ramanujam, and P. S. Thiagarajan. A logical study of distributed transition systems. *Information and Computation*, 119(1):91–118, 1995.
- [MT92] M. Mukund and P. S. Thiagarajan. A logical characterization of well branching event structures. *Theoretical Computer Science*, 96:35–72, 1992.
- [Rei85] W. Reisig. *Petri Nets—an Introduction*. Springer-Verlag, 1985.
- [Win80] G. Winskel. *Events in Computation*. PhD thesis, University of Edinburgh, 1980.
- [Win86] G. Winskel. Event structures. In *Advances in Petri Nets, Part 2*, LNCS 255, pages 325–392. Springer Verlag, 1986.
- [Win88] G. Winskel. An introduction to event structures. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS 354, pages 364–397. Springer-Verlag, 1988.
- [WN94] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky and D. Gabbay, editors, *Handbook of Logic in Computer Science*, volume 3. Oxford University Press, 1994.