Chapter 7

Asynchronous Automata

Wiesław Zielonka
Université Bordeaux I, LaBRI, URA CNRS 1304
351, cours de la Libération, 33405 Talence Cedex, France
zielonka@geocub.greco-prog.fr

Contents

7.1 Introduction

The problem of inventing a suitable machine-like model for traces was implicitly present since the advent of trace theory. Such devices should exhibit two properties

- they should have an adequate computational power, i.e. they should accept exactly recognizable[2] sets of traces and
- the independency of actions should be reflected by the "true" concurrency of their executions and not just by the interleaving.

[2] "Recognizable" means here recognizable in the abstract way — by means of morphisms from the free partially commutative monoids into finite monoids

The condition-event nets used in Mazurkiewicz's seminal paper [189], which introduced traces in the domain of concurrency, satisfy the second condition, but their computational power is clearly insufficient — as it is well-known there are recognizable subsets of the free monoid that cannot be recognized by unlabelled Petri nets. The direct possibility of enhancing the power of condition-event nets passes by endowing them with a labelling of transitions. Unfortunately some labellings give rise to behaviours that are expressed by labelled acyclic graphs that are not interpretable in the framework of trace theory. Thus actually in this case we should specify a class of admissible labellings but such objects become too complex and difficult to handle.

Asynchronous automata introduced in this chapter overcome these difficulties, they have the desired computational power, allow the "real" concurrency of actions and have a nice regular structure. Moreover they place trace theory in the framework of the well-established theory of finite automata.

The chapter is organized as follows.

In Section 7.2 asynchronous transition systems are introduced.

Two natural non-interleaving semantics for these systems are proposed in Section 7.3.

The first of these semantics has been patterned on the process semantics of Petri nets. However we can note that while processes in Petri nets are always represented by directed acyclic graphs, to represent faithfully the behaviour of finite asynchronous transition systems we are obliged to admit in general some cycles (still if an asynchronous transition system can be directly represented as a Petri net then both process semantics are identical, in particular all processes are acyclic.)

The second semantics examined in this section is directly related to traces, actually it yields a trace representation adapted for asynchronous transition systems.

In Section 7.4 a special class of asynchronous transition systems, finite asynchronous cellular transition systems, is presented. In the next Section 7.5 we show that not only asynchronous cellular transition systems have particularly simple and appealing internal structure but in fact they can simulate without any loss of potential concurrency between actions a much larger class of asynchronous transition systems.

In Section 7.6 asynchronous (cellular) transition systems are equipped with the sets of initial and final states and the recognizability power of the automata obtained in this way is examined.

7.2 Asynchronous Transition Systems

The following notation will be used throughout the chapter. By $id_X = \{(x,x) \mid x \in X\}$ we shall denote the identity relation over a set X, if the domain of this relation is clear from the context then the subscript X will often be skipped. Let E be a binary relation. For any element x and a set B: $xE = \{y \mid (x,y) \in E\}$ is the image of x under E, $Ex = \{y \mid (y,x) \in E\}$ is the inverse image of x under E, $BE = \bigcup_{x\in B} xE$, $EB = \bigcup_{x\in B} Ex$ are respectively the image and the inverse image of B under E, finally $E^{-1} = \{(x,y) \mid (y,x) \in E\}$ is the inverse of E. The composition of two binary relations E and F is defined as $E \circ F = \{(x,y) \mid \exists z, (x,z) \in E \text{ and } (z,y) \in F\}$.

The transitive closure of a relation $E \subseteq X \times X$ is defined as $E^+ = \bigcup_{i=0}^{\infty} E^i$, where $E^1 = E$ and $E^{i+1} = E^i \circ E$, while $E^* = id_X \cup E^+$ denotes the transitive and reflexive closure of E.

The relation E is said to be acyclic if its graph is acyclic, i.e. if $id_X \cap E^+ = \emptyset$.

For any two sets X and Y, $\mathbf{F}(X;Y)$ denotes the family of all mappings from X into Y and $\mathcal{P}(X)$ stands for the family of all subsets of X.

For any word $x \in \Sigma^*$ by $|x|$ we denote the length of x, while $|x|_a$ is the number of occurrences of $a \in \Sigma$ in x. The empty word is denoted by $\mathbf{1}$, Rec_Σ stands for the family of recognizable subsets of Σ^*.

After these preliminaries we can pass to the main subject of this chapter. A *signature* is a triple $\sigma = (\Sigma, R, E)$, where

— Σ is the set actions,

— R is the set of registers,

— $E \subseteq \Sigma \times R \cup R \times \Sigma$ is the connection relation.

We assume that R and Σ are disjoint.

Intuitively, a signature represents a static structure of a distributed system, for each action $a \in \Sigma$, $Ea = \{r \in R \mid (r,a) \in E\}$ is the set of registers that a reads when it is executed and $aE = \{r \in R \mid (a,r) \in E\}$ is the set of registers that a modifies (i.e. where it writes a new value).

A *finite asynchronous transition system* (or a fat-system for short) is a tuple $\tau = (\Sigma, R, E, X, \Delta)$, where

— (Σ, R, E) is a signature of τ,

— X is a finite set of values (local states),

— Δ is a family of local transition relations.

The set $\Delta = \{\delta_a \mid a \in \Sigma\}$ consists of local transition relations, for each action $a \in \Sigma$, Δ contains a transition relation δ_a

$$\delta_a \subseteq \mathbf{F}(Ea; X) \times \mathbf{F}(aE; X)$$

To describe precisely how fat-systems work and how local transition relations are used by the actions we need some additional notation.

We assume that at each moment every register contains some value from the set X of values, thus the global state of the system is described by a mapping $s : R \longrightarrow X$ assigning to each register $r \in R$ its values $s(r) \in X$. By S we shall denote the set $\mathbf{F}(R;X)$ of all global states of τ.

For every $s \in S$ and $\alpha \subseteq R$ by $s_{|\alpha}$ we denote the restriction of s to α. Note that $s_{|\alpha} \in \mathbf{F}(\alpha; X)$. In general the mappings from a subset α of R into X will be called partial states.

Elements of the local transition relation δ_a are called *local transitions*. A local transition $(s_1, s_2) \in \delta_a$ is said to be *enabled* at a global state $s' \in S$ if $s_1 = s'_{|Ea}$. The action a is enabled at $s' \in S$ if some of its local transition is enabled. If transition $(s_1, s_2) \in \delta_a$ is enabled at $s' \in S$ then it can be executed by a yielding a new global state $s'' \in S$ such that

$$\forall r \in R, \quad s''(r) = \begin{cases} s'(r) & \text{if } r \notin aE \\ s_2(r) & \text{otherwise} \end{cases}$$

Intuitively, the execution of an action a can be interpreted in the following way. First a reads the values of all registers from its reading domain Ea obtaining a partial state $s' \in \mathbf{F}(Ea; X)$. Next it chooses a partial state $s'' \in \mathbf{F}(aE; X)$ such that $(s', s'') \in \delta_a$ and modifies the values of registers of its writing domain aE writing to each register $r \in aE$ the value $s''(r)$. The execution of a described above is atomic.

Formally, the modifications of global states by execution of actions are specified by means of the global transition relation Δ:

$$\Delta \subseteq S \times \Sigma \times S$$

(Let us note that the symbol Δ is slightly overloaded since it is used also to denote the family of local transition relations of τ, however the context will always indicate unambiguously the actual meaning of Δ.)

Let $s', s'' \in S$ and $a \in \Sigma$. Then $(s', a, s'') \in \Delta$ if

$$(s'_{|Ea}, s''_{|aE}) \in \delta_a$$

and

$$s''_{|R\setminus(aE)} = s'_{|R\setminus(aE)}$$

Sometimes it is convenient to interpret Δ as a mapping from $S \times \Sigma$ into $\mathcal{P}(S)$, where $\Delta(s, a) = \{s' \in S \mid (s, a, s') \in \Delta\}$ for $s \in S$ and $a \in \Sigma$.

A fat-system τ is *deterministic* if for each $a \in \Sigma$ and each global state $s \in S$ there is at most one global state s' such that $(s, a, s') \in \Delta$. In this case Δ can be interpreted as a partial mapping from $S \times \Sigma$ into S, where $s' = \Delta(s, a)$ iff $(s, a, s') \in \Delta$. Note that τ is deterministic if all relations δ_a are (partial) functions — $\forall a \in \Sigma, \forall s \in \mathbf{F}(Ea; X), \operatorname{card}\{s' \in \mathbf{F}(aE; X) \mid (s, s') \in \delta_a\} \leq 1$, i.e. for each action a and each global state there is at most one local transition of a enabled at this state.

The global transition relation can be extended in the standard way to finite sequences of actions:

$$\Delta \subseteq S \times \Sigma^* \times S$$

by setting $(s, \mathbf{1}, s) \in \Delta$ for each $s \in S$ and $(s', ua, s'') \in \Delta$ iff there exists $s \in S$ such that $(s', u, s) \in \Delta$ and $(s, a, s'') \in \Delta$ for any word $u \in \Sigma^*$ and $a \in \Sigma$.

Example 7.2.1 Let $R = \{r_1, r_2\}$, $\Sigma = \{a, b, c\}$, $X = \{0, 1, 2\}$ and let σ be as on Figure 7.1

Here and in the sequel the elements of R are represented by circles while the elements of Σ by boxes.

The local transition relations of τ are given by the following tables

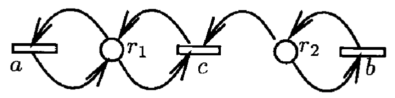


Figure 7.1: A signature σ.

δ_a

r_1	r_1
0	1
1	2

δ_b

r_2	r_2
2	1
1	2

δ_c

r_1	r_2	r_1
2	1	0
1	2	0

In general, for every $a \in \Sigma$, a row of a transition table of δ_a corresponds to a local transition $(s', s'') \in \delta_a$, s' is given by the first part of the row, while s'' is coded by the part after the double vertical line. For example the second row of the transition table for c shows that $(s', s'') \in \delta_c$, where $s' \in \mathbf{F}(\{r_1, r_2\}; X)$, $s'' \in \mathbf{F}(\{r_1\}; X)$ are such that $s'(r_1) = 1$, $s'(r_2) = 2$, $s''(r_1) = 0$. In other words, this transition describes the fact that if r_1 contains 1 and r_2 contains 2 then c is enabled and if this transition is executed then 0 is written to r_1.

Figure 7.2 presents the graph of the global transition relation Δ. (In the graphical representation of Δ global states are given by tuples of values from X, where the i-th element of the tuple gives the value of the register r_i. A directed edge from s' to s'' labelled by $a \in \Sigma$ denotes the fact that $(s', a, s'') \in \Delta$.) This fat-system is deterministic.

$$\begin{tikzcd}
(0,2) \arrow[r, bend left, "b"] \arrow[d, "a"] & (0,1) \arrow[l, "b"] \arrow[d, "a"] & & (0,0) \arrow[r, "a"] & (1,0) \arrow[r, "a"] & (2,0) \\
(1,2) \arrow[u, bend left, "c"] \arrow[r, bend left, "b"] \arrow[rrr, bend right, "a"] & (1,1) \arrow[l, "b"] \arrow[r, "a"] & (2,1) \arrow[lu, "c"] \arrow[r, bend left, "b"] & (2,2) \arrow[l, bend left, "b"]
\end{tikzcd}$$

Figure 7.2: Global transition relation

The main interest in the fat-systems results from their ability to execute some actions in parallel. Intuitively, the execution of an action $a \in \Sigma$, consisting of reading the contents of registers from Ea and modifying the values of registers from aE is atomic. This implies that the execution of another action $b \in \Sigma$ cannot overlap the execution of a if b writes to one of the registers $r \in aE \cup Ea$ used by a. Dependence and potential parallelism between actions are captured by means of binary relations over Σ that are defined below.

With each signature $\sigma = (\Sigma, R, E)$ we associate four binary relations over Σ:
— $C = \{(a,b) \in \Sigma^2 \mid aE \cap Eb \neq \emptyset\}$ — *direct causality* relation,
— $W = \{(a,b) \in \Sigma^2 \mid aE \cap bE \neq \emptyset\}$ — *writing conflict* relation,
— $D = C \cup C^{-1} \cup W$ — *conflict* relation,
— $I = \Sigma^2 \backslash D$ — *independence* relation.

Note that two actions $a, b \in \Sigma$ are independent, $(a,b) \in I$, if $aE \cap bE = aE \cap Eb = Ea \cap bE = \emptyset$. In fact these are well-known Bernstein's conditions [10] specifying the pairs of non-interfering actions that can be executed in any order without altering the result of the computation, actually any pair of independent actions can be executed even simultaneously. This point is discussed in detail in Section 7.3.

Given a fat-system τ we can classify all actions as belonging to one of the following four classes: external, test, reset and internal actions:

- An action $a \in \Sigma$ is *external* if $aE \cup Ea = \emptyset$, i.e. a neither reads nor writes to any register. Note that in this case either δ_a is empty or $\delta_a = \{(\emptyset, \emptyset)\}$, i.e. only one local transition consisting of the pair of the empty mappings is possible for a. In the first case a is never enabled, while in the second case it is always enabled but its execution does not change the global state and, since $\forall b \in \Sigma$, $(a,b) \in I$, a can be executed simultaneously with any other action. Thus a does not interact at all with the system and therefore can be considered as external.

- An action a is a *test* action if $aE = \emptyset$ but $Ea \neq \emptyset$. Then $\delta_a \subseteq \mathbf{F}(Ea; X) \times \{\emptyset\}$, i.e. the second component of each local transition in δ_a is the empty mapping.
  Let $Test_a = \{s' \in \mathbf{F}(Ea; X) \mid (s', \emptyset) \in \delta_a\}$. For any global state $s \in S$, if $s_{|Ea} \in Test_a$ then $\Delta(s,a) = s$, otherwise $\Delta(s,a) = \emptyset$. Thus the action a can be interpreted as a test of values of registers in Ea, if the corresponding partial state belongs to $Test_a$ then a can be executed successfully (but this execution does not change the global state of the system), otherwise a is not enabled and its execution fails.

- Now suppose that $aE \neq \emptyset$ and $Ea = \emptyset$. Then $\delta_a \subseteq \{\emptyset\} \times \mathbf{F}(aE; X)$, i.e. the first component of each local transition in δ_a is the empty mapping. Let $Reset_a = \{s \in \mathbf{F}(aE; X) \mid (\emptyset, s) \in \delta_a\}$. Now for all global states $s_1, s_2 \in S$, $(s_1, a, s_2) \in \Delta$ if and only if $s_{2|aE} \in Reset_a$ and $s_{2|R\backslash aE} = s_{1|R\backslash aE}$. In other words, since a does not read anything, it is always enabled and upon its execution a partial state is chosen from $Reset_a$ and the registers from the set aE are modified according to this partial state. Thus intuitively, a can be interpreted as a *reset* action that can always be executed and whenever activated it resets the values of registers from aE.

- Finally, by *internal* action we mean any action $a \in \Sigma$ such that both aE and Ea are non-empty.

Example 7.2.2 Let $\Sigma = \{a,b,c\}$, $R = \{r_1, r_2\}$, $X = \{0,1\}$. The signature is presented on Figure 7.3.

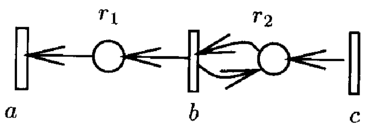


Figure 7.3: A signature σ.

Note that a is a test action, c is a reset action and b is an internal action. The following tables give the transition relations of a, b and c.

δ_a

r_1
0

δ_b

r_2	r_1	r_2
0	0	1
1	1	1

δ_c

r_2
0

Figure 7.4 presents the global transition graph of this system.

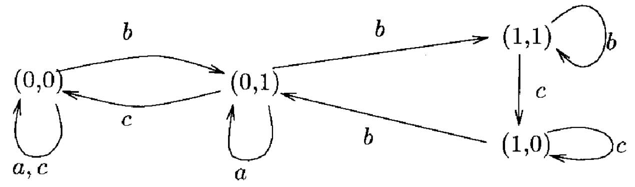


Figure 7.4: Global transition graph

In the rest of the paper we restrict our attention to asynchronous transition systems that have only internal and reset actions. (Since the executions of the other actions does not have any influence on the system state they can be ignored, if necessary it is always possible to include them in the system by the addition of supplementary registers.)

Thus from now on we assume that each signature $\sigma = (\Sigma, R, E)$ satisfies the following condition

$$\forall a \in \Sigma, \ aE \neq \emptyset$$

i.e. each action writes to some register. Let us note that this condition implies that the writing conflict relation W is reflexive, $id_\Sigma \subseteq W$.

The graphical representation of the signatures that is used here is the same as the graphical representation of unmarked Petri nets. This resemblance is in fact intended — actions and registers of fat-systems play the same role as transitions and places in Petri nets and in some cases a direct translation of the fat-systems into (labelled) Petri nets is possible [277]. To some extent fat-systems can be considered as special coloured Petri nets where the values of registers are viewed as coloured tokens. However some significant differences between these models exist. In coloured Petri nets places can contain any number of tokens, in particular they can be empty, while in fat-systems each register contains always exactly one token. Even more notable difference resides in the execution mode. A firing of a transition in a Petri net results in removing some tokens from the input places and adding new tokens to the output places. The execution semantics of fat-systems is quite different — the input registers are only read, their contents is not modified if they are not in the writing domain of the executed action. Implicitly this implies that different actions sharing only their reading registers can be executed simultaneously. Also the contents of the output registers changes in a different way, rather than adding a new token to the existing one a fat-system replaces the old token by a new one.

7.3 Non-Interleaving Semantics of Asynchronous Transition Systems

In order to unify the terminology it is useful to introduce the following notion.

Labelled relational structures are (isomorphism classes) of tuples $(X, R_1, \ldots, R_n, \lambda)$, where X is a set, $R_1, \ldots, R_n$ are binary relations over X and λ is a labelling associating with each element x of X a label $\lambda(x)$.

Such systems can be viewed as vertex labelled graphs with arcs of various types: the set X is the set of vertices, $\bigcup_{i=1}^n R_i$ is the set of arcs, R_i being the set of arcs of type R_i. The relations R_i are not necessarily disjoint, thus an arc (x, y), $x, y \in X$ can belong to several different types at the same time. Two labelled relational structures $(X, R_1, \ldots, R_n, \lambda)$ and $(X', R'_1, \ldots, R'_n, \lambda')$ are isomorphic if they represent the same graph up to a renaming of vertices, i.e. if there exists a bijection $f : X \longrightarrow X'$ such that

$$\forall x \in X,\ \lambda(x) = \lambda'(f(x))$$

and

$$\forall i, 1 \le i \le n,\ \forall x_1, x_2 \in X,\ (x_1, x_2) \in R_i \iff (f(x_1), f(x_2)) \in R'_i$$

In the sequel we identify isomorphic labelled relational structures.

To describe precisely how the asynchronous transition systems work we should present semantics that is capable to reflect the asynchronous and parallel aspects of computations of such systems.

Let us recall that with each signature $\sigma = (\Sigma, R, E)$ we have associated the following conflict relation

$$D = \{(a, b) \in \Sigma^2 \mid aE \cap bE \neq \emptyset \text{ or } aE \cap Eb \neq \emptyset \text{ or } Ea \cap bE \neq \emptyset\}$$

and the independence relation $I = \Sigma^2 \setminus D$. The conflict relation D is obviously symmetric and, since we have assumed that $\forall a \in \Sigma$, $aE \neq \emptyset$, it is also reflexive.

Let $\sim_I$ be the least congruence over the free monoid Σ^* such that $ab \sim_I ba$ for all $(a, b) \in I$. Let $\mathbb{M}(\Sigma, I)$ be the corresponding free partially commutative monoid, i.e. the quotient of the free monoid Σ^* by $\sim_I$. As always the elements of $\mathbb{M}(\Sigma, I)$ are called traces, $[u]_I$ will denote the trace generated by a word $u \in \Sigma^*$.

Intuitively, the relation $\sim_I$ identifies the strings in Σ^* that generate the same computation. In fact a suitable representation of traces shows explicitly which actions can be executed in parallel — this is the main subject of the present section.

Immediately we can note the following remark:

Remark 7.3.1 For all $x, y \in \Sigma^*$ and for all global states $s \in S$ of a fat-system over the signature σ, if $x \sim_I y$ then $\Delta(s, x) = \Delta(s, y)$.

Proof: Direct verification shows that for each pair of independent actions, $(a, b) \in I$, we have $\Delta(s, ab) = \Delta(s, ba)$.

Now note that $x \sim_I y$ iff there exists a sequence $z_0, z_1, \ldots, z_k$ of words of Σ^* such that $x = z_0$, $y = z_k$ and $z_i = w_i a_i b_i v_i$, $z_{i+1} = w_i b_i a_i v_i$ for some $w_i, v_i \in \Sigma^*$ and $(a_i, b_i) \in I$, $i = 0, \ldots, k-1$, therefore $\Delta(s, z_i) = \Delta(s, z_{i+1})$ results immediately from the preceding remark. □

Remark 7.3.1 shows that the global transition relation can be extended to traces:

$$\Delta \subseteq S \times \mathbb{M}(\Sigma, I) \times S$$

by setting for $s, s' \in S$, $t \in \mathbb{M}(\Sigma, I)$, $(s, t, s') \in \Delta$ if $(s, x, s') \in \Delta$ for some (or equivalently for all) $x \in \Sigma^*$ such that $[x]_I = t$. As in the case of words we shall sometimes view Δ as a mapping $\Delta : S \times \mathbb{M}(\Sigma, I) \longrightarrow \mathcal{P}(S)$ or as a partial mapping from $S \times \mathbb{M}(\Sigma, I)$ into S if the fat-system is deterministic.

7.3.1 Processes

Let $\tau = (\Sigma, R, E, X, \Delta)$ be a fat-system. In this section we shall note local transitions $(s', s'') \in \delta_a$, $a \in \Sigma$, of τ as triples (s', a, s''). We begin with the definition of sequential processes.

A sequential process in τ is a sequence

$$p = (s_1, a_1, s'_1), (s_2, a_2, s'_2), \ldots, (s_n, a_n, s'_n)$$

of transitions such that

Seq1: for all i, j $(1 \le i < j \le n)$ and for all $r \in R$, if $r \in Ea_i \cap Ea_j$ and $\forall k$, $(i \le k < j)$, $r \notin a_k E$ then $s_i(r) = s_j(r)$.

Seq2: for all i, j $(1 \leq i < j \leq n)$ and for all $r \in R$, if $r \in a_iE \cap Ea_j$ and $\forall k, i < k < j, r \notin a_kE$ then $s'_i(r) = s_j(r)$,

Intuitively, **Seq1** states that if two actions a_i and a_j read the value of a register r and this register was not modified by any action following a_i and preceding a_j then they should read the same value. Condition **Seq 2** specifies which value a_j reads from a register r, this is the value written by the last action preceding a_j and writing into r. Note that **Seq1** is redundant for actions preceded by some writing action, since in this case **Seq2** implies **Seq1**. However, **Seq1** is necessary if a_i and a_j are not preceded by any action writing into r in order to specify that in this case both actions read the same "initial" value of r.

Example 7.3.2 Let us consider the fat-system of Example 7.2.1. Then $p = u_1u_2u_3u_4u_5u_6u_7$ is a sequential process in τ, where
$u_1 = (\{(r_1, 0)\}, a, \{(r_1, 1)\})$, $u_2 = (\{(r_2, 1)\}, b, \{(r_2, 2)\})$,
$u_3 = (\{(r_1, 1), (r_2, 2)\}, c, \{(r_1, 0)\})$, $u_4 = (\{(r_2, 2)\}, b, \{(r_2, 1)\})$,
$u_5 = (\{(r_1, 0)\}, a, \{(r_1, 1)\})$, $u_6 = (\{(r_1, 1)\}, a, \{(r_1, 2)\})$,
$u_7 = (\{(r_1, 2), (r_2, 1)\}, c, \{(r_1, 0)\})$.

Intuitively, to obtain a concurrent process from a sequential process p we transform each local transition (s_i, a_i, s'_i) of p into the labelled graph presented on Figure 7.5, where $Ea = \{r_{i_1}, \ldots, r_{i_k}\}$, $aE = \{r'_{j_1}, \ldots, r'_{j_n}\}$, $\forall l$ $(1 \leq l \leq k)$ $s_i(r_{i_l}) = x_{i_l}$, $\forall l$ $(1 \leq l \leq n)$ $s'_i(r'_{j_l}) = x'_{j_l}$.

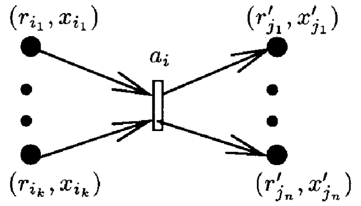


Figure 7.5: Causal dependence relation for a single local transition

The vertices labelled by (r_{i_l}, x_{i_l}) represent the events consisting of reading the value x_{i_l} from register r_{i_l}, the vertices labelled by (r'_{j_l}, x'_{j_l}) represent the events of writing x'_{j_l} into r'_{j_l}, finally the vertex labelled by a_i represents the execution of a_i. Next we simply identify the reading events with the corresponding writing events produced by preceding local transitions. In this way we obtain the causal precedence relation *Caus* over the set of events. As it turns out there are temporal precedence relations between the events that do not result from the causality relation, e.g. when we have two actions writing into the same register that mutually do not communicate directly. This temporal precedence is captured by the relation *Pred*

over the events. The description of *Pred* being a bit tricky, is postponed for a moment.

Now we pass to the formal definition.

A concurrent process is a labelled relational structure $cp = (Events, Caus, Pred, \lambda)$, where

— *Events* is a set of events,

— *Caus* and *Pred* are two relations over *Events*, they are called respectively *direct causality* and *direct precedence* relations,

— $\lambda : Events \longrightarrow \Sigma \cup (R \times X)$ is a labelling of events.

Intuitively, if an event $e \in Events$ is labelled by $a \in \Sigma$, i.e., $\lambda(e) = a$, then e represents an execution of the action a in cp, otherwise, if $\lambda(e) = (r, x) \in R \times X$ then e represents a state of r, where r contains the value x.

The relation *Caus* describes the causal dependence between events, if $z_1 Caus^* z_2$, $z_1 \neq z_2$, then z_1 is, possibly indirect, cause of z_2. The relation *Pred* describes the necessary temporal relations between events, if $z_1 Pred^* z_2$, $z_1 \neq z_2$, then the event z_1 precedes z_2. We always have the inclusion $Caus \subseteq Pred$, i.e. causality implies temporal precedence.

Instead of giving a list of condition characterizing concurrent processes we describe how a sequential process can be transformed into a concurrent one. This transformation will be denoted by Ψ.

Let
$$p = (s_1, a_1, s'_1), \ldots, (s_n, a_n, s'_n) \tag{7.1}$$

be a sequential process. The corresponding concurrent process is the labelled relational structure

$$cp = \Psi(p) = (Events, Caus, Pred, \lambda)$$

The set *Events* is the union of two disjoint sets $Events_\Sigma$ and $Events_R$. The set $Events_\Sigma$ consists of exactly n elements:

$$Events_\Sigma = \{e_1, \ldots, e_n\}$$

where $n = |p|$ is the number of transition occurrences in p. An event $e_i \in Events_\Sigma$ represents the action occurrence a_i, hence

$$\lambda(e_i) = a_i, \quad 1 \leq i \leq n$$

The set $Events_R$ is the union

$$Events_R = \bigcup_{r \in R} Events_r$$

where

$$Events_r = \lambda^{-1}(\{r\} \times X), \quad r \in R$$

is the set of events representing consecutive states of the register r. This set is obtained in the following way.

Let $\Pi_r : \Sigma^* \longrightarrow \Sigma^*$ be the erasing morphism preserving only actions writing into r:

$$\Pi_r(a) = \begin{cases} a & \text{if } (a,r) \in E \\ \mathbf{1} & \text{otherwise} \end{cases}$$

Let $m_r = |\Pi_r(a_1 \dots a_n)|$, i.e. there are exactly m_r action occurrences in p writing into r. Then

$$Events_r = \begin{cases} \{w_{r0}, w_{r1}, \dots, w_{rm_r}\} & \text{if } \exists k \ (1 \le k \le n) \ (r, a_k) \in E \text{ and} \\ & \Pi_r(a_1 \dots a_{k-1}) = \mathbf{1} \\ \{w_{r1}, \dots, w_{rm_r}\} & \text{otherwise} \end{cases}$$

Intuitively, w_{rk}, $1 \le k \le m_r$, represents the state of r after k-th writing into r. If there is k, $1 \le k \le n$, such that the k-th transition of p reads r but no preceding transition writes into r then $Events_r$ contains also the event w_{r0} corresponding to the "initial" state of r.

For each k, $1 \le k \le m_r$, we set

$$i_k = \min\{j \mid 1 \le j \le n \text{ and } |\Pi_r(a_1 \dots a_j)| = k\} \tag{7.2}$$

From the definition of Π_r it follows that $(a_{i_k}, r) \in E$ and, in fact, a_{i_k} is the k-th action occurrence writing into r in p. The label of w_{rk} is defined by

$$\lambda(w_{rk}) = (r, s'_{i_k}(r))$$

i.e. w_{rk} is labelled by a pair (r, x), where $x \in X$ is the k-th value written into r. If $w_{r0} \in Events_r$ then we set additionally

$$\lambda(w_{r0}) = (r, s_k(r))$$

for any k such that $(r, a_k) \in E$ and $\Pi_r(a_1 \dots a_{k-1}) = \mathbf{1}$. (Note that if there are several actions reading the value of r before this value is overwritten by other actions of p then **Seq1** assures that all of them read the same value, i.e. this definition is consistent.)

The relation $Caus$ is the union of two relations

$$Caus = Read_R \cup Write_R$$

where

$$Read_R = \bigcup_{r \in R} Read_r \text{ and } Write_R = \bigcup_{r \in R} Write_r$$

The relation $Write_r$ codes the direct causality relation between action occurrences writing into r and events representing the state of r immediately after this writing:

$$Write_r = \{(e_{i_k}, w_{rk}) \in Events_\Sigma \times Events_r \mid 1 \le k \le m\}$$

where i_k is given by Eq. 7.2.

The relation $Read_r$, $r \in R$, consists of pairs of events (w_{rj}, e_l) such that a_l is the action occurrence reading the state of r represented by w_{rj}:

$$Read_r = \{(w_{rj}, e_l) \in Events_r \times Events_\Sigma \mid (r, a_l) \in E \text{ and } j = |\Pi_r(a_1 \dots a_{l-1})|\}$$

Note that in particular if $\Pi_r(a_1 \dots a_{l-1}) = \mathbf{1}$ and $r \in Ea_l$ then $(w_{r0}, e_l) \in Read_r$, i.e. a_l reads the initial value of r.

As an example we show on Figure 7.6 the concurrent process $\Psi(p) = (Events, Caus, Pred, \lambda)$ obtained for the sequential process p from Example 7.3.2.

From now on we assume that the following rule is adopted in the graphical representation of concurrent processes: continuous arcs represent the elements of the direct causality relation $Caus$, while dashed arcs represent the elements of $Pred \setminus Caus$, i.e. the pairs of events that should be added to $Caus$ to obtain the direct precedence relation $Pred$; since always $Caus \subseteq Pred$ this representation is unambiguous. (The relation $Pred$ is defined below.)

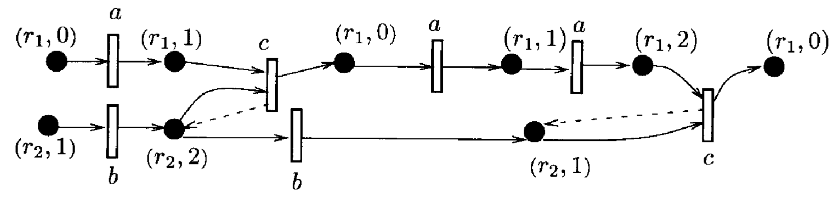


Figure 7.6: The concurrent process $\Psi(p)$.

The direct precedence relation $Pred$ is the union

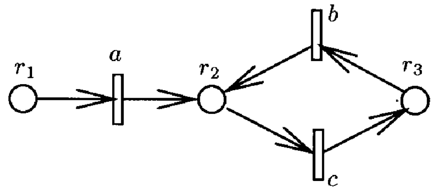
$$Pred = Caus \cup Before_R \cup While_R$$

where

$$Before_R = \bigcup_{r \in R} Before_r \text{ and } While_R = \bigcup_{r \in R} While_r$$

To introduce the relation $Before_R$ let us consider the following example:

Example 7.3.3 Let



be a signature of a fat-system with the following local transitions relations

δ_a

r_1	r_2
1	1

δ_b

r_3	r_2
2	2

δ_c

r_2	r_3
1	2
2	0

Let $u_a = (\{(r_1,1)\}, a, \{(r_2,1)\})$, $u_b = (\{(r_3,2)\}, b, \{(r_2,2)\})$, $u_1 = (\{(r_2,1)\}, c, \{(r_3,2)\})$, $u_2 = (\{(r_2,2)\}, c, \{(r_3,0)\})$ and let $p_1 = u_a u_b, p_2 = u_b u_a$ be two sequential processes in this system. Both processes yield the same causality relation $(Events, Caus, \lambda)$ that is presented on Figure 7.7.

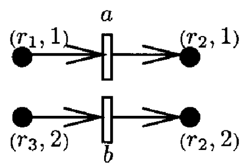


Figure 7.7: Causality relation for the sequential processes p_1 and p_2.

Note that the events generated by the transition u_a are causally unrelated to the events generated by the transition u_b, actually neither the value written by a is used by b nor the result of execution of b is used by a. Although, the transitions u_a and u_b are completely causally unrelated, their order is important for the final state of the system — in p_1 the final value of r_2 is 2, in p_2 it is 1. The difference between p_1 and p_2 is even more evident if we remark that $u_a u_b u_2$ and $u_b u_a u_1$ are valid sequential processes while $u_a u_b u_1$ and $u_b u_a u_2$ are not valid. The causality relations for $u_a u_b u_2$ and $u_b u_a u_1$ are presented on Figure 7.8.

From this example we see that the information provided by the relation *Caus* is somehow incomplete. For these reason we add information about the temporal precedence of events which is captured by the relation *Pred*. In fact in the processes p_1 and p_2 in Example 7.3.3 actions a and b cannot be executed in parallel since u_a and u_b modify r_2, in p_1 we cannot start the execution of b before the modification of r_2 by a is completed, which yields the situation presented on Figure 7.9.

The temporal precedence just described is captured by the relations $Before_r$, $r \in R$. Intuitively, $Before_r$ consists of all pairs $(w_r, e) \in Events_r \times Events_\Sigma$ such that the action occurrence e operates on r (i.e., either reads r or writes into r) and w_r is the last writing event for r preceding e. Thus $(w_r, e) \in Before_r$ represents the fact that writing into r should be completed before the next action operating on r can be executed

$$\begin{aligned} Before_r \;=\; & \{(w_{rj}, e_l) \in Events_r \times Events_\Sigma \mid r \in a_l E \cup E a_l \text{ and} \\ & |\Pi_r(a_1 \ldots a_{l-1})| = j\} \end{aligned}$$

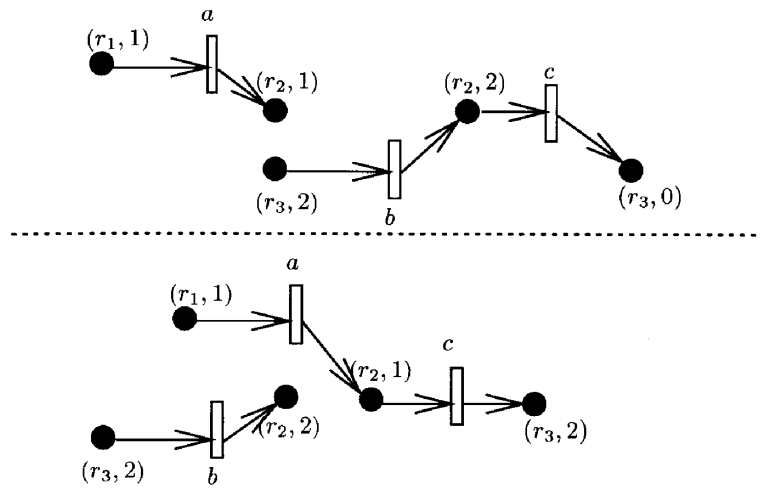


Figure 7.8: Causality relations for $u_a u_b u_2$ and $u_b u_a u_1$.

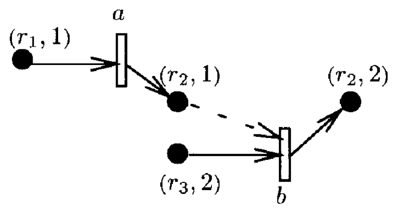


Figure 7.9: *Caus* and $Before_R$ relations for p_1

Note that always $Read_r \subseteq Before_r$. To complete the definition we set $Before_R = \bigcup_{r \in R} Before_r$.

The following remark shows that the relation $Caus \cup Before_R$ enables to order totally all occurrences of an action $a \in \Sigma$ and all events corresponding to the states of a register $r \in R$ in any concurrent process.

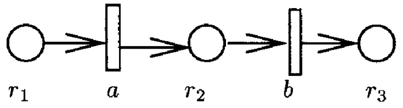
Remark 7.3.4 Suppose that $Events, Caus, \lambda, Before_R$ are defined as above relatively to a sequential process p. Then for all $a \in \Sigma$ and all $r \in R$, the sets $\lambda^{-1}(a)$ and $Events_r = \lambda^{-1}(\{r\} \times X)$ are totally ordered by $(Caus \cup Before_R)^*$.

Proof: Let $e_i, e_j \in \lambda^{-1}(a)$ be two consecutive occurrences of a in a sequential process p and let $r \in aE$ be a register in the writing domain of a. Then there exists an event $w_{rk} \in Events_r$ such that $(e_i, w_{rk}) \in Write_r$ and $(w_{rk}, e_j) \in Before_r$. Thus $(e_i, e_j) \in (Write_R \cup Before_R)^2$. Similarly we show the second assertion of the remark.

□

As it turns out the relation $Caus \cup Before_R$ does not yet reflect all temporal dependencies between elements of $Events$.

Example 7.3.5 Let us consider a fat-system with the signature



and the following transition relations

δ_a

r_1	r_2
0	1
1	0

δ_b

r_2	r_3
0	0
1	1

Let $u_1 = (\{(r_2,0)\}, b, \{(r_3,0)\})$ and $u_2 = (\{(r_1,1)\}, a, \{(r_2,0)\})$ and let us consider the sequential processes $p_1 = u_1u_2$ and $p_2 = u_2u_1$. Figure 7.10 represents the relations $Caus$ and $Before_R$ induced by p_1, while Figure 7.11 gives the same relations for p_2.

Note that Figure 7.11 indicates clearly that a is executed before b, while Figure 7.10 may suggest that a and b can be executed in parallel. However, since a modifies r_2 it cannot be executed until the action b operating also on r_2 is completed.

To complete the definition of the direct precedence relation $Pred$ we should closely analyse the situation. First let us note that the events of $Events_\Sigma$ and of $Events_R$ are of quite different nature. Each element e of $Events_\Sigma$ represents an action occurrence, i.e. an event that is instantaneous (or at least that can be considered as such by the assumption of atomicity of the action execution). On the other hand, the elements of $Events_R$ represent states of registers, i.e. by their very nature

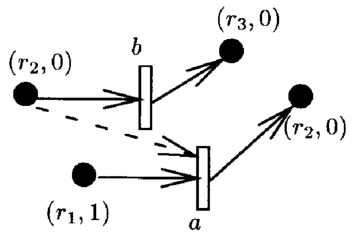


Figure 7.10. Relations $Caus$ and $Before_R$ for p_1 (dashed arrow represents an element of $Before_R \setminus Caus$).

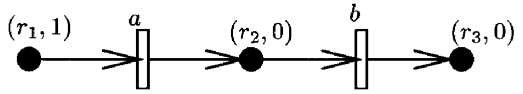


Figure 7.11. $Caus$ and $Before_R$ relations for p_2 (here $Before_R \subseteq Caus$, therefore $Before_R$ is not visible on the picture).

they are not instantaneous but rather span over some period of time. This point of view can also be justified by the following consideration. As it was mentioned previously, intuitively, elements of $Events_R$ are obtained by an identification of events consisting in writing into registers with subsequent matching reading events. While both writing and reading regarded separately can be considered as instantaneous (atomic), we cannot pretend by any means that the resulting composed events are instantaneous, they rather last during some time.

Let us consider specifically a local transition $(s, s') \in \delta_a$ of $a \in \Sigma$ and a sequential process $p' = (s, a, s')$ consisting of just this one transition. The causality relation of the corresponding concurrent process $\Psi(p')$ was presented on Figure 7.5. While this graph represents accurately the causality relation between events, it may not reflect all possible temporal dependencies. Suppose that there is a register $r \in (Ea)\setminus(aE)$. Then $Events_r = \{w_{r0}\}$, $Events_\Sigma = \{e_1\}$, where $\lambda(w_{r0}) = (r, s(r))$ and $\lambda(e_1) = a$. Moreover, $(w_{r0}, e_1) \in Caus$ which implies that $(w_{r0}, e_1) \in Pred$, i.e. w_{r0} precedes e_1 in the temporal order. However in some sense w_{r0} also coexists with e_1 and immediately follows e_1 in the temporal order. In fact, during the execution of a no other action can write into r, therefore during this execution and immediately after it the value of r remains unchanged and we can assume that also $(e_1, w_{r0}) \in Pred$. Thus the parallel process $\Psi((s, a, s'))$ should be represented by the labelled relational structure given on Figure 7.12.

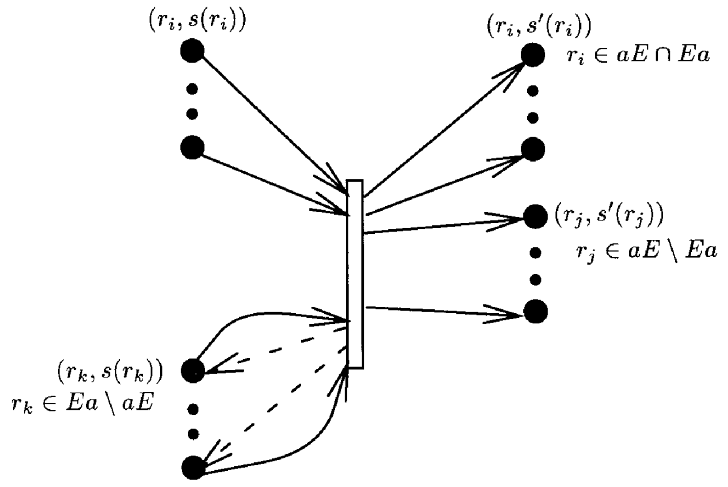


Figure 7.12: Concurrent process generated by one transition (s, a, s').

In general for a sequential process p of the form (7.1) the precedence relation discussed above is captured by the relations $While_r$:

$$While_r = U_r \cup U_r^{-1}, \text{ where}$$
$$U_r = \{(w_{rj}, e_k) \in Events_r \times Events_\Sigma \mid \ r \in (Ea_k)\setminus(a_kE) \text{ and } j = |\Pi_r(a_1 \ldots a_{k-1})|\}$$

Now we set

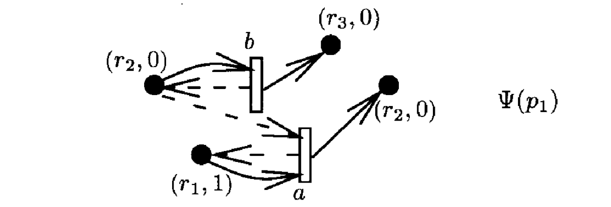
$$While_R = \bigcup_{r\in R} While_r$$

and we can complete the definition of concurrent processes by defining

$$Pred = Caus \cup Before_R \cup While_R$$

Note that $U_r \subsetneq Write_r$ and if $U_r \neq \emptyset$ for some $r \in R$ then the relation $Pred$ is not acyclic.

We shall illustrate the complete definition of processes with a few examples. First, to terminate Example 7.3.5 we give on Figure 7.13 the representation of the concurrent processes $\Psi(p_1)$ and $\Psi(p_2)$.



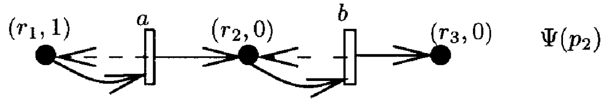


Figure 7.13: Concurrent processes $\Psi(p_1)$ and $\Psi(p_2)$.

Example 7.3.6 Let τ be a fat-system with the signature σ presented on Figure 7.14.

The transition relations of τ are presented below:

δ_a

r_1	r_3	r_1
0	0	1
1	1	0

δ_b

r_3	r_2
0	1
1	0

δ_c

r_1	r_2	r_3
1	1	1
0	0	0

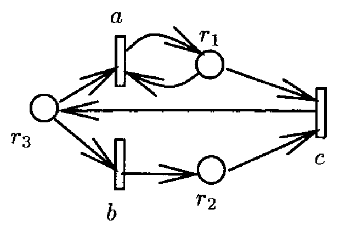


Figure 7.14: A signature σ.

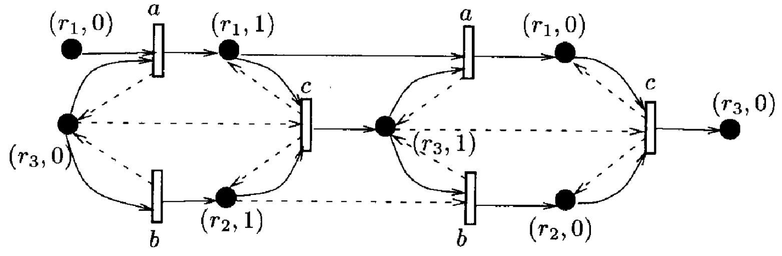


Figure 7.15: A concurrent process.

Figure 7.15 represents a concurrent process in τ.

At the end let us note that there is also an alternative possibility of defining the concurrent processes as equivalence classes of sequential processes.

Let Tr be the set of local transitions of a fat-system τ and let I be the independence relation over Σ induced by the signature of τ. This relation extends naturally to Tr: two transitions $u_1 = (s_1, a, s_1')$ and $u_2 = (s_2, b, s_2')$ are said to be independent, $(u_1, u_2) \in I$, if the underlying actions are independent, $(a, b) \in I$. Analogously we can define the relation $\sim_I$ over the set Tr^* of sequences of transitions as the smallest congruence such that $\forall (u_1, u_2) \in I, \quad u_1 u_2 \sim_I u_2 u_1$. Let $\Pi : Tr^* \longrightarrow \Sigma^*$ be the morphism mapping each transition (s, a, s') to the underlying action a. Then for all sequences $p_1, p_2 \in Tr^*$ of transitions, $p_1 \sim_I p_2$ iff $\Pi(p_1) \sim_I \Pi(p_2)$. As it turns out for each sequential process p in τ we can identify the corresponding concurrent process $\Psi(p)$ with the equivalence class of p under $\sim_I$:

Proposition 7.3.7 *(i) Let p be a sequential process in τ and let $p' \in Tr^*$ be any sequence of transitions such that $p \sim_I p'$. Then p' is a sequential process.*
(ii) Let p, p' be two sequential processes in τ. Then $p \sim_I p'$ if and only if $\Psi(p) = \Psi(p')$.

Proof: Straightforward but tedious induction on the length of p. □

Proposition 7.3.7 implies that we can identify concurrent processes with the equivalence classes $[p]_I$ of $\sim_I$ for p ranging over sequential processes. This definition would have had the advantage of being technically simpler than the definition of the labelled relational structures $\Psi(p)$, which is rather complicate. Nevertheless, we have preferred to introduce the concurrent processes by means of $\Psi(p)$ since this representation reflects better the intuitive ideas lying behind this concept — it provides explicitly the causality and temporal precedence relations between events giving direct insight into the behaviour of the fat-systems.

7.3.2 Semantics Based on Partial Order of Action Occurrences

In this subsection we introduce non-interleaving semantics of the fat-systems that takes into account only the action occurrences. As we shall see below cp-graphs representing the behaviour of the fat-systems in this semantics are directly related to traces, actually they constitute a representation of traces suitable for the fat-systems.

Let $\sigma = (\Sigma, R, E)$ be a fixed signature and let x be a word of Σ^*, $x = a_1 a_2 \dots a_n$. A *causal and precedence graph* (cp-graph for short) for x is the labelled relational structure $\Psi_\sigma(x) = (Ev_x, Ca_x, Pr_x, \lambda_x)$, where

— $Ev_x = \{e_1, \dots, e_n\}$ is the set of events, $\mathrm{card}(Ev_x) = |x|$,

— $\lambda_x : Ev_x \longrightarrow \Sigma$ is a labelling such that $\forall i, \ (1 \le i \le n), \ \lambda(e_i) = a_i$,

— Ca_x and Pr_x are acyclic relations over Ev_x called respectively *direct causality relation* and *direct precedence relation.*

The direct causality relation is defined in the following way:

$$Ca_x = \{(e_i, e_j) \in Ev_x \times Ev_x \mid \begin{array}{l} 1 \le i < j \le n \text{ and} \\ \exists r \in a_i E \cap E a_j,\ \forall l\ (i < l < j)\ r \notin a_l E\} \end{array}$$

The direct precedence relation is defined in the following way:
$(e_i, e_j) \in Pr_x$ if $1 \le i < j \le n$ and at least one of the following conditions holds:

- $\exists r \in a_i E \cap E a_j,\ \forall l\ (i < l < j)\ r \notin a_l E$
- $\exists r \in a_i E \cap a_j E,\ \forall l\ (i < l < j)\ r \notin a_l E$
- $\exists r \in E a_i \cap a_j E,\ \forall l\ (i \le l < j)\ r \notin a_l E$

The elements of Ev_x correspond to the occurrences of actions in the word x, e_i being the occurrence of a_i. A pair of events (e_i, e_j) is in Ca_x if for some register r the action occurrence $a_i = \lambda_x(e_i)$ writes into r a value that is subsequently read by the action occurrence $a_j = \lambda_x(e_j)$, i.e. no other action between a_i and a_j overwrites the value written into r by a_i (intuitively, there is a direct communication from a_i to a_j by means of the register r).

The direct precedence relations Pr_x consists of pairs of events (e_i, e_j) that are in conflict for some register r. Three cases are distinguishable:

- the action occurrence a_i writes into r a value that is subsequently read by the action occurrence a_j,
- the action occurrences a_i and a_j write to the same register and a_j overwrites the value written by a_i (there is no action occurrence between a_i and a_j writing into r),
- the action occurrence a_i reads the value of r and a_j is the first subsequent action occurrence modifying r.

Note that the first of these conditions corresponds to the definition of Ca_x, i.e. $Ca_x \subseteq Pr_x$.

Intuitively, Ca_x illustrates how the information passes from one action occurrence to another during the execution of x by any fat-system with the signature σ while $Prec_x$ represents the the pairs of action occurrences that are in conflict over some register and for this reason their order is significant for the final outcome of the execution and cannot be altered. Taking the reflexive and transitive closures of Ca_x and Pr_x we obtain partial orders over the set Ev_x: $(e_i, e_j) \in Ca_x^*$ if e_i is (possibly indirect) cause of e_j and $(e_i, e_j) \in Pr_x^*$ if e_i necessarily precedes e_j. If neither $(e_i, e_j) \in Pr_x^*$ nor $(e_j, e_i) \in Pr_x^*$ then the events e_i and e_j can occur simultaneously or in any order.

cp-graphs are closely related to concurrent processes. Let τ be a fat-system over a signature σ. Let $p = u_1 \dots u_n$ be a sequential process in τ, where $u_i = (s_i, a_i, s_i')$ and let $\Psi(p) = (Events, Caus, Pred, \lambda)$ be the corresponding concurrent process. It is easy to see that $\Psi(p)$ determines (up to isomorphism) the cp-graph $\Psi_\sigma(y)$ of the sequence $y = a_1 \dots a_n$ of actions of p.

Example 7.3.8 Let σ be the signature of Example 7.2.1 and let $x = abcbabac$. The cp-graph $\Psi_\sigma(x)$ is presented on Figure 7.16. As in the case of parallel processes, continuous arcs show elements of Ca_x while dashed arcs indicate the elements of Pr_x that are not in Ca_x.

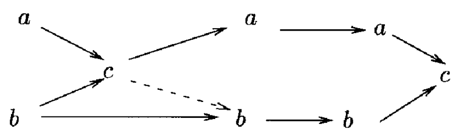


Figure 7.16: cp-graph $\Psi_\sigma(abcbaac)$

As it turns out the cp-graphs constitute in fact a representation of traces:

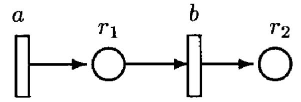
Proposition 7.3.9 *Let σ be a signature, $x, y \in \Sigma^*$, $\Psi_\sigma(x) = (Ev_x, Ca_x, Pr_x, \lambda_x)$, $\Psi_\sigma(y) = (Ev_y, Ca_y, Pr_y, \lambda_y)$ and finally let I be the independence relation induced by σ. Then the following facts are equivalent*

- $x \sim_I y$,
- $\Psi_\sigma(x) = \Psi_\sigma(y)$ *(i.e. the cp-graphs of x and y are isomorphic),*
- (Ev_x, Pr_x, λ_x) *and* (Ev_y, Pr_y, λ_y) *are isomorphic labelled graphs,*

Proof: Straightforward induction on the length of words. □

One may wonder why we use the cp-graph $\Psi_\sigma(x)$ to characterize the trace $[x]_I$, as Proposition 7.3.9 shows that the trace is characterized unambiguously by the triple (Ev_x, Pr_x, λ_x) and therefore the relation Ca_x seems to be useless. However, the relation Ca_x that is essential for the characterization of the runs of the asynchronous automata over traces and therefore determines the recognizability power of asynchronous automata (cf. Section 7.6). On the other hand the relation Pr_x is also necessary since Ca_x cannot in general characterize traces as the following example shows.

Example 7.3.10 Let σ be the following signature:



Then the traces $[y]_I$ and $[z]_I$, where $y = aba$, $z = aab$, are different but (Ev_y, Ca_y, λ_y) and (Ev_z, Ca_z, λ_z) are isomorphic labelled relational structures.

7.4 Asynchronous Cellular Transition Systems

In this section we introduce a special subclass of fat-systems — asynchronous cellular transition systems. This class is characterized by a particularly simple form of signatures. The simplicity of asynchronous cellular transition systems often facilitates formal reasoning and enables better insight into constructions carried out in the next sections. The importance of this class is also emphasizes by the fact that, as we shall see in Section 7.5, a large class of fat-systems is equivalent with asynchronous cellular transition systems, which permits to focus our attention on the latter class in the sequel.

Definition 7.4.1 *A signature $\sigma = (\Sigma, R, E)$ is simple if $\forall a \in \Sigma, \mathrm{card}(aE) = 1$ and the mapping associating with each action $a \in \Sigma$ the unique register of aE is a bijection between Σ and R.*

Formally, *finite asynchronous cellular transition systems* (or fact-systems in short) are asynchronous transition systems with simple signatures.

Up to now we have assumed that the sets Σ and R are disjoint, however in the case of fact-systems it is convenient to identify every action a with the unique register r_a that a can modify, $aE = \{r_a\}$.

Let $\tau = (\Sigma, R, E, X, \Delta)$ be a finite asynchronous cellular transition system. Let $C = \{(a,b) \in \Sigma \mid aE \cap Eb \neq \emptyset\}$ be the direct causality relation induced by σ. Let us note that $E \cap (R \times \Sigma) = \{(r_a, b) \in R \times \Sigma \mid (a,b) \in C\}$, while $E \cap (\Sigma \times R) = \{(a, r_a) \mid a \in \Sigma\}$. Thus we see that if we identify every register r_a with a then $E \cap (\Sigma \times R)$ becomes the identity relation while $E \cap (R \times \Sigma)$ collapses to C.

Let $s \in \mathbf{F}(\alpha; X)$, $\alpha \subseteq R$, be a partial state and let $\alpha' = E\alpha = \{a \in \Sigma \mid r_a \in \alpha\}$. Then s will be identified with the mapping $s' \in \mathbf{F}(\alpha'; X)$ such that $s'(a) = s(r_a)$, in particular, for each $a \in \Sigma$, $\mathbf{F}(Ea; X)$ is replaced by $\mathbf{F}(Ca; X)$. Therefore, instead of speaking of the state of register r_a we can speak about the state of the cell $a \in \Sigma$. (In the case of fact-systems we use double terminology when speaking about elements of Σ— we call them cells if we have in mind the registers associated with $a \in \Sigma$, or actions if we consider them rather as actions in the system).

Let us consider the set $\mathbf{F}(aE; X) = \mathbf{F}(\{r_a\}; X)$. Now this set is replaced by $\mathbf{F}(\{a\}; X)$. However, the set of mappings from a one element domain is naturally isomorphic with the co-domain X, under this isomorphism each $s \in \mathbf{F}(\{a\}; X)$ is identified with the value $x = s(a)$. Therefore we replace the local transition relations $\delta_a \subseteq \mathbf{F}(Ea; X) \times \mathbf{F}(aE; X)$, $a \in \Sigma$ by the relations

$$\delta_a \subseteq \mathbf{F}(Ca; X) \times X, \quad a \in \Sigma \tag{7.3}$$

In conclusion, an asynchronous cellular transition system can be viewed as a quadruple

$$\tau = (\Sigma, C, X, \Delta)$$

where

— Σ is the set of cells (actions),

— C is a binary relation over Σ,

— X is the set of values (local states),

— $\Delta = \{\delta_a \mid a \in \Sigma\}$ is the set of transition relations that have the form defined by (7.3).

As previously the global states of τ are mappings from Σ into X and $S = \mathbf{F}(\Sigma; X)$ will stand for the set of global states.

The global transition relation

$$\Delta \subseteq S \times \Sigma \times S$$

is defined in the following way:

for all $s', s'' \in S$ and $a \in \Sigma$, $(s', a, s'') \in \Delta$ if

$$(s'_{|Ca}, s''(a)) \in \delta_a \text{ and} \tag{7.4}$$

$$\forall b \in \Sigma \setminus \{a\},\ s''(b) = s'(b) \tag{7.5}$$

To maintain the analogy with fat-systems the pair $\sigma = (\Sigma, C)$ will be called the signature of the fact-system τ.

We can imagine an asynchronous cellular transition system in the following way. Let $\mathcal{G} = (\Sigma, C)$ be the (directed) graph of the relation C. A global state $s \in \mathbf{F}(\Sigma; X)$ associates with each vertex $a \in \Sigma$ of $\mathcal{G}$ its state $s(a) \in X$. How an action a is executed in this automaton? Intuitively, a reads the states of all vertices b such that there is an arc from b to a in $\mathcal{G}$ ($(b,a) \in C$), which yields a mapping $s \in \mathbf{F}(Ca; X)$ — the partial state of the system induced by the neighbourhood of a. Next a chooses a value $x \in X$ such that $(s, x) \in \delta_a$ and takes this value as its new state. (If for all $x \in X$, $(s,x) \notin \delta_a$ then a is not enabled and cannot be executed.) Thus the execution of a does not change the states of the other cells (cf. 7.5), and the new state of a depends only on the states of its neighbours from Ca (cf. 7.4).

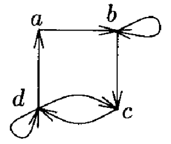
Let us note that in fact-systems the writing conflict relation W is the identity thus the conflict relation has the form

$$D = C \cup C^{-1} \cup id_{\Sigma}$$

i.e. two different actions cannot be executed simultaneously iff their cells are adjacent in the graph $\mathcal{G}$.

From now on we shall always note fact-systems as quadruples $\tau = (\Sigma, C, X, \Delta)$ defined above. Thus formally fact-systems have different form than fat-systems, nevertheless we continue to view them as a special kind of fat-systems since we can always recover the underlying fat-system by reintroducing a register r_a for each action $a \in \Sigma$.

Example 7.4.2 Let C be the binary relation over $\Sigma = \{a, b, c, d\}$ presented below and let $X = \{0, 1\}$.



We shall consider an fact-system with the following local transition relations

δ_a

d	a
0	1
1	0

δ_b

a	b	b
1	0	1
0	1	0

δ_c

b	d	c
0	0	1
1	1	0

δ_d

c	d	d
1	0	1
0	1	0

The graph of the global transition relation for this system is presented on Figure 7.17.

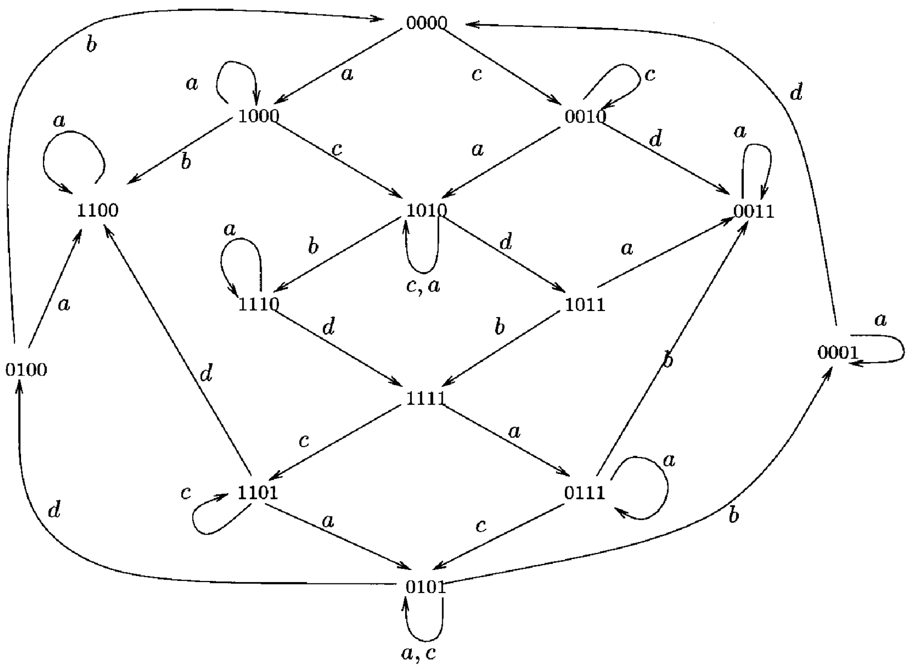


Figure 7.17. Global transition relation. Each global state s is represented by the vector $(s(a), s(b), s(c), s(d))$ of four values of cells a, b, c, d.

At the end let us note that cp-graph semantics for fact-systems has particularly simple form. Let $\sigma = (\Sigma, C)$ be a signature of a fact-system and let $y = a_1 \dots a_n$. Then the cp-graph $\Psi_\sigma(y)$ for y is the labelled relational structure

$$\Psi_\sigma(y) = (Ev_y, Ca_y, Pr_y, \lambda_y)$$

where
— $Ev_y = \{e_1, \dots, e_n\}$, $\mathrm{card}(Ev_y) = n$,
— $\lambda_y(e_i) = a_i$ for $1 \le i \le n$,
— $Ca_y = \{(e_i, e_j) \in Ev_y \times Ev_y \mid i < j$ and $(a_i, a_j) \in C$ and $\forall k\ (i < k < j)\ a_i \neq a_k \neq a_j\}$
— $Pr_y = \{(e_i, e_j) \in Ev_y \times Ev_y \mid i < j$ and $(a_i, a_j) \in D$ and $\forall k\ (i < k < j)\ a_i \neq a_k \neq a_j\}$, where $D = C \cup C^{-1} \cup id_\Sigma$ is the conflict relation induced by σ.

As in the general case of fat-systems, $Ca_y \subseteq Pr_y$ and both these relations are acyclic.

7.5 Simulations Between Asynchronous Transition Systems

In this section we examine the question of how to simulate one fat-system by another. First we fix the terminology.

A subset U of the set S of global states of a fat-system τ is *closed under transitions* if $\forall s \in U, \forall a \in \Sigma, \Delta(s, a) \subseteq U$.

Let τ_i, $i = 1, 2$, be fat-systems over the same alphabet Σ and let S_i, Δ_i be respectively the sets of global states and the global transition relations of τ_i.

Definition 7.5.1 *The fat-system τ_2* covers *τ_1 if there exists a subset U of S_2 closed under transitions and a surjective mapping (called* covering *of τ_2) $c : U \longrightarrow S_1$ such that $\forall s \in U, \forall a \in \Sigma$, c maps bijectively the set $\Delta(s, a)$ onto $\Delta(c(s), a)$.*

Intuitively the covering relation describes the simulation of τ_2 by τ_1.

Proposition 7.5.2 *Let τ_i, $i = 1, 2$, be two fat-systems and let $c : U \longrightarrow S_1$ be a covering of τ_1 by τ_2, $U \subseteq S_2$. Then*
(i) for all $s' \in U$, for all $x \in \Sigma$, if $s'' \in \Delta_2(s', x)$ then $s'' \in U$ and $c(s'') \in \Delta_1(c(s'), x)$
(ii) for all $s_1', s_1'' \in S_1$, for all $x \in \Sigma^*$, if $s_1'' \in \Delta_1(s_1', x)$ then $\forall s_2' \in c^{-1}(s_1')$, $\exists s_2'' \in c^{-1}(s_1'')$, $s_2'' \in \Delta_2(s_2', x)$

Proof: Straightforward. □

The covering relation is transitive, if τ_1 is covered by τ_2 and τ_2 is covered by τ_3 then τ_3 covers τ_1. It is also reflexive, τ covers itself by the identity mapping. Not all covering mappings are of interest, for example each fat-system τ_1 is covered by a sequential fat-system τ_2 with the signature $(\Sigma, \{r\}, \Sigma \times \{r\} \cup \{r\} \times \Sigma)$, τ_2 stores in r the global state of τ_1. This trivial covering replacing a distributed system by its sequential simulation is not very useful. The coverings that are of interest should not diminish the degree of parallelism of the distributed system — we shall call them faithful.

The rest of this section is devoted to simulations between fat-systems and fact-systems.

One of the main structural properties of fact-systems is the absence of writing conflicts between distinct actions. As it turns out this property is essential since a fat-system τ_1 can be simulated faithfully by an fact-system τ_2 if in τ_1 all writing conflicts between different actions are in some sense redundant.

Let $\sigma = (\Sigma, R, E)$ be a fixed signature. Let us recall that two distinct actions $a, b \in \Sigma$ are in writing conflict if $aE \cap bE \neq \emptyset$. We say that this writing conflict is *redundant* if $(a,b) \in C \cap C^{-1}$, where $C = (E \circ E) \cap \Sigma^2$ is the direct causality relation induced by σ.

In the sequel we shall study signatures σ where all writing conflicts between distinct actions are redundant, i.e. we assume that the condition

$$W \setminus id_\Sigma \subseteq C \cap C^{-1} \qquad (7.6)$$

holds, where W and C are respectively the writing conflict and the direct causality relation induced by σ. A signature satisfying (7.6) will be called *w-redundant.*

Example 7.5.3 Let us consider the signature presented on Figure 7.18.

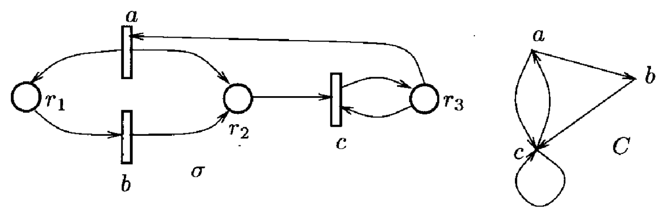


Figure 7.18. A signature with a non-redundant writing conflict relation and its direct causality relation.

The actions a and b are in writing conflict since they write into the same register r_2. The action a can send informations directly to b by means of the register r_1, $(a,b) \in C$, however $(b,a) \notin C$, i.e. this writing conflict is not redundant.

Proposition 7.5.4 *Let $\sigma = (\Sigma, R, E)$ be a w-redundant signature and let $\tau = (\Sigma, R, E, X, \Delta)$ be an asynchronous transition system over σ. Let $C = (E \circ E) \cap \Sigma^2$ be the direct causality relation induced by σ. Then there exists an asynchronous cellular transition system $\tau' = (\Sigma, C, X', \Delta')$ covering τ.*

Proof: In our simulation each local state of a cell $a \in \Sigma$ of the constructed fact-system τ' will consist of two components f and st.

The first component f is a mapping from the set aE of registers that are modified by a in τ into the set X of local states of τ. Intuitively, $f(r)$, for $r \in aE$, gives the last value written into r by a. If a was not yet performed then $f(r)$ contains any initial value of r. However to simulate the behaviour of τ the information provided by f is insufficient.

Suppose for example that for some $r \in R$, $Er = \{a_{i_0}, \ldots, a_{i_{k-1}}\}$ and $b \in rE$ (cf. Figure 7.19).

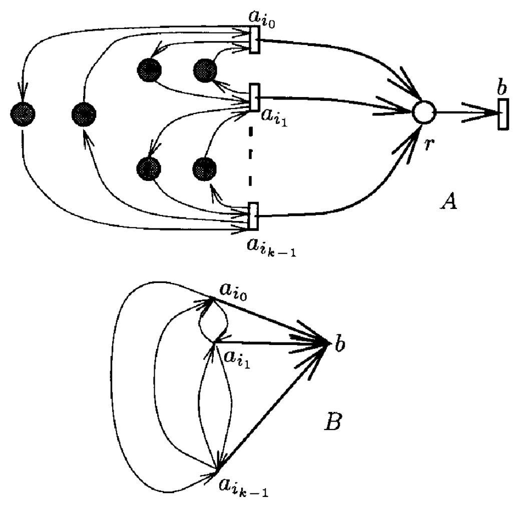


Figure 7.19. (A) In τ, $Er = \{a_{i_0}, \ldots, a_{i_{k-1}}\}$ and $b \in rE$. If σ is w-redundant then $\forall a_{i_m}, a_{i_l} \in Er,\ m \neq l \Longrightarrow a_{i_m}E \cap Ea_{i_l} \neq \emptyset$. (B) In τ', $\{a_{i_0}, \ldots, a_{i_{k-1}}\} \times \{b\} \subseteq C$. If σ is w-redundant then $\forall a_{i_m}, a_{i_l} \in Er,\ (m \neq l)\ (a_{i_m}, a_{i_l}) \in C$.

Thus $(a_{i_m}, b) \in C$ for all $a_{i_m} \in Er$ and, since the writing conflicts are redundant in τ, we have $(a_{i_m}, a_{i_l}) \in C$ for all $a_{i_m}, a_{i_l} \in Er$, $a_{i_m} \neq a_{i_l}$, i.e. Er is a clique of C. Now suppose that b is executed in τ. Then b reads the contents of r and this value, together with the values of the other registers read by b, is used to determine the transition performed by b.

Let us consider the execution of b in τ': b reads the states of the cells $a_{i_0}, \ldots a_{i_{k-1}}$ obtaining mappings $f_{i_0}, \ldots f_{i_{k-1}}$ respectively and $f_{i_l}(r)$, $0 \leq l < k$, give the last value written into r by a_{i_l}. However to simulate τ, b should know the actual value of r in τ, i.e. it should be able to determine the action of Er that was performed as the last.

To this end the second component of each state of τ' is used — it is called time-stamp. Intuitively, looking solely at the time-stamps of the cells $a_{i_0}, \ldots, a_{i_{k-1}}$ the agent b will be able to determine which of them was performed as the last.

For each register $r \in R$, $TS_r = \{0, \ldots, k-1\}$, where $k = \mathrm{card}(Er)$, will stand for the set of time-stamps associated with r. For each action $a \in \Sigma$ we need a time-stamp for every register $r \in aE$, thus the time-stamps for a are elements of the direct product $T_a = \prod_{r \in aE} TS_r$. Now we can specify formally the set X'_a of local states of $a \in \Sigma$ in τ':

$$X'_a = \mathbf{F}(aE; X) \times T_a$$

Thus $X' = \cup_{a \in \Sigma} X'_a$ is the set of all local states of τ'.

A partial state $u \in \mathbf{F}(\alpha; X')$, $\alpha \subseteq \Sigma$, is said to be valid if $\forall a \in \alpha$, $u(a) \in X'_a$. To explain how time-stamps are used in τ' we fix a linear order $\prec$ over the set Σ of actions.

If $u \in \mathbf{F}(\alpha; X')$ is a valid partial state of τ' then for each register $r \in R$ such that $Er \subseteq \alpha$ we can determine the last action that wrote into r by means of the following algorithm.

Let

$$Er = \{a_{i_0} \prec \ldots \prec a_{i_{k-1}}\} \tag{7.7}$$

i.e. we order the elements of Er according to $\prec$. To determine the value of r we use the local states of the cells of Er.

Let

$$u(a_{i_0}) = (f_{i_0}, ts_{i_0}), \ldots, u(a_{i_{k-1}}) = (f_{i_{k-1}}, ts_{i_{k-1}})$$

and

$$p = \sum_{l=0}^{k-1} ts_{i_l}(r) \bmod k$$

Then the action a_{i_p}, i.e. the $(p+1)$st action on the ordered list (7.7) is considered as the last writing into r. This action will be denoted by $last_u(r)$. Accordingly, $f_{i_p}(r) \in X$ will be taken as the state of r determined by u, this value will be denoted by $value_u(r)$.

Let us note that formally the algorithm given above defines for each valid state $u \in \mathbf{F}(\alpha; X')$ of τ' two mappings

$$last_u : R_u \longrightarrow \Sigma$$

and

$$value_u : R_u \longrightarrow X$$

where $R_u = \{r \in R \mid Er \subseteq \alpha\}$ and such that $\forall r \in R_u$, $last_u(r) \in Er$.

Moreover to compute $last_u(r)$ and $value_u(r)$ we use in fact only the local states $u(a_{i_0}), \ldots, u(a_{i_{k-1}})$, i.e. the partial state $u_{|Er}$.

Now we can define the covering mapping c, its domain is the set

$$U_{valid} = \{u \in \mathbf{F}(\Sigma; X') \mid \forall a \in \Sigma, u(a) \in X'_a\}$$

of valid global states and for $u \in U_{valid}$ we set

$$c(u) = value_u.$$

Now we are able to describe in detail how an execution of an action $a \in \Sigma$ in τ is simulated in τ'.

Let $s_1, s_2 \in S$ be two global states of τ such that $(s_1, a, s_2) \in \Delta$. Let us suppose that s_2 is obtained from s_1 by means of a transition $(s', s'') \in \delta_a$, i.e.

$$s_{1|Ea} = s', \quad s_{2|aE} = s'' \quad \text{and} \quad s_{1|R\setminus(aE)} = s_{2|R\setminus(aE)}.$$

Let $u_1 \in U_{valid}$ be a valid global state of τ' such that

$$c(u_1) = value_{u_1} = s_1, \tag{7.8}$$

i.e. u_1 covers s_1.

Simulating the transition (s_1, a, s_2) of τ in τ' we should obtain a valid global state $u_2 \in U_{valid}$ of τ' such that

(A) $c(u_2) = value_{u_2} = s_2$ (u_2 covers s_2),

(B) $\forall b \in \Sigma \setminus \{a\}$, $u_1(b) = u_2(b)$ (execution of a changes only the local state of a in τ'),

(C) the new local state $u_2(a)$ of a in τ' depends only on the partial state $u_{1|Ca}$ (a cannot access cells that are outside of Ca).

Let $u' = u_{1|Ca}$. First of all let us note that for each $r \in Ea, Er \subseteq Ca$, i.e. r is in the domain of $value_{u'}$ and since u_1 covers s_1 we have

$$\forall r \in Ea, \; value_{u'}(r) = value_{u_1}(r) = s_1(r) = s'(r)$$

Thus, intuitively, a can reconstruct s' by means of the partial state u'. Now note that s'' gives the new values of all registers in aE that are written during the execution of a, which implies that s'' is the first component of the new state $u_2(a)$ of a in τ'.

The new time-stamp ts'' of a should be chosen in such a way that immediately after the execution of a this action is indicated as the last action writing into each register $r \in aE$. We shall show how to calculate $ts''(r)$ for $r \in aE$ by means of u'. First note that the signature σ is w-redundant, i.e.

$$(Er \times Er) \setminus id_\Sigma \subseteq C \cap C^{-1}$$

which implies that in particular

$$Er \setminus \{a\} \subseteq Ca \tag{7.9}$$

(see Figure 7.19). Let $Er = \{a_{i_0}, \ldots, a_{i_{k-1}}\}$, where $a_{i_0} \prec \ldots \prec a_{i_{k-1}}$. Since $a \in Er$, there exists m, $0 \leq m < k$, such that $a = a_{i_m}$. From 7.9 it follows that $Er \setminus \{a\}$ is included in the domain of u'. Let

$$(f_{i_l}, ts_{i_l}) = u'(a_{i_l}) \quad \text{for} \quad l \neq m,\ 0 \leq l < k$$

and

$$p_r = \sum_{l=0, l\neq m}^{k-1} ts_{i_l}(r)$$

There exists a unique integer w_r, $0 \leq w_r < k$, such that

$$(p_r + w_r) \bmod k = m$$

and we set

$$ts''(r) = w_r$$

to obtain the time-stamp with required property. In this way we have translated a transition $(s', s'') \in \delta_a$ of τ into a transition $(u', u'') \in \delta'_a$ of τ', where $u'' = (s'', ts'')$. □

Note that the covering constructed in Proposition 7.5.4 is faithful, the conflict relation in the simulated fat-system τ is equal $D = C \cup C^{-1} \cup W$, where W is the writing conflict relation induced by σ, while the conflict relation in the constructed fact-system τ' is equal $D' = C \cup C^{-1} \cup id_\Sigma$. Since $id_\Sigma \subseteq W$ we see that the actions independent in τ remain independent in τ'. The time-stamp system used in the proof of Proposition 7.5.4 is adapted from [178].

Proposition 7.5.5 *Let $\tau = (\Sigma, C, X, \Delta)$ be an fact-system and let $\sigma = (\Sigma, R, E)$ be a w-redundant signature inducing the same causality relation C. Then there exists a fat-system $\tau' = (\Sigma, R, E, X', \Delta')$ over σ covering τ.*

Proof: The main idea of the simulation is the following: at each moment every register $r \in R$ of the constructed fat-system τ' will contain a partial state of the simulated fact-system τ, more precisely it will contain the values of the cells from the set Er in τ.

For each register $r \in R$ of τ' the local states of r are mappings from Er into X, by $X'_r = \mathbf{F}(Er; X)$ we denote the set of all such mappings and $X' = \bigcup_{r\in R} X'_r$ is the set of local states of τ'.

Let $\alpha \subseteq R$ and let $u \in \mathbf{F}(\alpha; X')$ be a partial state of τ'. The state u is said to be consistent if

1. $\forall r \in \alpha,\ u(r) \in X'_r$ and
2. $\bigcup_{r\in\alpha} u(r)$ is a mapping from $\bigcup_{r\in\alpha} Er \subseteq \Sigma$ into X.

To explain the last condition let us note that for $r \in \alpha$, $u(r) \in \mathbf{F}(Er; X)$ and the union of these mappings is again a mapping if they all agree on common parts of their domains, i.e. if $u(r')(a) = u(r'')(a)$ for all $r', r'' \in \alpha$ and $a \in Er' \cap Er''$. This

mapping will be denoted by $\overline{u}$, i.e. $\overline{u}(a) = u(r)(a)$, where $r \in \alpha$ and $a \in Er$. Note that if u is a consistent global state of τ' then $\overline{u}$ is a mapping from $\bigcup_{r\in R} Er = \Sigma$ into X, i.e. $\overline{u}$ is a global state of τ.

On the other hand, if $s \in \mathbf{F}(\Sigma; X)$ is a global state of τ then it determines in a natural way the corresponding global state $u \in \mathbf{F}(R; X')$:

$$\text{for all } r \in R,\ u(r) = s_{|Er}.$$

This global state u is consistent and moreover $\overline{u} = \bigcup_{r\in R} u(r) = s$.

Let U be the set of consistent global states of τ'. As we have seen the mapping $c : u \longmapsto \overline{u}$, $u \in U$, is a bijection between the set U of global consistent states of τ' and the set S of global states of τ. The transition relations of τ' will be constructed in such a way that this bijection will constitute the required covering of τ by τ'.

First we shall prove the following facts.

Let $u_1 \in \mathbf{F}(Ea; X')$ be a consistent partial state of τ'. Then

$$\begin{aligned} &\overline{u_1} \in \mathbf{F}(Ca; X) \text{ and} \\ &\forall r \in aE,\ Er \setminus \{a\} \subseteq Ca \end{aligned} \tag{7.10}$$

To prove the first assertion let us note that by the definition of the direct causality relation

$$\bigcup_{r\in Ea} \mathrm{dom}(u_1(r)) = \bigcup_{r\in Ea} Er = (E \circ E)a = Ca$$

($\mathrm{dom}(f)$ denotes here the domain of a partial function f).

Now suppose that $r \in aE$ and $b \in Er \setminus \{a\}$. Thus a and b are in writing conflict and, since σ is w-redundant, $(a, b) \in C \cap C^{-1}$, in particular $b \in Ca$. This proves second assertion.

Now we define transition relations of τ'.

Let δ'_a, $a \in \Sigma$, be a transition relation of τ' and let $u_1 \in \mathbf{F}(Ea; X')$, $u_2 \in \mathbf{F}(aE; X')$. Then $(u_1, u_2) \in \delta'_a$ if

1. u_1 is a consistent partial state of τ' and
2. there exists $x \in X$ such that $(\overline{u_1}, x) \in \delta_a$ and $\forall r \in aE,\ \forall b \in Er,$

$$u_2(r)(b) = \begin{cases} x & \text{if } b = a \\ \overline{u_1}(b) & \text{if } b \neq a \end{cases}$$

By 7.10 this definition is sound and the state u_2 is obviously consistent.

Now it suffices to observe two facts (their easy verification is left to the reader)

1. if $s_1, s_2 \in S$ are global states of τ and $(s_1, a, s_2) \in \Delta$ then $(u_1, a, u_2) \in \Delta'$, where $u_1, u_2 \in U$ are such that $\overline{u_i} = s_i$, $i = 1, 2$,
2. if $u_1 \in U$ and $(u_1, a, u_2) \in \Delta'$ then $u_2 \in U$ and $(\overline{u_1}, a, \overline{u_2}) \in \Delta$.

Since $U \ni u \longmapsto \overline{u}$ is a bijection, these condition imply that this mapping is a covering of τ.

□

7.6 Language Recognizability by Asynchronous Automata

In this section we enrich the structure of asynchronous transition systems by adding to them initial and final states and we examine the recognizability power of the automata obtained in this way.

A *finite asynchronous automaton* (faa in short) is a tuple $\mathcal{A} = (\Sigma, R, E, X, \Delta, I, F)$, where $\tau = (\Sigma, R, E, X, \Delta)$ is a fat-system and $I, F \subset \mathbf{F}(R; X)$ are respectively the sets of initial and final states. The language $L(\mathcal{A})$ recognized by $\mathcal{A}$ is defined in the standard way:

$$L(\mathcal{A}) = \{x \in \Sigma^* \mid \exists s_0 \in I, \Delta(s_0, x) \cap F \neq \emptyset\}$$

Finite asynchronous cellular automata (faca) are defined in the analogous way.

A finite asynchronous (cellular) automaton is *deterministic* if it has at most one initial state, $\mathrm{card}(I) \leq 1$, and its global transition relation is a partial mapping, i.e. for each global state s and $a \in \Sigma$, $\mathrm{card}(\Delta(s, a)) \leq 1$.

The main problem considered here is which classes of languages are recognized by finite asynchronous (cellular) automata over a fixed signature.

Let σ be a signature ($\sigma = (\Sigma, R, E)$ for faa or $\sigma = (\Sigma, C)$ for faca). By $\mathcal{L}^d_\sigma$ ($\mathcal{L}^n_\sigma$) we denote the class of languages recognized by deterministic (respectively non-deterministic) finite asynchronous (cellular) automata over signature σ.

Lemma 7.6.1 *Let τ_1 and τ_2 be two fat-systems. Let $\mathcal{A}_1 = (\tau_1, I_1, F_1)$ be a finite asynchronous automaton, $I_1, F_1 \subseteq S_1$. If τ_2 covers τ_1 then there exist sets $I_2, F_2 \subseteq S_2$ such that* $\mathrm{card}(I_2) = \mathrm{card}(I_1)$ *and $L(\mathcal{A}_1) = L(\mathcal{A}_2)$, where $\mathcal{A}_2 = (\tau_2, I_2, F_2)$ and S_i is the set of global states of τ_i.*

Proof: Let $c : U \longrightarrow S_1$ be a covering of τ_1 by τ_2. First we set $F_2 = c^{-1}(F_1)$. To obtain the set I_2 of initial states of $\mathcal{A}_2$ we choose exactly one global state from each set $c^{-1}(s)$, $s \in I_1$. (Thus I_2 is any subset of S_2 such that $I_2 \subseteq c^{-1}(I_1)$ and $\forall s \in I_1, \mathrm{card}(I_2 \cap c^{-1}(s)) = 1$.)

From the definition of covering and from the fact that $\mathrm{card}(I_1) = \mathrm{card}(I_2)$ it results that if $\mathcal{A}_1$ is deterministic then $\mathcal{A}_2$ is also. Directly from Proposition 7.5.2 we deduce that $L(\mathcal{A}_1) = L(\mathcal{A}_2)$. □

Proposition 7.6.2 *Let $\sigma_1 = (\Sigma, R, E)$ be a w-redundant signature and let $\sigma_2 = (\Sigma, C)$ be a signature of a fat-system such that $C = (E \circ E) \cap \Sigma^2$ is the direct causality relation induced by σ_1. Then*

$$\mathcal{L}^d_{\sigma_1} = \mathcal{L}^d_{\sigma_2} \text{ and } \mathcal{L}^n_{\sigma_1} = \mathcal{L}^n_{\sigma_1}$$

Proof: Directly from Proposition 7.5.4, Proposition 7.5.5 and Lemma 7.6.1. □

Note that if σ_1 and σ_2 are as in Proposition 7.6.2 then the conflict relation induced by σ_2 is included in the conflict relation for σ_1. Thus any language recognized by a w-redundant faa can be recognized by a faca without any loss of concurrency. For this reason in the sequel we restrain our attention to languages recognized by finite asynchronous cellular automata.

First we shall prove that without loss of generality we can assume in the sequel that there is always only one initial state.

Lemma 7.6.3 *For each faca $\mathcal{A} = (\Sigma, C, X, \Delta, I, F)$ there exists a faca $\mathcal{A}'$ over the same signature $\sigma = (\Sigma, C)$ and with only one initial state recognizing the same language.*

Proof: Let $I = \{s^1_o, \ldots, s^k_0\}$. The local states of the constructed automaton $\mathcal{A}'$ are k-tuples of the local states of $\mathcal{A}$, $X' = X^k$. For each i, $1 \leq i \leq k$, using the i-th component of the local states of $\mathcal{A}'$ we can easily simulate the behaviour of $\mathcal{A}$ with the initial state s^i_0. The details are left to the reader.

□

Lemma 7.6.4 *Let $\sigma_1 = (\Sigma_1, C_1)$ and $\sigma_2 = (\Sigma_2, C_2)$ be two signatures such that $\Sigma_1 \subseteq \Sigma_2$ and $C_1 = C_2 \cap (\Sigma_1 \times \Sigma_1)$. Then for each language $L \subseteq \Sigma_1^*$*

$$L \in \mathcal{L}^u_{\sigma_1} \textit{ iff } L \in \mathcal{L}^u_{\sigma_2}$$

where u means either d or n.

Proof: Let us note that for each $a \in \Sigma_1$, $C_2a = C_1a \cup (C_2a \setminus C_1a)$, i.e. the reading domain of a in σ_2 is the disjoint union of the reading domain C_1a of a in σ_1 and the subset $C_2a \setminus C_1a$ of $\Sigma_2 \setminus \Sigma_1$. This property is crucial in the construction.

Let $\mathcal{A}_1 = (\Sigma_1, C_1, X_1, \Delta_1, s^1_0, F_1)$ be a fca recognizing L. The idea is to build a faca $\mathcal{A}_2 = (\Sigma_2, C_2, X_2, \Delta_2, s^2_0, F_2)$ that works exactly as $\mathcal{A}_1$ for the words of Σ_1^* by ignoring the local states of the cells of $\Sigma_2 \setminus \Sigma_1$. As the set of local states of $\mathcal{A}_2$ we take $X_2 = X_1 \cup \{y_1, y_2\}$, where y_1, y_2 are new values not belonging to X_1. The initial state s^2_0 is such that $s^2_{0|\Sigma_1} = s^1_0$ and $s^2_0(a) = y_1$ for all $a \in \Sigma_2 \setminus \Sigma_1$. The execution of any action $a \in \Sigma_2 \setminus \Sigma_1$ writes the value y_2 into the cell a. For $a \in \Sigma_1$ and $s \in \mathbf{F}(C_2a; X_2)$, $(s, x) \in \delta^2_a \in \Delta_2$ if $(s_{|C_1a}, x) \in \delta^1_a$, i.e. a ignores the local states of $C_2a \setminus C_1a$ and is executed exactly as in $\mathcal{A}_1$. Now it suffices to take as final states of $\mathcal{A}_2$ the states $s \in \mathbf{F}(\Sigma_2; X_2)$ such that $s_{|\Sigma_1} \in F_1$ and $s(a) = y_1$ for all $a \in \Sigma_2$.

Suppose now that $\mathcal{A}_2 = (\Sigma_2, C_2, X_2, \Delta_2, s^2_0, F_2)$ is a faca recognizing $L \subseteq \Sigma_1^*$. We construct a faca $\mathcal{A}_1 = (\Sigma_1, C_1, X_1, \Delta_1, s^1_0, F_1)$ that simulates $\mathcal{A}_2$ for all computations generated by the words of Σ_1^* and starting at s^2_0. Note that since $L(\mathcal{A}_2) \subseteq \Sigma_1^*$, $s \in \Delta(s^2_0, u) \cap F_2$ implies that $s_{|\Sigma_2\setminus\Sigma_1} = s^2_{0|\Sigma_2\setminus\Sigma_1}$ (otherwise some action of $\Sigma_2 \setminus \Sigma_1$ should have been executed in u.)

Thus it suffices to take: $X_1 = X_2$, $s^1_0 = s^2_{0|\Sigma_1}$, $F_1 = \{s \in \mathbf{F}(\Sigma_1; X_1) \mid \exists s' \in F_2, s'_{|\Sigma_1} = s \text{ and } s'_{|\Sigma_2\setminus\Sigma_1} = s^2_{0|\Sigma_2\setminus\Sigma_1}\}$ and finally, for $a \in \Sigma_1$, $s \in \mathbf{F}(C_1a; X_1)$, $x \in X_1$, $(s, x) \in \delta^1_a \in \Delta_1$ if there exists $(s', x) \in \delta^2_a \in \Delta_2$ such that $s'_{|C_1a} = s$ and

$s'_{|C_2a\setminus C_1a} = s^2_{0|C_2a\setminus C_1a}$. It is clear that $\forall u \in \Sigma_1^*$, $(s_0^1, u, s) \in \Delta_1$ iff $(s_0^2, u, s') \in \Delta_2$, where $s'_{|\Sigma_1} = s$ and $s'_{|\Sigma_2\setminus\Sigma_1} = s^2_{0|\Sigma_2\setminus\Sigma_1}$.

□

Lemma 7.6.5 *Let $C_1, C_2 \subseteq \Sigma^2$ and $C_1 \subseteq C_2$, $\sigma_i = (\Sigma, C_i)$, $i = 1, 2$. Then $\mathcal{L}^d_{\sigma_1} \subseteq \mathcal{L}^d_{\sigma_2}$ and $\mathcal{L}^n_{\sigma_1} \subseteq \mathcal{L}^n_{\sigma_2}$.*

Proof: Let $\mathcal{A}_1$ be a faca over σ_1. To obtain a faca $\mathcal{A}_2$ over σ_2 recognizing the same language it suffices to modify slightly the transition relations. Intuitively, although the automaton $\mathcal{A}_2$ executing an action a reads the states of all cells of C_2a it will use effectively only the states of the cells of C_1a. Formally, for $a \in \Sigma$, $s \in \mathbf{F}(C_2a; X)$, $x \in X$, $(s, x) \in \delta^2_a \in \Delta_2$ iff $(s_{|C_1a}, x) \in \delta^1_a \in \Delta_1$. Note that if $\mathcal{A}_1$ is deterministic then $\mathcal{A}_2$ is deterministic as well. □

In the sequel we shall use the notion of computation paths in a faca $\mathcal{A} = (\Sigma, C, X, \Delta, s_0, F)$. Such a path is a sequence

$$s_{i_0} \overset{a_1}{\Longrightarrow} s_{i_1} \overset{a_2}{\Longrightarrow} \ldots \overset{a_n}{\Longrightarrow} s_{i_n}$$

where s_{i_k} are global states, a_k actions and $\forall k$, $(1 \le k \le n)$ $s_{i_k} \in \Delta(s_{i_{k-1}}, a_k)$. This computation path is *initial* if $s_{i_0} = s_0$, and is *accepting* if it is initial and s_{i_n} is a final state.

Lemma 7.6.6 *Let $\Sigma = \{a\}$, $C = \emptyset$, $\sigma = (\Sigma, C)$. Then $\mathcal{L}^d_\sigma = \mathcal{L}^n_\sigma = \{\emptyset, \mathbf{1}, a^+, a^*\}$.*

Proof: First we show that all four languages $\emptyset, \mathbf{1}, a^+, a^*$ belong to $\mathcal{L}^d_\sigma$. Taking $\delta_a = \{(\emptyset, (a, 1))\}$, $s_0 = \{(a, 0)\}$, $x = \{0, 1\}$ we get a faca recognizing either $\mathbf{1}$ if $F = \{\{(a, 0)\}\}$ or a^+ if $F = \{\{(a, 1)\}\}$. The trivial automata recognizing a^* and $\emptyset$ are left to the reader.

Let $L \in \mathcal{L}^d_\sigma$. To prove that L is one of the four languages given in the thesis it suffices to show that $a^i \in L$, $i > 0$, implies $a^+ \subseteq L$.

Let $\mathcal{A}$ be a faca over σ recognizing a word a^i. We consider an accepting computation path for a^i:

$$s_0 \overset{a}{\Longrightarrow} s_1 \overset{a}{\Longrightarrow} \ldots \overset{a}{\Longrightarrow} s_{i-1} \overset{a}{\Longrightarrow} s_i$$

where s_0 is the initial and s_i a final state. Since $C = \emptyset$ the last transition used in this computation is always enabled and leads directly to the final state. Thus for each $n > 0$, using n times this transition we obtain the computation path

$$s_0 \underbrace{\overset{a}{\Longrightarrow} s_i \overset{a}{\Longrightarrow} s_i \overset{a}{\Longrightarrow} \ldots \overset{a}{\Longrightarrow} s_i}_{n \text{ times}}$$

accepting a^n. Thus $\mathcal{L}^n_\sigma \subseteq \{\emptyset, \mathbf{1}, a^+, a^*\} \subseteq \mathcal{L}^d_\sigma$, and since $\mathcal{L}^d_\sigma \subseteq \mathcal{L}^n_\sigma$ trivially we get the thesis. □

Lemma 7.6.7 *Let $\Sigma = \{a\}$, $C = \{(a, a)\}$, $\sigma = (\Sigma, C)$. Then $\mathcal{L}^d_\sigma = \mathcal{L}^n_\sigma = \mathrm{Rec}_\Sigma$.*

Proof: Let us note that this is just a special case of the general Theorem 7.6.11. Intuitively, if $\mathcal{A}$ is a faca over σ then we get immediately an equivalent finite automaton $\mathcal{B}$ by identifying the state of the unique cell of $\mathcal{A}$ with the state of $\mathcal{B}$. This yields the thesis since the deterministic and non-deterministic finite automata recognize the class Rec_Σ of languages. □

Lemma 7.6.8 *Let $\Sigma = \{a, b\}$ $(a \neq b)$, $C_1 = \{(b, a)\}$, $C_2 = \{(a, a), (a, b), (b, b)\}$, $\sigma_i = (\Sigma, C_i)$ $(i = 1, 2)$.*
Let $L = a^+b^+$, $L' = b^+a(a \cup b)^*b$.
Then $L \in \mathcal{L}^d_{\sigma_1}$, $L \notin \mathcal{L}^d_{\sigma_2}$, $L \in \mathcal{L}^n_{\sigma_2}$, $L' \in \mathcal{L}^n_{\sigma_1}$, $L' \notin \mathcal{L}^d_{\sigma_1}$.

Proof: To simplify the notation we shall present the global states $s \in \mathbf{F}(\{a, b\}; X)$ as pairs $(s(a), s(b))$ of values of X.

To show that $L \in \mathcal{L}^d_{\sigma_1}$ we set $X = \{0, 1\}$, $s_0 = (0, 0)$, $F = \{(1, 1)\}$ and

δ_a

b	a
0	1

δ_b

b
1

Figure 7.20 presents the transition diagram of the faca defined above.

$$\begin{tikzcd}
{} \arrow[r] & (0,0) \arrow[r, "a"] \arrow[d, "b"] & (1,0) \arrow[loop above, "a"] \arrow[r, "b"] & (1,1) \arrow[loop above, "b"] \arrow[r] & {} \\
& (0,1) \arrow[loop below, "b"] & & &
\end{tikzcd}$$

Figure 7.20: A deterministic finite asynchronous cellular automaton recognizing L

To show that $L \in \mathcal{L}^n_{\sigma_2}$ we take the following faca: $X = \{0, 1\}$, $s_0 = (0, 0)$, $F = \{(1, 1)\}$,

δ_a

a	a
0	0
0	1

δ_b

a	b	b
1	0	1
1	1	1

The transition diagram of this automaton is given on Figure 7.21.

Finally we shall prove that $L \notin \mathcal{L}^d_{\sigma_2}$.

$$\begin{tikzcd}
{} \arrow[r] & (0,0) \arrow[loop above, "a"] \arrow[r, "a"] & (1,0) \arrow[r, "b"] & (1,1) \arrow[loop above, "b"] \arrow[r] & {}
\end{tikzcd}$$

Figure 7.21. A non-deterministic finite asynchronous cellular automaton recognizing L. Note that only the global states reachable from the initial state are presented on faca diagrams.

Suppose that $\mathcal{A}_2 = (\Sigma, C_2, X, \Delta, s_0, F)$ is a deterministic faca recognizing L and let $n = \mathrm{card}(X)$. Let us examine the accepting computation path for the word $a^n b \in L$:

$$s_0 \overset{a}{\Longrightarrow} s_1 \overset{a}{\Longrightarrow} \dots \overset{a}{\Longrightarrow} s_n \overset{b}{\Longrightarrow} s_{n+1} \in F \tag{7.11}$$

Let $s_0 = (x_0, y_0)$. Since a can change only its local state we have $s_i = (x_i, y_0)$, $0 \le i \le n$, for some $x_i \in X$. Similarly as b can modify only its local state we have $s_{n+1} = (x_n, y_1)$ for some $y_1 \in X$. Since $n = \mathrm{card}(X)$, some elements should occur more then once in the sequence

$$x_0, \dots, x_n \tag{7.12}$$

i.e. $\exists i, j$, $0 \le i < j \le n$, $x_i = x_j$ (which implies $s_i = s_j$.) Now we can deduce that for some l, $0 \le l < n$, $x_l = x_n$, i.e. the last element in (7.12) occurs at least twice. (Otherwise suppose that x_k is the last element occuring more than once in (7.12) and let $x_i = x_k$ for $0 \le i < k \le n$. If $k \ne n$ then $s_k = s_i \overset{a}{\Longrightarrow} s_{i+1}$ and $s_k \overset{a}{\Longrightarrow} s_{k+1}$ and as $\mathcal{A}_2$ is deterministic, we get $s_{i+1} = s_{k+1}$ and $x_{i+1} = x_{k+1}$, in contradiction with our choice of k.)

Now we take the subpath

$$s_n = s_l = (x_l, y_0) \overset{a}{\Longrightarrow} s_{l+1} = (x_{l+1}, y_0) \overset{a}{\Longrightarrow} \dots \overset{a}{\Longrightarrow} s_n = (x_n, y_0) \tag{7.13}$$

of the computation path (7.11). Since $C_2 a = \{a\}$ the executions of a do not depend of the local state of b, therefore replacing y_0 by y_1 in each global state of (7.13) we obtain again a valid computation path:

$$s_{n+1} = (x_n, y_1) = (x_l, y_1) \overset{a}{\Longrightarrow} (x_{l+1}, y_1) \overset{a}{\Longrightarrow} \dots \overset{a}{\Longrightarrow} (x_n, y_1) = s_{n+1}$$

which shows that $s_{n+1} = \Delta(s_{n+1}, a^{n-l})$. Therefore $s_{n+1} = \Delta(\Delta(s_0, a^n b), a^{n-l}) = \Delta(s_0, a^n b a^{n-l}) \cap F$, i.e. $a^n b a^{n-l} \in L(\mathcal{A}_2)$ and $\mathcal{A}_2$ recognizes a word not belonging to L.

To show that $L' \in \mathcal{L}^n_{\sigma_1}$ we set:

$X = \{0, 1, 2\}$, $s_0 = (0, 0)$, $F = \{(1, 2)\}$ and

δ_a

b	a
1	1

δ_b

b
1
2

Figure 7.22 presents the transition graph for this faca.

$$\begin{tikzcd}
{} \arrow[r] & (0,0) \arrow[r, "b"] \arrow[d, "b"'] & (0,1) \arrow[loop above, "b"] \arrow[r, "a"] \arrow[dl, bend left, "b"] & (1,1) \arrow[loop above, "{a,b}"] \arrow[d, bend right, "b"'] & \\
& (0,2) \arrow[loop below, "b"] \arrow[ur, bend left, "b"] & & (1,2) \arrow[loop below, "b"] \arrow[u, bend right, "b"'] \arrow[r] & {}
\end{tikzcd}$$

Figure 7.22. A non-deterministic finite asynchronous cellular automaton recognizing L'

To prove that $L' \notin \mathcal{L}^d_{\sigma_1}$ let us take a deterministic faca $\mathcal{A}_1 = (\Sigma, C_1, X, \Delta, s_0, F)$ over σ_1. Since $\mathcal{A}_1$ is deterministic and the reading domain of b is empty, δ_b is either empty or consists of just one transition: $\delta_b = \{(\emptyset, \{(b, y_1)\})\}$ for some $y_1 \in X$. In the first case $\mathcal{A}_1$ can never execute b and $L(\mathcal{A}_1) \ne L'$. In the second case b is always enabled and the first execution of b writes y_1 into the cell b and all subsequent executions of b do not change the local state of b. Thus executing ba or bab^n, $n > 0$, at the initial state we arrive at the same global state, i.e. $ba \in L(\mathcal{A}_1)$ iff $bab \in L(\mathcal{A}_1)$ and $L' \ne L(\mathcal{A}_1)$.

□

Theorem 7.6.9 (Hierarchy Theorem) *Let $C_1, C_2 \subseteq \Sigma^2$, $\sigma_i = (\Sigma, C_i)$, $i = 1, 2$. Then $\mathcal{L}^d_{\sigma_1} \subseteq \mathcal{L}^d_{\sigma_2}$ iff $C_1 \subseteq C_2$.*

Proof: The right to left implication is given by Lemma 7.6.5.

Now suppose that C_1 is not included in C_2. There are two cases to examine.

CASE 1: There exists $a \in \Sigma$ such that $(a, a) \in C_1 \setminus C_2$.

Then by Lemma 7.6.6, Lemma 7.6.7 and Lemma 7.6.4 any recognizable subset of a^* different from $\emptyset, 1, a^+, a^*$ belongs to $\mathcal{L}^d_{\sigma_1}$ but not to $\mathcal{L}^d_{\sigma_2}$, for example $(aa)^* \in \mathcal{L}^d_{\sigma_1} \setminus \mathcal{L}^d_{\sigma_2}$.

CASE 2: There exist $a, b \in \Sigma$, $a \ne b$, such that $(b, a) \in C_1 \setminus C_2$.

Let $\Sigma_r = \{a, b\}$ and $C' = C_1 \cap (\Sigma_r \times \Sigma_r)$, $C'' = C_2 \cap (\Sigma_r \times \Sigma_r)$, $\sigma' = (\Sigma_r, C')$, $\sigma'' = (\Sigma_r, C'')$. Then $\{(b, a)\} \subseteq C'$ and $C'' \subseteq \{(a, a), (a, b), (b, b)\}$. By Lemma 7.6.5 and Lemma 7.6.8, $a^+ b^+ \in \mathcal{L}^d_{\sigma'} \setminus \mathcal{L}^d_{\sigma''}$ and applying Lemma 7.6.4 we see that $a^+ b^+ \in \mathcal{L}^d_{\sigma_1} \setminus \mathcal{L}^d_{\sigma_2}$.

□

Theorem 7.6.9 describes all possible inclusions between families $\mathcal{L}^d_{(\Sigma,C)}$ with C ranging over all binary relations over Σ. We see that $\mathcal{L}^d_{(\Sigma,\emptyset)}$ is the least while $\mathcal{L}^d_{(\Sigma,\Sigma^2)}$ the greatest family in this hierarchy. Moreover this hierarchy is proper: $\mathcal{L}^d_{(\Sigma,C_1)} = \mathcal{L}^d_{(\Sigma,C_2)}$ iff $C_1 = C_2$.

The precise characterization of all possible inclusions in the hierarchy of the classes $\mathcal{L}^n_{(\Sigma,C)}$ of languages recognized by non-deterministic faca is an open problem. The only fact known for this classes is the trivial inclusion provided by Lemma 7.6.5. We should note here that Lemma 7.6.8 gives two examples of signatures for which the inclusion $\mathcal{L}^d_\sigma \subset \mathcal{L}^n_\sigma$ is strict, which shows that deterministic and non-deterministic hierarchies are different.

The main open problem in the theory of asynchronous automata is to characterize the classes $\mathcal{L}^d_\sigma$ and $\mathcal{L}^n_\sigma$ for any signature σ. However, there is one important case where such a characterization is known.

Let $\simeq_C$ be the smallest congruence over Σ^* such that

- $ab \simeq_C ba$ if $(a,b) \notin C \cup C^{-1}$
- $aa \simeq_C a$ if $(a,a) \notin C$

Let

$$\mathrm{Rec}(\Sigma^*, C) = \{L \subseteq \Sigma^* \mid L \in \mathrm{Rec}_\Sigma \text{ and } \forall x,y \in \Sigma^*,\ x \simeq_C y \Longrightarrow (x \in L \Leftrightarrow y \in L)\}$$

Thus $\mathrm{Rec}(\Sigma^*, C)$ consists of all recognizable languages that are closed under $\simeq_C$.

The following inclusions are obvious.

Fact 7.6.10 *If $C \subseteq \Sigma^2$ and $\Sigma = (\Sigma, C)$ then*

$$\mathcal{L}^d_\sigma \subseteq \mathcal{L}^n_\sigma \subseteq \mathrm{Rec}(\Sigma^*, C)$$

Proof: It suffices to note that if $\mathcal{A}$ is a faca over σ then $ab \simeq_C ba$ and $aa \simeq_C a$ imply $\Delta(s, ab) = \Delta(s, ba)$ and $\Delta(s, aa) = \Delta(s, a)$ respectively. □

The main result of the theory of asynchronous automata (cf. [46, 277]) indicates that under some condition imposed on C the inclusions in Fact 7.6.10 can be replaced by equalities.

Theorem 7.6.11 (Main Theorem) *Let $\sigma = (\Sigma, C)$. If the relation $C \subseteq \Sigma^2$ is symmetric and reflexive ($C = C \cup C^{-1} \cup id_\Sigma$) then*

$$\mathcal{L}^n_\sigma = \mathcal{L}^d_\sigma = \mathrm{Rec}(\Sigma^*, C)$$

Let us note that if the relation C is symmetric and reflexive then C is equal with the conflict relation D and the family $\mathrm{Rec}(\Sigma^*, C)$ can be identified with the family of recognizable subsets of the free partially commutative monoid $\mathbb{M}(\Sigma, I)$, where $I = \Sigma^2 \setminus D$ is the independency relation induced by σ. Thus Theorem 7.6.11 states that if C is symmetric and reflexive then finite asynchronous cellular automata can recognize exactly all recognizable subsets of the corresponding free partially commutative monoid.

On the other hand if C is not symmetric or not reflexive, i.e. if C is strictly included in D then by the Hierarchy Theorem (Theorem 7.6.9) $\mathcal{L}^d_\sigma$ is also strictly included in the family of recognizable subsets of $\mathbb{M}(\Sigma, I)$.

We end this section with a discussion of the adequacy of cp-graph semantics for finite asynchronous automata, for the sake of simplicity of presentation we restrain our considerations to cellular automata. As we noted in Subsection 7.3.2 (Proposition 7.3.9) cp-graphs constitute actually a representation of traces. However there are other trace representations, for example by equivalence classes of words under $\sim_I$ relation or by dependence graphs (cf. Chapter 2, [1]) and the question can be raised why we still need this new trace model. Let $\sigma = (\Sigma, C)$ and $u \in \Sigma^*$. First note that each event of the cp-graph $\Psi_\sigma(u) = (Ev_u, Ca_u, Pr_u, \lambda_u)$ has uniformly bounded in-degree for Ca_u and Pr_u:

if $e \in Ev_u$ and $\lambda_u(e) = a$ then $\mathrm{card}(\{e' \in Ev_u \mid (e', e) \in Ca_u\}) \le \mathrm{card}(Ca)$ and $\mathrm{card}(\{e' \in Ev_u \mid (e', e) \in Pr_u\}) \le \mathrm{card}(Da)$, where D is the conflict relation induced by σ (cf. the end of Section 7.4 for the definition of cp-graphs adapted for cellular automata). This boundedness property enables to introduce directly finite automata over cp-graphs. For such an automaton $\mathcal{B}$ we should specify

- a finite set X of states,
- initial states that are associated with each initial event $e \in Ev_u$ (e is initial if $\{e' \in Ev_u \mid (e', e) \in Ca_u\} = \emptyset$, we can have several initial events with the same label, cf. Example 7.3.10, in this case they all have the same initial state); these states depend on the label of e but not of e itself,
- a transition mapping enabling to calculate the state of each non-initial event $e \in Ev_u$ if the following information is provided
  - the label $\lambda_u(e)$ of e and
  - the labels and the states of all the events e' such that $(e', e) \in Ca_u$,
- the set of final states, these states are partial mappings from Σ into X.

A run of the automaton $\mathcal{B}$ over cp-graph $\Psi_\sigma(u) = (Ev_u, Ca_u, Pr_u, \lambda_u)$ is a mapping r from Ev_u into X associating with each initial event its initial state and with the other events the state calculated by means of the transition mapping of $\mathcal{B}$. The run r is accepting if

$$\{(\lambda_u(e), r(e)) \mid e \in Ev_u \text{ is a final event}\} \tag{7.14}$$

is a final state (an event e is final if $\forall e' \in Ev_u, \lambda_u(e') = \lambda_u(e) \Longrightarrow (e', e) \in Pr_u^*$).

Note that Pr_u^* totally orders all events with the same label, i.e. (7.14) defines a partial mapping from Σ into X, the domain of this mapping is the set of letters that occur in u.

Given a faca $\mathcal{A} = (\Sigma, C, X, \Delta, s_0, F)$ it is trivial to construct a corresponding cp-graph automaton $\mathcal{B}$. The set of states of $\mathcal{B}$ is simply equal X. For $\alpha \subseteq \Sigma$,

$s : \alpha \longrightarrow X$ is a final state of $\mathcal{B}$ iff the mapping $s' : \Sigma \longrightarrow X$ such that $\forall a \in \alpha$, $s'(a) = s(a)$ and $\forall a \in \Sigma \setminus \alpha$, $s'(a) = s_0(a)$ is a final state of $\mathcal{A}$.

Let $u \in \Sigma^*$ and $\Psi_\sigma(u) = (Ev_u, Ca_u, Pr_u, \lambda_u)$. We shall describe how a run r of $\mathcal{B}$ over $\Psi_\sigma(u)$ simulates the execution of $\mathcal{A}$ on u (this will implicitly explain how the transition of $\mathcal{B}$ is constructed from the transition relations of $\mathcal{A}$).

The initial state $r(e)$ of each initial event $e \in Ev_u$ is set to be equal $s_0(\lambda_u(e))$. Let e be a non-initial event such that r is already defined for all events e' such that $(e', e) \in Ca_u$. Suppose that $\lambda_u(e) = a$. Then we set $r(e) = x \in X$ if there exists a transition $(s, x) \in \delta_a$ of $\mathcal{A}$ such that

- $\forall e'$ such that $(e', e) \in Ca_u$, $s(\lambda_u(e')) = r(e')$,
- $\forall b \in Ca \setminus \{\lambda_u(e') \mid (e', e) \in Ca_u\}$, $s(b) = s_0(b)$ (note that always $\{\lambda_u(e') \mid (e', e) \in Ca_u\} \subseteq Ca$).

In a similar way we can easily transform a cp-graph automata to finite asynchronous cellular automata.

Thus we see that asynchronous cellular automata can be interpreted, after some minor modifications, as automata over cp-graphs.

It should be noted that, actually only the direct causality relation Ca_u is really essential for cp-graph automata, the role of the direct precedence relation Pr_u is limited to indicate the last occurrence of each action — this information is needed in order to find the final state of a run. If the relation C is reflexive then we can get rid of Pr_u at all since in this case Ca_u^* orders totally all occurrences of the same action in Ev_u in the same way as Pr_u.

The fact that asynchronous automata can be viewed as finite automata over labelled acyclic graphs associated to traces was noted explicitly by Thomas[263]. However Thomas originally used reduced dependence graphs, which makes the transformations between asynchronous automata and graph automata a bit more complex. Let us note also that for non-reduced dependence graphs there is no natural way to define finite automata since the in-degree of vertices of these graphs is not bounded.

7.7 Bibliographical Remarks

The Main Theorem (Theorem 7.6.11) was proved in Zielonka[277]. Subsequently the proof was simplified and improved in Cori et al.[46]. The reader can find one of the direct general constructions of deterministic asynchronous (cellular) automata in Chapter 8. Unfortunately all known such constructions are quite involved and give rise to a considerable explosion of the number of states. We should note however that in some interesting cases it is possible to present simpler and more transparent constructions.

The first case is when we allow nondeterminism. Nondeterministic asynchronous automata for a given trace language were built by Pighizzini[228, 227]. Recently methods for converting nondeterministic asynchronous automata to deterministic ones were devised by Muscholl[198] and Klarlund et al.[154].

The construction of deterministic asynchronous automata becomes also simpler if we add some constraints on their topology, for example if the signature can be represented by an undirected tree Métivier[196], Perrin[223], Bertoni et al.[21]. Let us note that the last paper [21] can be considered as a precursor of the theory, although written in the context of labelled Petri nets, it actually provides a non-trivial special case of Theorem 7.6.11. Unfortunately, this paper had little impact on the further development of the theory since it did not appear in the proceedings and its circulation was very limited. An elegant extension of these methods to the case of triangulated dependency graphs was proposed by Diekert and Muscholl and is presented in Chapter 8.

Reductions between various types of asynchronous automata were studied by Cori et al.[48], Pighizzini[229, 227]. Section 7.5 generalizes these results.

The minimalization problem for asynchronous automata was considered by Bruschi et al.[31], and Pighizzini[229, 227]. The main result is that in general the category of deterministic asynchronous (cellular) automata recognizing a given trace language does not admit the unique minimal automaton.

Probabilistic asynchronous automata were examined by Jesi et al. [145] and Pighizzini [227].

Recently asynchronous automata were successfully used to recognize sets of infinite traces (Gastin et al.[113], Diekert et al.[62].)

Asynchronous cellular automata were introduced by Zielonka[278]. In this paper also a “safety” property of asynchronous automata is considered. (A safe automaton is an automaton such that each initial computation path can be extended to an accepting path.)

Theorem 7.6.9 (Hierarchy Theorem) is new. Also the general definitions of asynchronous (cellular) automata given in this chapter are new; previously only automata inducing symmetric and reflexive conflict relations were considered in the literature.

Let us note finally that although asynchronous automata were introduced explicitly in [277], actually similar models have much longer history and were used to solve problems in distributed computing. In this domain especially some topologies, for example with circular connection relation, give rise to appealing and non-trivial problems. For example the computational model used by Dijkstra [65] in his famous self-stabilization problem has immediate representation as an asynchronous automaton (intuitively the system considered in [65] consists of a ring of finite state automata each of them able to read the state of its two neighbours). As another interesting algorithmic problem solved by Mazurkiewicz in a similar model we can mention the ranking problem [191]. We should indicate that the problems such as the two ones mentioned above have quite different flavour than the problems considered in this chapter — the possible sequences of actions executed in the system are of no interest for self-stabilization or ranking problems, in both of them we want to construct automata such that ultimately all reachable global states verify some special property.