Decomposition of sequential machines into PLAM networks

Kolsteren, M.A.J.

Award date:
1997

Decomposition of Sequential Machines into PLAM Networks

graduation report
by M.A.J. Kolsteren

Supervision: dr.ir. L. Jóźwiak
Ordered by: prof.ir. M.P.J. Stevens

October 1994,
Eindhoven University of Technology,
Faculty of Electrical Engineering,
Section of Digital Information Systems.

Abstract

The growing complexity of sequential control and processing units of digital systems, and the increasing use of field programmable devices to implement them, has yielded a great demand for logic synthesis tools that are able to decompose large sequential machines into networks of limited building blocks. In the Section of Digital Information Systems of the Eindhoven University of Technology, a method has been developed that is able to decompose a given sequential machine into a network of Programmable Logic Arrays with Memory (PLAMs) with limited maximal size, such that the number of PLAMs and the number of connections in the network are minimized. This method solves the problem rather good, but is not suitable for sequential machines with large states (i.e. states with many outgoing transitions and many input bits and output bits occuring in these transitions) and not good for delay. The aim of the reported research was to develop a modified method which eliminates these drawbacks.

For realising the aim, we have derived a new *decomposition model*, that describes PLAM network decompositions in terms of task distributions. This new decomposition model solves the problems with large states, is good for delay, and yields not only sequential, but also simultaneous decompositions. The decomposition model describes a large subclass of all possible decompositions of a certain sequential machine.

We have also developed a new *decomposition method*. This method determines a good subtask set of the sequential machine, such that the subtasks in this set together cover all elementary tasks of the sequential machine. Afterwards, a good distribution of these subtasks is constructed. Finally, the task distribution is translated into a PLAM network.

A *formal framework* has been designed for presenting our new model and method in a precise way. The formal treatment not only facilitates final software implementation of the method, but also makes it possible to prove that the method yields correct decompositions under all circumstances.

Contents

1 Introduction

Logic synthesis is an important stage in the design process of digital systems. It consists of decomposing a functional module into a network of logic building blocks that realizes the behaviour of the functional module. The functional module can be seen as a black box, for which only the inputs, outputs and functionality are defined. The network of logic building blocks that implements the functional module must be suitable for direct implementation with a chosen technology, for example as a custom integrated circuit or a field programmable device.

Usually, the network of logic building blocks is constrained in many ways. These constraints are not only imposed by the specification of the functional module, but also by the technology which has to be used to implement the module. The specification may limit the area, delay and power dissipation of the network. The technology may demand that the logic building blocks are chosen from a limited set. This limited set can be a standard cell library when aiming at ASIC realization, or a set of programmable logic blocks of a certain family when aiming at realization with field programmable devices. The technology can also impose restrictions on the way of connecting the building blocks. For example, a connection restriction can consist of a limited fan-in and fan-out of each building block.

In addition to these 'hard' constraints on the network of logic building blocks, there may also be 'soft' constraints, i.e. optimization objectives. Example objectives are area minimization, delay minimization, power dissipation minimization, minimization of the number of building blocks and minimization of the number of connections between building blocks.

For solving logic synthesis problems with soft constraints, some specific methods are available. For problems with hard constraints, very few methods are available, which is not surprising because problems with hard constraints are more difficult to solve. The introduction of field programmable devices, which are important for rapid prototyping and application specific digital systems, has increased the need for logic synthesis methods that can solve problems with many hard constraints. In this report, a method is proposed that can cope with many hard and soft constraints simultaneously. The proposed method is restricted to solving one practical logic synthesis problem with many constraints, but the basic ideas behind the method can be used as a framework for designing methods that solve other problems with many constraints.

This report focusses on logic synthesis for control units, aiming at implementation with a network of Programmable Logic Arrays with Memory (PLAMs). The control unit is described as a sequential Mealy machine

with encoded inputs, symbolic states, encoded outputs and a possibly incompletely specified next state function. A restriction on the PLAM network is that all PLAMs in this network must have the same maximal dimensions, i.e. the same maximal number of input bits, state bits, output bits and term lines. Chosen maximal dimensions of the PLAMs in the network form hard constraints. In addition to these hard constraints, two soft constraints are introduced. The first soft constraint is that the number of PLAMs in the network should be as small as possible. The second soft constraint is that the number of connections in the network should be as small as possible. These soft constraints are not of equal importance. The second soft constraint must be fulfilled under the condition that the first one is fulfilled.

The problem of implementing a sequential machine with PLAMs has been previously considered by Levin. In his article [6], he proposes a method which makes it possible to translate a sequential machine into a PLAM network that realizes the behaviour of the sequential machine. This method is based on partitioning the symbolic states of the sequential machine, and constructing a PLAM for each block of this state partition, such that this PLAM computes all next state and output information for the states that are contained in the corresponding block. The communications between the PLAMs in the network, which are needed to exchange next state information, are based on a hierarchy. Levin only describes how a network of PLAMs can be derived from a given partition on the states of a sequential machine. The problem of multi-constraint optimization for PLAM networks has been stated more recently by Jóźwiak and Kolsteren [4]. They developed a complete method and automatic synthesis tool for solving the problem. For a given sequential machine, and given maximal PLAM dimensions, the method describes how to find a state partition such that the corresponding network satisfies all hard constraints and requires a minimum number of PLAMs and a minimum number of interconnections. This tool has proven to be efficient and effective for many practical examples. Another tool, which is also based on state partitioning, but uses direct communications between PLAMs instead of communications along a hierarchy tree, is presented in [1].

The method which is described in [4] was developed in the Section of Digital Information Systems of the Eindhoven University of Technology. The good results of this method motivated further research.

The aim of the reported graduation project was to modify the existing method, in order to make it suitable for large states and better for delay.

A state is called large if the number of outgoing transitions from this state and the number of inputs and outputs for all these transitions are close to, or even larger than the chosen PLAM dimensions. A known drawback of the existing method is that large states of the sequential machine can have great negative impact on the quality of the final PLAM network, and even on the existance of a decomposition of the described type. This problem follows from the fact that the decomposition is

based on state partitioning and all the transitions from a certain state, i.e. computations of the next state and output values defined by these transitions, must be completely implemented in the PLAM which implements the state. Thus, improvement of the method for large states is possible by allowing the various next-state and output computations for a certain present-state to be implemented in various PLAMs. This means replacing the existing decomposition model which is based on state partitioning by a new decomposition model based on a more elementary computation partitioning.

The existing method constructs a network which is based on a hierarchy. In a hierarchy, communication between PLAMs takes place along the edges of the hierarchy tree. The advantage of using a hierarchy is that the number of communication lines is relatively small when compared to the number of communication lines needed for direct communications between PLAMs and that the number of PLAMs' inputs used for communications can be much smaller. However, hierarchical communication causes long delays, especially when the hierarchy tree has many levels. The existing method can be made better for delay, by using direct communication between PLAMs.

In order to realize the aim of the project, it was necessary to analyze the problem, to develop a modified decomposition model, and to construct a modified decomposition method.

The first chapter of this report analyses the problem of decomposing a sequential machine into a PLAM network that realizes the behaviour of the sequential machine and satisfies the hard and soft constraints. In chapter two, a new decomposition model is proposed which is based on elementary computation task distribution. It is described how the task distribution can be made for a given sequential machine, and how this task distribution can be translated into a PLAM network that implements the sequential machine. Chapter three gives an algorithm that constructs a PLAM network that satisfies all hard constraints and soft constraints, based on the task distribution model of chapter two. The conclusions are given in chapter five.

The decomposition model and method are presented in a formal framework. The main reason for the frequent use of precise formal notation, is that it facilitates future software implementation. For the convenience of the readers, this report contains a complete list of all formal notations.

2 Problem Analysis

The analysis of the problem consists for a great part of deriving precise and consistent definitions for all notions on which the problem is based. The main notions are the ones that occur in the title of this report: sequential machines, PLAM networks and decomposition. The first three sections of this chapter work out precisely these three main notions. The fourth section gives a precise problem statement, based on the definitions that are introduced in the first three sections.

2.1 Sequential Machines

2.1.1 Terms over Sets of Bits

Let us begin with defining what we mean precisely when we talk about a term. A term is always defined over a set of bits. A term over a set of bits B is an assignment of the bits of B. Bits of B can be assigned the value zero (0), one (1) or don't care ($\times$ or $-$). The following definition gives a formal description of a term.

Definition 2.1 A *term* p over a set of bits B is a function $p : B \rightarrow \{0, 1, \times\}$. The set of all terms over B is denoted with $\mathcal{T}(B)$.

We define the 0-bits of a term as the bits that are assigned the value 0 by the term, and we define the 1-bits and $\times$-bits of a term in the same way. The 0-bits and the 1-bits are called the care bits, and the $\times$-bits are called the don't care bits.

Definition 2.2 Let t be a term over a set of bits B, and let a be an element of the set $\{0, 1, \times\}$. Then a bit $b \in B$ is an *a-bit* of a term t if $t(b) = a$. The set of all a-bits of term t is denoted with $B_a(t)$. A bit $b \in B$ is a *care bit* of t if b is a 0-bit or a 1-bit of t, otherwise b is a *don't care bit* of t.

A k-term is a term for which the number of don't care bits equals k. A minterm is 0-term.

Definition 2.3 Let t be a term over a set of bits B and let k be a nonnegative integer. Then t is a *k-term* over B iff $|B_\times(t)| = k$. The set $\mathcal{T}_k(B)$ is used to denote the set of all k-terms over B. A *minterm* over B is a 0-term over B.

A term over B that assigns 0 to all its bits, is called an all-0-term over B. In the same way, all-1-terms and all-$\times$-terms are defined.

Definition 2.4 Let B be a set of bits. Let a be an element of the set $\{0, 1, \times\}$. Then the *all-a-term* over B, which is denoted with $t_a(B)$ is the term over B which assigns a to each bit in B: $\forall_{b \in B}\ t_a(B)(b) = a$.

Two terms which are defined over disjunct sets of bits can be combined into a new term, which is called the concatenation of both terms.

Definition 2.5 Let B and B' be two disjunct sets of bits, and let $t \in \mathcal{T}(B)$ be a term over B and let $t' \in \mathcal{T}(B')$ be a term over B'. Then the *concatenation* of t and t', which is denoted with $t \cdot t'$, is a term over $B \cup B'$ defined by

$$\forall_{b \in B \cup B'}\ t \cdot t'(b) = \begin{cases} t(b) & \text{if } b \in B; \\ t'(b) & \text{if } b \in B'. \end{cases}$$

The set of bits over which a term is defined can be reduced by projecting the term on a subset of this set of bits.

Definition 2.6 Let t be a term over a set of bits B, and let B' be a subset of B. Then the *projection* of term t on B' is a term over B' which is denoted by $[t]_{B'}$ and given by

$$\forall_{b \in B'}\ [t]_{B'}(b) = t(b).$$

When two terms t and t' are defined over the same set of bits, and it holds that each 0-bit of t is also a 0-bit of t', and each 1-bit of t is also a 1-bit of t', then we say that term t covers term t'.

Definition 2.7 Let B be a set of bits. Then a term $t \in \mathcal{T}(B)$ *covers* a term $t' \in \mathcal{T}(B)$ iff

$$B_0(t) \subseteq B_0(t') \wedge B_1(t) \subseteq B_1(t')$$

and this will be denoted with $t \geq t'$.

Definition 2.8 Let t be a term, and let k be an integer such that $0 \leq k \leq |B_\times(t)|$. Then the *set of covered k-terms* of t, which is denoted with $C_k(t)$, is defined by

$$C_k(t) = \{t' \in T_k(B) \mid t' \leq t\}.$$

2.1.2 Sequential Machine Definition

In this report, we will use the term sequential machine for a sequential Mealy machine with binary encoded inputs, symbolic states, binary encoded outputs, a next state function which may be incompletely specified and an output function which may be incompletely specified.

Definition 2.9 A *sequential machine* is a structure

$$(IB, S, OB, D, \delta, \lambda),$$

where

- IB is the finite non-empty set of *input bits*;
- S is the finite non-empty set of *states*;
- OB is the finite set of *output bits*;
- $D \subseteq S \times \mathcal{T}_0(IB)$ is the *machine domain*;
- $\delta : D \rightarrow S$ is the *next-state function*;
- $\lambda : D \rightarrow \mathcal{T}(OB)$ is the *output function*.

Example 2.1 Consider the sequential machine which is given by the structure $(IB, S, OB, D, \delta, \lambda)$, where

$$\begin{aligned} IB &= \{x_1, x_2\}, \\ S &= \{s_1, s_2, s_3\}, \\ OB &= \{y_1, y_2, y_3\}, \end{aligned}$$

and D, δ and λ are given by table 2.1. It should be clear that D consists of all (s, x) pairs that appear in this table.

Table 2.1 Next state and output function of M

s	x		$\delta(s,x)$	$\lambda(s,x)$		
	x_1	x_2		y_1	y_2	y_3
s_1	0	0	s_1	1	0	0
	0	1	s_1	–	1	0
	1	0	s_2	1	0	0
s_2	0	0	s_1	–	–	0
	0	1	s_3	1	0	1
	1	0	s_2	0	1	0
	1	1	s_3	1	0	1
s_3	0	0	s_2	0	1	0
	0	1	s_3	–	0	0
	1	0	s_2	0	1	0
	1	1	s_3	–	0	0

For a sequential machine M, it may be the case that the values that are assigned to a subset OB' of output bits are exactly known when the current state is known and the values of a subset IB' of the input bits are known. In this case, we say that the sequential machine M contains a functional dependency from IB' to OB'. This is formalised in the next definition.

Definition 2.10 Let $M = (IB, S, OB, D, \delta, \lambda)$ be a sequential machine, and let IB' and OB' be subsets of IB and OB, respectively. Then M contains a *functional dependency* from IB' to OB' iff the following two

conditions are satisfied:

- $\forall_{(s,x)\in D}\ OB' \cap B_x(\lambda(s,x)) = \emptyset$;
- $\forall_{(s,x),(s,x')\in D}\ [x]_{IB'} = [x']_{IB'} \Rightarrow [\lambda(s,x)]_{OB'} = [\lambda(s,x')]_{OB'}$.

Example 2.2 The sequential machine of example 2.1 has a functional dependency from $\{x_2\}$ to $\{y_3\}$, and from $\{x_1, x_2\}$ to $\{y_3\}$.

When we want to consider the behaviour of a sequential machine when we apply a sequence of input minterms to the machine, we need to introduce the notion of sequences, together with the corresponding formal notations.

Definition 2.11 Let A be a set of arbitrary elements. Then a *sequence* over A is a possibly empty succession of elements of A, denoted with ε if it is empty and with

$$\langle a_1, a_2, a_3, \ldots, a_n \rangle,$$

where $n > 0$ and $a_i \in A$ for $1 \leq i \leq n$, if it is non-empty. The set of all sequences over A is denoted with $\mathcal{Q}_0(A)$, and the set of all non-empty sequences over A is denoted with $\mathcal{Q}(A)$. Two sequences X and X' over the same set A can be *concatenated*. The result is written as $X \cdot X'$ and consists of the sequence which is formed by appending the sequence X' to the tail of X.

The extended next state function of a sequential machine will be defined as the function that, given a certain start state and a certain sequence of input minterms which is offered to the machine, computes the resulting state of the machine. This resulting state may be undefined (equal to $\Box$), when the domain of the sequential machine does not contain all possible state-input combinations.

Definition 2.12 Let $M = (IB, S, OB, D, \delta, \lambda)$ be a sequential machine. Then the *extended next state function* of M is the function

$$\overline{\delta} : S \times \mathcal{Q}_0(\mathcal{T}_0(IB)) \rightarrow S \cup \{\Box\},$$

which is defined by

$$\begin{aligned} \overline{\delta}(s, \varepsilon) &= s, \\ \overline{\delta}(s, X \cdot \langle x \rangle) &= \begin{cases} \delta(\overline{\delta}(s, X), x) & \text{if } (\overline{\delta}(s, X), x) \in D \\ \Box & \text{otherwise} \end{cases} \end{aligned}$$

for $s \in S$, $X \in \mathcal{Q}_0(\mathcal{T}_0(IB))$ and $x \in \mathcal{T}_0(IB)$.

Example 2.3 Consider the sequential machine of example 2.1. For this machine, application of the extended next state function is demonstrated

by

$\overline{\delta}(s_1, \varepsilon) = s_1;$
$\overline{\delta}(s_1, \langle 10 \rangle) = s_2;$
$\overline{\delta}(s_1, \langle 10, 00 \rangle) = s_1;$
$\overline{\delta}(s_1, \langle 10, 00, 11 \rangle) = \Box.$

In the same way, an extended output function is defined. This function determines the output term that results when a certain non-empty sequence of input minterms is applied to the sequential machine, starting in a certain state. This extended next state function yields the result undefined ($\Box$) if the input sequence can not be applied from the start state, because the domain of the machine is exceeded.

Definition 2.13 Let $M = (IB, S, OB, D, \delta, \lambda)$ be a sequential machine. Then the *extended output function* of M is the function

$$\overline{\lambda} : S \times \mathcal{Q}(\mathcal{T}_0(IB)) \rightarrow \mathcal{T}(OB) \cup \{\Box\},$$

which is defined by

$$\overline{\lambda}(s, X \cdot \langle x \rangle) = \begin{cases} \lambda(\overline{\delta}(s, X), x) & \text{if } (\overline{\delta}(s, X), x) \in D \\ \Box & \text{otherwise} \end{cases}$$

for $s \in S$, $X \in \mathcal{Q}_0(\mathcal{T}_0(IB))$ and $x \in \mathcal{T}_0(IB)$.

Example 2.4 Consider the sequential machine of example 2.1. For this machine, application of the extended output function is demonstrated by

$\overline{\lambda}(s_1, \langle 10 \rangle) = 100;$
$\overline{\lambda}(s_1, \langle 10, 00 \rangle) = --0;$
$\overline{\lambda}(s_1, \langle 10, 00, 11 \rangle) = \Box.$

2.1.3 Transition Tables

When we want to present a sequential machine in a compact and clear form, we mostly use transition tables or transition graphs. In this report, we only use transition tables. The next two definitions make clear what a transition table is.

Definition 2.14 A *horizontal* over a set of input bits IB, a set of states S and a set of output bits OB, is a structure (cs, i, ns, o), where

- $cs \in S$ is the *current state*;
- $i \in \mathcal{T}(IB)$ is the *input term*;
- $ns \in S$ is the *next state*;
- $o \in \mathcal{T}(OB)$ is the *output term*.

The set of all horizontals over IB, S and OB will be denoted with

$\mathcal{H}(IB, S, OB)$.

Definition 2.15 A *transition table* is a structure (IB, S, OB, H), where

- IB is the set of *input bits*;
- S is the set of *states*;
- OB is the set of *output bits*;
- $H \subseteq \mathcal{H}(IB, S, OB)$ the a set of *horizontals*.

It is required that all horizontals of H are orthogonal, i.e. for each $(s, x) \in S \times \mathcal{T}_0(IB)$ there is at most one horizontal $h \in H$ such that $h.cs = s$ and $x \leq h.i$.

The next definition relates transition tables to sequential machines.

Definition 2.16 Let $H = (IB, S, OB, H)$ be a transition table. Then the *described sequential machine* of H is the sequential machine given by $(IB, S, OB, D, \delta, \lambda)$. D, δ and λ are defined as follows. Let $(s, x) \in S \times \mathcal{T}_0(IB)$. Then (s, x) is an element of D if and only if there is a horizontal $h \in H$ such that $h.cs = s$ and $x \leq h.i$. When this is the case, $\delta(s, x)$ equals $h.ns$, and $\lambda(s, x)$ equals $h.o$.

Example 2.5 In table 2.2, a transition table is given. The described sequential machine of this table is the sequential machine M of example 2.1.

Table 2.2 Transition table of machine M

cs	i		ns	o		
	x_1	x_2		y_1	y_2	y_3
s_1	0	0	s_1	1	0	0
	0	1	s_1	–	1	0
	1	0	s_2	1	0	0
s_2	0	0	s_1	–	–	0
	–	1	s_3	1	0	1
	1	0	s_2	0	1	0
s_3	–	0	s_2	0	1	0
	–	1	s_3	–	0	0

2.1.4 Realization

One can think about sequential machines at different levels of abstraction. A sequential machine description can be very close to a physical implementation in hardware, but it can also be a machine specification at a high abstraction level. Realization can be seen as a process

of transforming a machine to a machine at a lower abstraction level, thereby introducing more specific implementation details and limiting the freedom that is left in the sequential machine. During the realization process, it is important to preserve the output behaviour of the original sequential machine.

The decomposition of sequential machines into PLAM networks is a special case of realizing a sequential machine with another sequential machine, because we can determine the sequential machine that is implemented by a PLAM network. So, when we describe precisely the conditions that must hold to obtain realization, we can verify the correctness of the PLAM network decompositions that are generated by the decomposition method.

A sequential machine M' realizes a sequential machine M if and only if for each input sequence applied to M and M', it holds that when M gives a defined output, machine M' gives a defined output that is covered by the output of M. The following definition gives further details.

Definition 2.17 Consider two sequential machines with the same set of input bits and the same set of output bits: $M = (IB, S, OB, D, \delta, \lambda)$ and $M' = (IB, S', OB, D', \delta', \lambda')$. Then M' *realizes the output behaviour* of M iff for each state $s \in S$ there exists a state $s' \in S'$ such that for each non-empty input sequence $X \in \mathcal{Q}(\mathcal{T}_0(IB))$:

$$\overline{\lambda}(s, X) \neq \square \Rightarrow (\overline{\lambda'}(s', X) \neq \square \wedge \overline{\lambda}(s, X) \geq \overline{\lambda'}(s', X))$$

In this report, a sequential machine will be realized by a PLAM network in a special way, that is by constructing a state homomorphism from the sequential machine M to the implemented machine M' of the PLAM network. A state homomorphism is a function that assigns a state of M' to every state of M, such that this function preserves the next state function and almost preserves the output function.

Definition 2.18 Consider two sequential machines with the same set of input bits and the same set of output bits: $M = (IB, S, OB, D, \delta, \lambda)$ and $M' = (IB, S', OB, D', \delta', \lambda')$. Then a *state homomorphism* from M to M' is a function $\phi : S \rightarrow S'$ which maps states of M on states of M', such that for all $(s, x) \in D$:

$$(\phi(s), x) \in D' \wedge \phi(\delta(s, x)) = \delta'(\phi(s), x) \wedge \lambda(s, x) \geq \lambda'(\phi(s), x)$$

It can be proved that existence of a state homomorphism is a sufficient condition for output behaviour realization. The next two theorems—the first being just a preparation for the second—work out this statement.

Theorem 2.1 Consider two sequential machines with the same set of input bits and the same set of output bits: $M = (IB, S, OB, D, \delta, \lambda)$ and $M' = (IB, S', OB, D', \delta', \lambda')$. Let ϕ be a state homomorphism from M to M'. Then for each state $s \in S$ and input sequence $X \in \mathcal{Q}_0(\mathcal{T}_0(IB))$:

$$\overline{\delta}(s, X) \neq \square \Rightarrow \phi(\overline{\delta}(s, X)) = \overline{\delta'}(\phi(s), X) \qquad (1)$$

Proof: Suppose that there is a state homomorphism ϕ from M to M'. Let $s \in S$ be a state of machine M. Now we prove by induction that (1) holds for each $X \in \mathcal{Q}_0(\mathcal{T}_0(IB))$.

When X is the empty sequence, it is very straightforward that (1) holds:

$$
\begin{array}{ll}
 & \overline{\delta}(s,\varepsilon) \neq \Box \\
\Rightarrow & \quad \{\phi(s) = \phi(s) \text{ is true, and } P \Rightarrow \mathit{true} \text{ holds for every } P\} \\
 & \phi(s) = \phi(s) \\
\Rightarrow & \quad \{\text{definition 2.12}\} \\
 & \phi(\overline{\delta}(s,\varepsilon)) = \overline{\delta'}(\phi(s),\varepsilon)
\end{array}
$$

Now, suppose that (1) holds for a sequence $X \in \mathcal{Q}_0(\mathcal{T}_0(IB))$. We derive now that in this case, equation (1) also holds for $X \cdot \langle x \rangle$, with $x \in \mathcal{T}_0(IB)$:

$$
\begin{array}{ll}
 & \overline{\delta}(s, X \cdot \langle x \rangle) \neq \Box \\
\Rightarrow & \quad \{\text{definition 2.12}\} \\
 & (\overline{\delta}(s,X),x) \in D \ \wedge \ \overline{\delta}(s,X) \neq \Box \\
\Rightarrow & \quad \{\text{definition 2.18 and induction hypothesis}\} \\
 & (\overline{\delta}(s,X),x) \in D \ \wedge \ (\phi(\overline{\delta}(s,X)),x) \in D' \\
 & \wedge \ \phi(\delta(\overline{\delta}(s,X),x)) = \delta'(\phi(\overline{\delta}(s,X)),x) \\
 & \wedge \ \phi(\overline{\delta}(s,X)) = \overline{\delta'}(\phi(s),X) \\
\Rightarrow & \quad \{\text{rewriting}\} \\
 & (\overline{\delta}(s,X),x) \in D \ \wedge \ (\overline{\delta'}(\phi(s),X),x) \in D' \\
 & \wedge \ \phi(\delta(\overline{\delta}(s,X),x)) = \delta'(\overline{\delta'}(\phi(s),X),x) \\
\Rightarrow & \quad \{\text{definition 2.12}\} \\
 & \phi(\overline{\delta}(s, X \cdot \langle x \rangle)) = \overline{\delta'}(\phi(s), X \cdot \langle x \rangle)
\end{array}
$$

Thus, we have proved that (1) holds. **Q.E.D.**

Theorem 2.2 Consider two sequential machines with the same set of input bits and the same set of output bits: $M = (IB, S, OB, D, \delta, \lambda)$ and $M' = (IB, S', OB, D', \delta', \lambda')$. Then M' realizes the output behaviour of M if there exists a state homomorphism from M to M'.

Proof: Suppose that ϕ is a state homomorphism from M to M'. Let $s \in S$ be a state of machine M, and let $X \in \mathcal{Q}(\mathcal{T}_0(IB))$. Then X can be written as $X = X' \cdot \langle x \rangle$, with $X' \in \mathcal{Q}_0(\mathcal{T}_0(IB))$ and $x \in \mathcal{T}_0(IB)$. Then we derive:

$$
\begin{array}{ll}
 & \overline{\lambda}(s,X) \neq \Box \\
\Rightarrow & \quad \{X = X' \cdot \langle x \rangle\} \\
 & \overline{\lambda}(s, X' \cdot \langle x \rangle) \neq \Box \\
\Rightarrow & \quad \{\text{definition 2.13}\} \\
 & (\overline{\delta}(s,X'),x) \in D \ \wedge \ \overline{\delta}(s,X') \neq \Box \\
\Rightarrow & \quad \{\text{definition 2.18 and theorem 2.1}\} \\
 & (\overline{\delta}(s,X'),x) \in D \ \wedge \ (\phi(\overline{\delta}(s,X')),x) \in D' \\
 & \wedge \ \lambda(\overline{\delta}(s,X'),x) \geq \lambda'(\phi(\overline{\delta}(s,X')),x) \\
 & \wedge \ \phi(\overline{\delta}(s,X')) = \overline{\delta'}(\phi(s),X') \\
\Rightarrow & \quad \{\text{rewriting}\}
\end{array}
$$

$$
\begin{array}{ll}
 & (\overline{\delta}(s,X'),x) \in D \ \wedge \ (\overline{\delta'}(\phi(s),X'),x) \in D' \\
 & \wedge \ \lambda(\overline{\delta}(s,X'),x) \geq \lambda'(\overline{\delta'}(\phi(s),X'),x) \\
\Rightarrow & \quad \{\text{definition 2.13}\} \\
 & \overline{\lambda'}(\phi(s),X' \cdot \langle x \rangle) \neq \Box \ \wedge \ \overline{\lambda}(s,X' \cdot \langle x \rangle) \geq \overline{\lambda'}(\phi(s),X' \cdot \langle x \rangle) \\
\Rightarrow & \quad \{X = X' \cdot \langle x \rangle\} \\
 & \overline{\lambda'}(\phi(s),X) \neq \Box \ \wedge \ \overline{\lambda}(s,X) \geq \overline{\lambda'}(\phi(s),X)
\end{array}
$$

From definition 2.17 and the previous derivation, we can conclude that indeed M' realizes the output behaviour of M (take $s' = \phi(s)$ in definition 2.17). **Q.E.D.**

2.2 PLAM Networks

2.2.1 PLAMs

One of the possible implementation structures of a sequential machine is the PLAM (Programmable Logic Array with Memory). A schematic view of a PLAM is presented in figure 2.1.

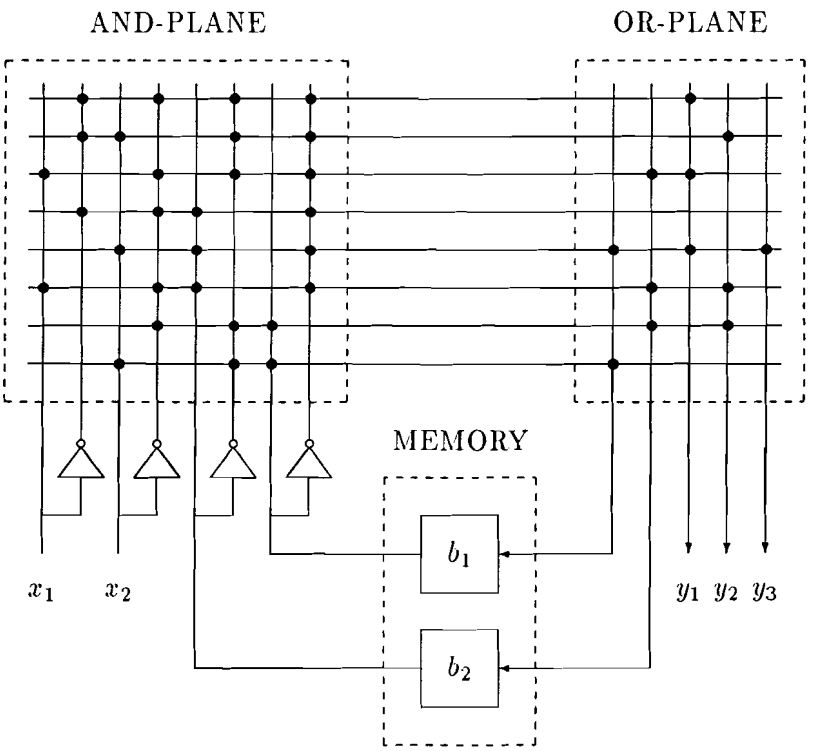

Figure 2.1 Schematic view of a PLAM

Let us explain how such a PLAM operates, and how it can be used to implement a sequential machine. The PLAM contains memory in the form of a limited number of clocked flipflops. These flipflops are used to store the current state of the implemented sequential machine. Each flipflop contains one state bit. The state bits are supplied to the AND-plane, in position as well as in negation. In addition, input bits are supplied to this AND-plane, in position and negation. The AND-plane consists of a matrix, with term lines as rows and literals over the input and state bits as columns. At each intersection of lines, a connection can be made. The value which is assigned to a term line by the matrix, is the logical AND of all literals of the AND-plane that are connected to this term line. When the term line has the value 1, we say that this term line is *active*. The term lines are used as inputs of the OR-plane, which consists of a matrix with term lines as rows, and next state bits and output bits as columns. At each intersection of lines, a connection can be made. The value which is assigned to a next state or output bit equals the logical OR of the values of the term lines that are connected to this next state or output bit. So, a state or output bit is one if and only if at least one of the term lines to which it is connected, is active. The next state bits are connected to the flipflops.

We will give a method that can be used to derive the sequential machine which is implemented by a given PLAM. Before we do so, we make a formal description of a PLAM.

Definition 2.19 A *term line* over a set of input bits IB, a set of state bits SB and a set of output bits OB, is a structure (cs, i, ns, o), where

- $cs \in \mathcal{T}(SB)$ is the *current state term*;
- $i \in \mathcal{T}(IB)$ is the *input term*;
- $ns \in \mathcal{T}_0(SB)$ is the *next state term*;
- $o \in \mathcal{T}_0(OB)$ is the *output term*.

The set of all term lines over IB, SB and OB will be denoted with $\mathcal{TL}(IB, SB, OB)$.

Definition 2.20 A *Programmable Logic Array with Memory (PLAM)* is defined as a structure (IB, SB, OB, TL), where

- IB is the finite non-empty set of *input bits*;
- SB is the finite non-empty set of *state bits*;
- OB is the finite non-empty set of *output bits*;
- $TL \subseteq \mathcal{TL}(IB, SB, OB)$ is a finite set of *term lines*.

Example 2.6 The PLAM which has been presented in figure 2.1 is given by $p = (IB, SB, OB, TL)$, where

$$\begin{aligned} IB &= \{x_1, x_2\}, \\ SB &= \{b_1, b_2\}, \\ OB &= \{y_1, y_2, y_3\}. \end{aligned}$$

The term line set TL is presented in table 2.3.

Table 2.3 Term lines of PLAM p

cs		i		ns		o		
b_1	b_2	x_1	x_2	b_1	b_2	y_1	y_2	y_3
0	0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	1	0
0	0	1	0	0	1	1	0	0
0	1	0	0	0	0	0	0	0
0	1	–	1	1	0	1	0	1
0	1	1	0	0	1	0	1	0
1	0	–	0	0	1	0	1	0
1	0	–	1	1	0	0	0	0

Using this definition of a PLAM, the implemented sequential machine of a PLAM can be easily defined.

Definition 2.21 Let $p = (IB, SB, OB, L)$ be a PLAM. Then the *implemented machine* of p is the sequential machine $(IB, S, OB, D, \delta, \lambda)$, where the state set S is given by $S = \mathcal{T}_0(SB)$, the domain D is given by $S \times \mathcal{T}_0(IB)$ and the next state and output function are defined as follows, for $(s, x) \in D$:

$$\forall_{sb \in SB}\ \delta(s,x)(sb) = \begin{cases} 1 & \text{if } \exists_{l \in TL}\ s \leq l.cs \wedge x \leq l.i \wedge l.ns(sb) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\forall_{ob \in OB}\ \lambda(s,x)(ob) = \begin{cases} 1 & \text{if } \exists_{l \in TL}\ s \leq l.cs \wedge x \leq l.i \wedge l.o(ob) = 1 \\ 0 & \text{otherwise} \end{cases}$$

The implemented machine of PLAM p is denoted with $M_{impl}(p)$.

Example 2.7 Consider the PLAM of example 2.6. The transition table of the implemented machine of this PLAM is given in table 2.4.

Table 2.4 Transition table of the implemented machine of PLAM p

cs		i		ns		o		
b_1	b_2	x_1	x_2	b_1	b_2	y_1	y_2	y_3
0	0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	1	0
0	0	1	0	0	1	1	0	0
0	0	1	1	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	1	–	1	1	0	1	0	1
0	1	1	0	0	1	0	1	0
1	0	–	0	0	1	0	1	0
1	0	–	1	1	0	0	0	0
1	1	–	–	0	0	0	0	0

When a sequential machine is implemented with a PLAM, it must be so that the implemented machine of this PLAM realizes the output behaviour of the sequential machine. Suppose that a sequential machine M is given, together with a PLAM p. Then we can prove that p implements M, by giving a state homomorphism from M to $M_{impl}(p)$. Remember from theorem 2.2 that existence of a state homomorphism is a sufficient condition for output behaviour realization. Because we have not proved that it is a necessary condition—which is probably not true anyway—the non-existence of a state homomorphism does not imply that the output behaviour is not realized.

Example 2.8 Consider the sequential machine M of example 2.1, and the PLAM p of example 2.6. Define the function $\phi : M.S \rightarrow M_{impl}(p).S$ by

$$\begin{aligned} \phi(s_1) &= 00; \\ \phi(s_2) &= 01; \\ \phi(s_3) &= 10. \end{aligned}$$

By comparing table 2.2 with 2.4, it can easily be checked that ϕ is a state homomorphism from M to $M_{impl}(p)$. Thus, the output behaviour of M is realized by the implemented machine of PLAM p. Thus, p implements M.

In this report, we especially aim at the realization of sequential machines with constrained PLAMs. The constraint limits the number of input bits, the number of state bits, the number of output bits and the number of term lines for the PLAMs that may be used.

Definition 2.22 A *PLAM-size constraint* c consists of a structure $(nr_{IB}, nr_{SB}, nr_{OB}, nr_{TL})$, where

- nr_{IB} is a positive integer, called the *input bit constraint*;
- nr_{SB} is a positive integer, called the *state bit constraint*;
- nr_{OB} is a positive integer, called the *output bit constraint*;

- nr_{TL} is a positive integer, called the *term line constraint.*

A PLAM p *satisfies constraint* c iff $|p.IB| \leq nr_{IB}$, $|p.SB| \leq nr_{SB}$, $|p.OB| \leq nr_{OB}$ and $|p.TL| \leq nr_{TL}$.

When it is not possible to implement a given sequential machine with one PLAM satisfying the constraint, a network of PLAMs can be used. This is the subject of the next section.

2.2.2 Output Independent PLAM Networks

When a sequential machine is large, compared to the constraint, it can be tried to implement it with a number of cooperating PLAMs, which all satisfy the constraint. These cooperating PLAMs form a PLAM network. In the following example, we will make clear what has to be understood by a PLAM network.

Example 2.9 In figure 2.2, an example of a PLAM network is given. This network contains two PLAMs, named p_1 and p_2. The term lines of these PLAMs are given in table 2.5 and 2.6, respectively.

A PLAM network contains three types of nets: external input nets, internal nets and external output nets. To each net, a bit is associated. The distinction between the three types of nets should be clear, given that the example network contains external input bits x_1 and x_2, internal bit z_{s_2} and external output bits y_1, y_2 and y_3. The input bits of each PLAM are external input bits or internal bits of the network, and the output bits of each PLAM are external output bits or internal bits of the network. We assume that each internal bit and external output bit has a default value of 0, and gets the value 1 if and only if there is a PLAM which assigns the value 1 to this bit. This can be implemented easily by OR-ing the values that are assigned to the bit by different PLAMs. The OR function can be implemented without costs if the PLAMs have open-collector or three state outputs.

Since there are two state bits per PLAM, the network has 16 different states. In general, a state of a PLAM network will be defined as a function which assigns a state s to each PLAM p of the network, where s must be a state of the implemented machine of p.

The number of connections of the network will be defined as the total number of used input and output bits of all PLAMs in the network. For the example network, PLAM p_1 has 6 connections, and PLAM p_2 has 5 connections, so the number of connections of the network equals 11.

The following definitions give further details about the concepts that have been introduced in the previous example.

Definition 2.23 A *PLAM network* is a structure (IB, EB, OB, P),

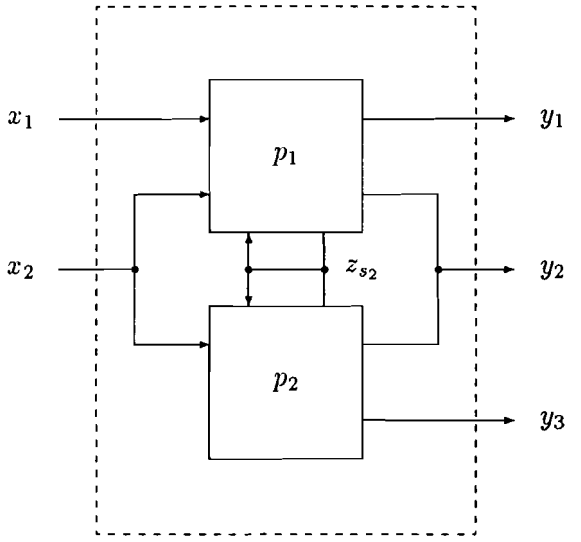

Figure 2.2 PLAM network n

Table 2.5 Component PLAM p_1 of PLAM network n

cs		i			ns		o		
b_1	b_2	x_1	x_2	z_{s_2}	b_1	b_2	y_1	y_2	z_{s_2}
0	1	0	0	–	0	1	1	0	0
0	1	0	1	–	0	1	0	1	0
0	1	1	0	–	1	0	1	0	1
1	0	0	0	–	0	1	0	0	0
1	0	–	1	–	0	0	1	0	0
1	0	1	0	–	1	0	0	0	1
–	–	–	–	1	1	0	0	0	0

Table 2.6 Component PLAM p_2 of PLAM network n

cs		i		ns		o		
b_1	b_2	x_2	z_{s_2}	b_1	b_2	y_2	y_3	z_{s_2}
0	1	0	–	0	0	1	0	0
0	1	1	–	1	0	0	1	0
1	0	0	–	0	1	1	0	1
1	0	1	–	1	0	0	0	0
–	–	–	1	0	1	0	0	0

where

- IB is a finite non-empty set of *external input bits*;
- EB is a finite set of *internal bits*;
- OB is a finite set of *external output bits*;
- P is a non-empty set of PLAMs, named *component PLAMs*, such that for each component PLAM p, $p.IB \subseteq IB \cup EB$, and $p.OB \subseteq EB \cup OB$.

An *implemented state* s of this network is a function

$$s : P \rightarrow \bigcup_{p \in P} M_{impl}(p).S$$

such that

$$\forall_{p \in P}\ s(p) \in M_{impl}(p).S.$$

The set of all implemented states of network n is denoted with $S_{impl}(n)$.

Example 2.10 The PLAM network n of example 2.9 is given by

$$\begin{array}{lcl} n.IB & = & \{x_1, x_2\}; \\ n.EB & = & \{z_{s_2}\}; \\ n.OB & = & \{y_1, y_2, y_3\}; \\ n.P & = & \{p_1, p_2\}. \end{array}$$

Definition 2.24 Let n be a PLAM network, and let p be a component PLAM of this network. Then the *set of external input bits* $EIB_n(p)$, the *set of internal input bits* $IIB_n(p)$, the *set of internal output bits* $IOB_n(p)$ and the *set of external output bits* $EOB_n(p)$ are defined as follows:

$$\begin{array}{ll} EIB_n(p) & = p.IB \cap n.IB; \\ IIB_n(p) & = p.IB \cap n.EB; \\ IOB_n(p) & = p.OB \cap n.EB; \\ EOB_n(p) & = p.OB \cap n.OB. \end{array}$$

The *number of connections* of n is equal to

$$\bigcup_{p \in n.P} |p.IB \cup p.OB|,$$

and is denoted with $nr_{conn}(n)$.

Example 2.11 For the PLAM network n of example 2.9:

$$
\begin{aligned}
EIB_n(p_1) &= \{x_1, x_2\};\\
IIB_n(p_1) &= \{z_{s_2}\};\\
IOB_n(p_1) &= \{z_{s_2}\};\\
EOB_n(p_1) &= \{y_1, y_2\};\\
\\
EIB_n(p_2) &= \{x_2\};\\
IIB_n(p_2) &= \{z_{s_2}\};\\
IOB_n(p_2) &= \{z_{s_2}\};\\
EOB_n(p_2) &= \{y_2, y_3\};
\end{aligned}
$$

$nr_{conn}(n) = 11.$

In general, it is not guaranteed that a PLAM network behaves like a sequential machine. However, the PLAMs and their interconnections can be chosen such that sequential behaviour is guaranteed. For example, sequential behaviour is guaranteed if each PLAM output bit only depends on the current state and external input bits of this PLAM. A PLAM network that satisfies this condition, will be called an output independent PLAM network.

Let us explain why an output independent network behaves like a sequential machine. From the definition of an output independent network, it follows that the values of the internal bits and external output bits of the network are completely determined by the current state and external input bits of the network. Since the network state and input uniquely determine all internal bit values, they also uniquely determine the input bit values of all PLAMs, and thus also the next states of all PLAMs, and thus also the next state of the network. Because of the fact that the network state and input uniquely determine the values of all external output bits, they also uniquely determine the network output.

In this report, we will implement sequential machines with output independent PLAM networks. Note, that in an output independent PLAM network, only direct communications between PLAMs are possible. When a PLAM sends a message indirectly to another PLAM, there must be an intermediate PLAM for which the internal output bits depend on the internal input bits. This is not possible when the PLAM network is output independent. Remark that the delay of an output independent PLAM network is at most equal to the maximum delay of a component PLAM, multiplied by two.

Definition 2.25 A PLAM network n is *output independent* iff for every component PLAM $p \in n.P$, the implemented machine of p has a functional dependency from $EIB_n(p)$ to $p.OB$. Let n be an output independent PLAM network. Then the *output function* of network n is the function

$$\alpha : S_{impl}(n) \times \mathcal{T}_0(n.IB) \rightarrow \mathcal{T}_0(n.EB \cup n.OB),$$

such that for each $s \in S_{impl}(n)$, $x \in \mathcal{T}_0(n.IB)$ and $b \in n.EB \cup n.OB$, the value of $\alpha(s,x)(b)$ is 1 if

$$\exists_{p \in n.P}\ b \in p.OB \wedge M_{impl}(p).\lambda(s(p), [x]_{EIB_n(p)} \cdot t_0(IIB_n(p)))(b) = 1$$

and 0 otherwise.

Definition 2.26 Let n be an output independent PLAM network. Then the *implemented machine* of n is the sequential machine

$$(n.IB, S_{impl}(n), n.OB, D, \delta, \lambda),$$

where D equals $S_{impl}(n) \times \mathcal{T}_0(n.IB)$. For each $(s,x) \in D$, the next state and output is given by

$$\forall_{p \in n.P}\ \delta(s,x)(p) = M_{impl}(p).\delta(s(p), [x \cdot \alpha(s,x)]_{p.IB}),$$
$$\forall_{ob \in n.OB}\ \lambda(s,x)(ob) = n.\alpha(s,x)(ob),$$

where α is the output function of network n. The implemented machine of network n is denoted with $M_{impl}(n)$.

Example 2.12 Consider the PLAM network n of example 2.9. The transition tables of the implemented machines of PLAM p_1 and p_2 are shown in table 2.7 and table 2.8, respectively. Using these transition tables, it is easy to check that the PLAM network n is output independent. Thus, an output function α can be computed, which gives the values of the internal bits and external output bits for each network state and input. For a subset of network states, this output function is presented in table 2.9. This table also gives partial information about the next state and output function of the implemented machine of n, by giving the next network state and network output for the same subset of states.

The network states which are placed in table 2.9 are not arbitrary chosen. The reason for this choice is that these states correspond with states of the sequential machine M, which has been introduced in example 2.1. Compare the transition table of M (table 2.2) to the table which contains partial information about the implemented machine of PLAM network n (table 2.9). Then, it is easy to check that the function ϕ from the states of M to the states of $M_{impl}(n)$, given by table 2.10, is a state homomorphism from M to the implemented machine of n. According to theorem 2.2, it may be concluded that the implemented machine of the PLAM network n realizes the output behaviour of sequential machine M. Therefore, we know that the PLAM network n may be used to implement M.

In the section about PLAMs, we have already said that we will only consider implementations based on restricted PLAMs, i.e. PLAMs satisfying a predefined constraint. The constraint which must be satisfied by each PLAM can easily be extended into a constraint which must be satisfied by a PLAM network.

Definition 2.27 Let c be a PLAM-size constraint. Then, a PLAM network n *satisfies constraint* c iff each component PLAM of n satisfies constraint c.

2.3 Decomposition

Finally, we are able to define precisely what we mean with a decomposition of a sequential machine, using an output independent PLAM network as the implementation structure.

Definition 2.28 Let M be a sequential machine. An *output independent PLAM network decomposition* of M is an output independent PLAM network n such that the implemented machine of n realizes M.

2.4 Precise Problem Statement

The decomposition problem which is handled in this report can now be formulated in a very compact way, using the definitions that have been given in this chapter. The problem is as follows.

Given: a transition table T and a PLAM-size constraint c.

Find: an output independent PLAM network decomposition n of the described machine of T such that n satisfies constraint c, n has a minimal number of PLAMs, and — under the condition that n has a minimal number of PLAMs—n has a minimal number of connections.

Table 2.7 Transition table of $M_{impl}(p_1)$

cs		i			ns		o		
b_1	b_2	x_1	x_2	z_{s_2}	b_1	b_2	y_1	y_2	z_{s_2}
0	0	–	–	0	0	0	0	0	0
0	0	–	–	1	1	0	0	0	0
0	1	0	0	0	0	1	1	0	0
0	1	0	0	1	1	1	1	0	0
0	1	0	1	0	0	1	0	1	0
0	1	0	1	1	1	1	0	1	0
0	1	1	0	–	1	0	1	0	1
0	1	1	1	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0
1	0	0	0	0	0	1	0	0	0
1	0	0	0	1	1	1	0	0	0
1	0	–	1	0	0	0	1	0	0
1	0	–	1	1	1	0	1	0	0
1	0	1	0	–	1	0	0	0	1
1	1	–	–	0	0	0	0	0	0
1	1	–	–	1	1	0	0	0	0

Table 2.8 Transition table of $M_{impl}(p_2)$

cs		i		ns		o		
b_1	b_2	x_2	z_{s_2}	b_1	b_2	y_2	y_3	z_{s_2}
0	0	–	0	0	0	0	0	0
0	0	–	1	0	1	0	0	0
0	1	0	0	0	0	1	0	0
0	1	0	1	0	1	1	0	0
0	1	1	0	1	0	0	1	0
0	1	1	1	1	1	0	1	0
1	0	0	–	0	1	1	0	1
1	0	1	0	1	0	0	0	0
1	0	1	1	1	1	0	0	0
1	1	–	0	0	0	0	0	0
1	1	–	1	0	1	0	0	0

Table 2.9 Partial output function α of network n, and partial next state function δ and output function λ of the implemented machine of n

s				x		$n.\alpha(s,x)$				$\delta(s,x)$				$\lambda(s,x)$		
p_1		p_2								p_1		p_2				
b_1	b_2	b_1	b_2	x_1	x_2	z_{s_2}	y_1	y_2	y_3	b_1	b_2	b_1	b_2	y_1	y_2	y_3
0	1	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0
0	1	0	0	0	1	0	0	1	0	0	1	0	0	0	1	0
0	1	0	0	1	0	1	1	0	0	1	0	0	1	1	0	0
0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0
1	0	0	1	0	1	0	1	0	1	0	0	1	0	1	0	1
1	0	0	1	1	0	1	0	1	0	1	0	0	1	0	1	0
1	0	0	1	1	1	0	1	0	1	0	0	1	0	1	0	1
0	0	1	0	0	0	1	0	1	0	1	0	0	1	0	1	0
0	0	1	0	1	0	1	0	1	0	1	0	0	1	0	1	0
0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0

Table 2.10 State homomorphism ϕ from M to $M_{impl}(n)$

s	$\phi(s)$			
	p_1		p_2	
	b_1	b_2	b_1	b_2
s_1	0	1	0	0
s_2	1	0	0	1
s_3	0	0	1	0

3 Decomposition Model

In this chapter, a new decomposition model is proposed, which is based on task distributions. With this new model, it is possible to describe PLAM network decompositions of sequential machines in terms of task distributions.

3.1 Overview of the Model

The main part of the research consisted of developing a new model for decomposing sequential machines into PLAM networks, with which it is possible to handle sequential machines with large states. First of all, it was important to have a good insight in decomposition problems. Because of the huge complexity of decomposition problems, it is very important to have a good feeling about decomposition. To obtain feeling about decomposition, one needs a good decomposition model at a high level of abstraction. Such a model makes it possible to reason about decompositions without seeing all complexity at the lower abstraction levels. Hartmanis has designed such a model which is based on information flows [2]. This model has proven to be very useful for thinking about decomposition. Decomposition can also be modeled as a distribution of tasks. For the specific decomposition problem which is reported, we have developed a new model which is based on task distribution concepts.

With the new task distribution model, a decomposition can be made by determining subtasks of a sequential machine, distributing them over PLAMs, and constructing an output independent PLAM network decomposition based on this task distribution. The model which is proposed in this report is more general than the model proposed by Levin in [6]. The drawback of the Levin model, which causes the problems with large states, is that the task distribution is coarse grained. One elementary subtask consists of computing all information given a certain current state, and such a subtask may be too large for one PLAM. The proposed new model uses fine grained subtasks called atoms. One atom corresponds with the task of performing a single machine operation for one current state and one input of the sequential machine. A machine operation consists of determining the next state or determining the value of an output bit.

It is important to note that when atoms are distributed over PLAMs instead of states, the decomposition looses its sequential behaviour, and becomes simultaneous. The distinction between sequential and simultaneous decomposition has been introduced in [3].

In the model of Levin, the state partition defines a task distribution among the PLAMs. The current state of the sequential machine determines which PLAM computes the next state and output information. This implies that only one PLAM is really active at a time, and therefore the model of Levin describes *sequential decompositions.*

In the new model which is described in this report, atoms are distributed over machines. Multiple PLAMs can be active simultaneously, because for a certain current state and input, the corresponding atoms can be assigned to different PLAMs, and performed simultaneously. Therefore, the decomposition model that is proposed in this report yields *simultaneous decompositions.*

3.2 Determination of Subtasks

This section describes how we can distinguish subtasks of the prototype machine, such that these subtasks can be assigned to PLAMs of a PLAM network. Firstly, we define the operations that can be performed by a PLAM in a PLAM-network, in terms of the prototype machine M. We distinguish operations related to the next state and operations related to the output. For the next state, an operation consists of making the next state equal to a certain state, and for the output an operation consists of setting a certain output bit at the value one. Making an output bit equal to zero is not considered as an operation, because all output bits of a PLAM-network are zero by default.

Definition 3.1 Let $M = (IB, S, OB, D, \delta, \lambda)$ be a sequential machine. A *δ-operation* δ_s of M, where $s \in S$, represents the operation of making the next state equal to s. A *λ-operation* λ_{ob} of M, where $ob \in OB$, represents the operation of making the output bit ob equal to 1. The set of all operations of M, which is denoted with $Operations(M)$, is defined as the set containing all δ- and λ-operations of M:

$$Operations(M) = \{\delta_s \mid s \in M.S\} \cup \{\lambda_{ob} \mid ob \in M.OB\}$$

An atomic task will now be defined as the performance of one operation for a certain current state and input minterm.

Definition 3.2 Let $M = (IB, S, OB, D, \delta, \lambda)$ be a sequential machine. Then an *atomic task (abbreviated as atom)* of M is a structure (cs, x, op), where

- $cs \in S$ is the *current state*;
- $x \in \mathcal{T}_0(IB)$ is the *input minterm*;
- $op \in Operations(M)$ is the *operation.*

Atom (cs, x, op) represents the task of performing operation op when the current state equals cs and the input equals x. The set of all atoms of

machine M is denoted with $Atoms(M)$.

The atoms can be separated into three groups: required atoms, allowed atoms and forbidden atoms. Required atoms are atoms that correspond to tasks that must be performed by the PLAM network, allowed atoms correspond to tasks that may be performed by the PLAM network but are not required, and forbidden atoms correspond to tasks that must not be performed by the PLAM network.

Definition 3.3 Let $M = (IB, S, OB, D, \delta, \lambda)$ be a sequential machine. Then an atom $(s, x, \delta_{s'})$ of M is

- *required* if $(s, x) \in D \wedge \delta(s, x) = s'$;
- *forbidden* if $(s, x) \in D \wedge \delta(s, x) \neq s'$;
- *allowed* in all other cases.

An atom (s, x, λ_{ob}) of M is

- *required* if $(s, x) \in D \wedge \lambda(s, x)(ob) = 1$;
- *forbidden* if $(s, x) \in D \wedge \lambda(s, x)(ob) = 0$;
- *allowed* in all other cases.

The notations

$$RequiredAtoms(M), AllowedAtoms(M), ForbiddenAtoms(M)$$

are used for the sets of required atoms, allowed atoms and forbidden atoms of M, respectively.

Example 3.1 Let M be the sequential machine of example 2.1. The number of atoms of machine M is given by

$$|M.S| \cdot 2^{|M.IB|} \cdot (|M.S| + |M.OB|) = 72.$$

Six of the 72 atoms of M are given below.

- $(s_1, \overline{x_1}\, x_2, \delta_{s_1})$ and $(s_2, x_1\, x_2, \lambda_{y_3})$ are required atoms of M;
- $(s_1, x_1\, x_2, \delta_{s_2})$ and $(s_3, \overline{x_1}\, x_2, \lambda_{y_1})$ are allowed atoms of M;
- $(s_1, \overline{x_1}\, \overline{x_2}, \delta_{s_3})$ and $(s_2, x_1\, \overline{x_2}, \lambda_{y_1})$ are forbidden atoms of M.

When we want to obtain a PLAM network that realizes the prototype machine, each required atom should be implemented by at least one PLAM in the network, and each forbidden atom must not be implemented by any PLAM in the network. Thus, distribution of the required atoms over PLAMs seems to be a good way of obtaining a PLAM network that realizes the prototype machine. However, this method of atom distribution has two large drawbacks.

- There is no direct relation between the structure of a PLAM and the atomic tasks that are performed by a PLAM. The absence of this relation makes it difficult to predict the effect of assigning a certain atom to a PLAM, i.e. the use of the input bit space, state

bit space, output bit space and term line space of this PLAM.
- The number of required atoms of the prototype machine can be very large, resulting in a lot of possible distributions. This makes it difficult to find an optimal distribution.

Looking at these drawbacks, we tried to improve the task distribution model by defining a new sort of task, which is larger than an atomic task (in the sense that it can cover many atomic tasks) and has a direct relation with the structure of a PLAM. This new sort of task will be called a cube.

Definition 3.4 Let $M = (IB, S, OB, D, \delta, \lambda)$ be a sequential machine. Then a *cube* of M is a structure (cs, i, OP), where

- $cs \in S$ is the *current state*;
- $i \in \mathcal{T}(IB)$ is the *input term*;
- $OP \subseteq Operations(M)$ is the *operation set*;

Let us explain how such a cube should be interpreted. The cube performs a task only if it is active. The cube is active only if the current state is cs, and the input minterm of the sequential machine is covered by i. Under the condition that the cube is active, it performs all operations that are contained in the set OP.

Knowing how to interpret a cube, it should be clear that we can define the atoms that are covered by a cube as follows:

Definition 3.5 Let c be a cube of a sequential machine M. Then the *set of covered atoms* of c, which is denoted with $CoveredAtoms(c)$, is given by

$$\{a \in Atoms(M) \mid a.cs = c.cs \wedge a.x \leq c.i \wedge a.op \in c.OP\}$$

When cubes are used for being distributed over PLAMs, they may not cover forbidden atoms. Cubes which do not contain forbidden atoms will be called molecules.

Definition 3.6 Let M be a sequential machine. Then a cube c of M is a *molecule* of M iff

$$CoveredAtoms(c) \subseteq RequiredAtoms(M) \cup AllowedAtoms(M).$$

Example 3.2 Consider the sequential machine M of example 2.1. Three cubes c_1, c_2 and c_3 of M are given below.

$$\begin{aligned} c_1 &= (s_1, \overline{x_1}, \{\delta_{s_1}\}); \\ c_2 &= (s_1, -, \{\lambda_{y_1}\}); \\ c_3 &= (s_2, \overline{x_2}, \{\delta_{s_1}, \lambda_{y_2}\}). \end{aligned}$$

The covered atoms of these cubes are given by table 3.1. The cubes c_1 and c_2 are molecules of M. The cube c_3 is not a molecule of M, because it covers the atom $(s_2, x_1\,\overline{x_2}, \delta_{s_1})$, which is forbidden.

Table 3.1 Covered atoms of c_1, c_2 and c_3

c	$CoveredAtoms(c)$
c_1	$\{(s_1, \overline{x_1}\,\overline{x_2}, \delta_{s_1}), (s_1, \overline{x_1}\,x_2, \delta_{s_1})\}$
c_2	$\{(s_1, \overline{x_1}\,\overline{x_2}, \lambda_{y_1}), (s_1, \overline{x_1}\,x_2, \lambda_{y_1}), (s_1, x_1\,\overline{x_2}, \lambda_{y_1}), (s_1, x_1\,x_2, \lambda_{y_1})\}$
c_3	$\{(s_2, \overline{x_1}\,\overline{x_2}, \delta_{s_1}), (s_2, \overline{x_1}\,\overline{x_2}, \lambda_{y_2}), (s_2, x_1\,\overline{x_2}, \delta_{s_1}), (s_2, x_1\,\overline{x_2}, \lambda_{y_2})\}$

It is straightforward that the set of molecules which have to be distributed, must be chosen such that each required atom is covered by at least one molecule. A molecule set which satisfies this condition will be called a molecule cover.

Definition 3.7 Let M be a sequential machine, and let C be a set of molecules of M. Then C is called a *molecule cover* of M iff

$$RequiredAtoms(M) \subseteq \bigcup_{c \in C} CoveredAtoms(c)$$

3.3 Distribution of Subtasks

A distribution of tasks over PLAMs will be described by a set of blocks, where each block contains molecules that have to be implemented by a single PLAM. Trivially, the union of all blocks must be a molecule cover, for otherwise required atoms are not implemented.

Definition 3.8 Let M be a sequential machine. A *molecule distribution* of M is a set containing non-empty sets of molecules called *blocks*, such that the union of all blocks is a molecule cover of M.

Example 3.3 Consider the sequential machine M which is described by transition table 3.2. An example molecule distribution π of this machine is given by $\{B_1, B_2\}$, where block B_1 and block B_2 are specified in table 3.3 and 3.4, respectively. Note that $B_1 \cup B_2$ is a molecule cover of M.

Table 3.2 Transition table of machine M

cs	i							ns	o								
	x_1	x_2	x_3	x_4	x_5	x_6	x_7		y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9
s_1	0	1	–	1	–	–	–	s_1	1	–	0	–	–	–	–	–	–
	–	–	0	0	–	–	–	s_1	–	1	–	–	1	–	0	–	–
	–	–	1	0	–	–	–	s_2	–	1	0	–	0	–	1	–	–
	1	1	–	1	–	–	–	s_3	0	–	0	–	0	–	–	–	–
	–	0	–	1	–	–	–	s_3	–	0	1	–	1	–	1	–	–
s_2	–	–	–	–	1	1	–	s_1	–	0	–	0	–	–	–	1	–
	–	–	–	–	0	0	–	s_2	–	1	–	1	–	–	0	1	–
	–	–	–	–	0	1	–	s_3	–	–	–	1	–	–	1	0	–
s_3	–	–	–	–	–	–	1	s_1	–	1	–	–	–	1	–	–	0
	–	–	–	–	–	–	0	s_3	–	0	–	–	–	0	–	–	1

Table 3.3 Molecules that are contained in block B_1 of π

cs	i							OP
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
s_1	0	1	–	1	–	–	–	$\{\delta_{s_1}, \lambda_{y_1}\}$
	1	1	–	1	–	–	–	$\{\delta_{s_3}\}$
	–	0	–	1	–	–	–	$\{\delta_{s_3}, \lambda_{y_3}, \lambda_{y_5}\}$
s_3	–	–	–	–	–	–	1	$\{\delta_{s_1}, \lambda_{y_2}, \lambda_{y_6}\}$
	–	–	–	–	–	–	0	$\{\delta_{s_3}, \lambda_{y_9}\}$

Table 3.4 Molecules that are contained in block B_2 of π

cs	i							OP
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
s_1	–	–	–	1	–	–	–	$\{\lambda_{y_7}\}$
	–	–	0	0	–	–	–	$\{\delta_{s_1}, \lambda_{y_2}, \lambda_{y_5}\}$
	–	–	1	0	–	–	–	$\{\delta_{s_2}, \lambda_{y_2}, \lambda_{y_7}\}$
s_2	–	–	–	–	1	1	–	$\{\delta_{s_1}, \lambda_{y_8}\}$
	–	–	–	–	0	0	–	$\{\delta_{s_2}, \lambda_{y_2}, \lambda_{y_4}, \lambda_{y_8}\}$
	–	–	–	–	0	1	–	$\{\delta_{s_3}, \lambda_{y_4}, \lambda_{y_7}\}$

3.4 Construction of the PLAM Network

The step which remains to be taken is to define an output independent PLAM network, based on a certain molecule distribution of a sequential machine M. This PLAM network must be such that it implements a sequential machine that realizes sequential machine M. The network is

constructed such that each component PLAM performs the tasks corresponding with the molecules of a block of the molecule distribution. When we talk about 'the PLAM of a block' or 'the block of a PLAM' in the sequel, we aim at this one-to-one correspondence between blocks of the molecule distribution and PLAMs of the network. This correspondence also makes it possible to talk about 'the molecules of' a certain PLAM or 'the molecules that are placed in' a certain PLAM, with which we mean the molecules that are contained in the block that corresponds with the PLAM.

3.4.1 Indirect Versus Direct Information Supply

We observe that each component PLAM needs information about the current state and about the input of the sequential machine M, since it must be able to determine which of its molecules are active and which are not. In general, information can be be supplied to a component PLAM in two ways: by direct or by indirect information supply. *Direct information supply* means that the PLAM gets information from its own external inputs and its own current state, whereas *indirect information supply* means that the PLAM gets information from other component PLAMs via internal inputs. With regard to indirect information supply, it should be remarked that the fact that the network must be output independent prevents that a PLAM gets information from another PLAM via an intermediate PLAM. Thus, each PLAM can only receive indirect information from its neigbour PLAMs.

With regard to state and input information of the sequential machine M, we choose that every PLAM gets the information directly. Note that this does not imply that the PLAMs work independently of eachother. Later on, we have to introduce communication between PLAMs in order to keep the state information of all PLAMs up-to-date: in certain cases, next state information will be supplied indirectly.

3.4.2 Task Redundancy

It is important to note that it is possible that atomic tasks (atoms) occur more than once in a molecule distribution. When atoms occur more than once, we say that there is *task redundancy*. Task redundancy can occur within blocks (when an atom is covered by two or more molecules of a block), or between blocks (when an atom is covered by two or more molecules belonging to different blocks).

For the network construction, task redundancy is not taken into account. This means that when a certain task has to be performed by a PLAM, we don't look whether this task is already performed by another PLAM. For example, suppose that PLAM p contains the molecule $(s_1, x_1, \{\delta_{s_1}\})$,

and PLAM p' also contains this molecule. When the current state is s_1 and the input bit x_1 is high, PLAM p computes that the next state is s_1, and sends this message to PLAM p', although this is not strictly needed because p' computes this information itself.

3.4.3 State Information Supply

Knowing that a component PLAM derives all needed state information from its own current state, we choose the states of a component PLAM as follows. Each state of M appearing as the current state of at least one molecule of a PLAM, is called a current state of the PLAM. The current states of a PLAM should all be available in this PLAM, since otherwise it would be impossible for the PLAM to know which molecules of its block are active. Therefore, the state set of a PLAM contains a copy of each state that is contained in the current state set of the PLAM. In addition, the state set of a PLAM contains an extra state, called the wait state. We must be sure that at each moment the current state of each PLAM really corresponds with the current state of the sequential machine M. In other words, when the sequential machine M has s as its current state, all PLAMs that have a copy of s should have this copy as their current state, and all other PLAMs should have the wait state as their current state. Initially, it is easy to activate the correct state in each PLAM, since we know the start state of the sequential machine M. Afterwards, the next state information that is computed by the molecules should be processed in such a way that at the next moment each PLAM is still in the correct state. These problems will be discussed later on. First, we will work out further how the state set of a PLAM is translated in a set of coded states.

As we know, the states of a PLAM are not symbolic but binary encoded. Thus, we must find a state assignment for the symbolic state set of each PLAM, which has been defined previously. The state set of a PLAM contains copies of all current states of its block, and in addition a wait state. Because we want to use as few state bits as possible, we choose for minimally encoded states. Then, the number of state bits that is needed for a PLAM equals the ceiled value of the 2-base logarithm of the number of current states in the PLAM, increased with 1 because of the wait state. For the state bits of the PLAMs, we will use the symbols b_1, b_2, etcetera. We will use a special code for the wait state: the code with all bits set to zero. This choice implies that for each PLAM, the wait state will be the next state if no term line is active. Thus, the wait state may be regarded as a sort of default next state for all the PLAMs.

The choice of the codes for other states than the wait state is not of importance here. The only thing which we have to take into account when we choose the state encoding, is that the wait state receives the code consisting of all zeros.

We finish the investigations about the state sets of the PLAMs by formalizing the previously described ideas, and giving an example of their practical use.

Definition 3.9 Let π be a molecule distribution, and let B be a block of π. Then the *current state set* $CS(B)$ and the *state bit set* $SB(B)$ of the PLAM of block B are given by

$$\begin{aligned} CS(B) &= \bigcup_{c \in B} c.cs; \\ SB(B) &= \{b_i \mid 1 \leq i \leq \lceil {}^2\log(|CS(B)| + 1) \rceil\}. \end{aligned}$$

A *state assignment* of the PLAM of block B is an injective function $\beta : CS(B) \rightarrow \mathcal{T}_0(SB(B))$ such that for all $s \in CS(B)$, $\beta(s) \neq t_0(SB(B))$. A *network assigment function* for molecule distribution π is a function γ such, that for each block $B \in \pi$, $\gamma(B)$ gives the state assignment for the PLAM of block B.

Example 3.4 Consider the molecule distribution π of example 3.3. For this molecule distribution, the current state set and the state bit set for the PLAMs of both blocks are given by

$$\begin{aligned} CS(B_1) &= \{s_1, s_3\}; \\ CS(B_2) &= \{s_1, s_2\}; \\ SB(B_1) &= \{b_1, b_2\}; \\ SB(B_2) &= \{b_1, b_2\}. \end{aligned}$$

An example network state assignment γ for molecule distribution π is given in table 3.5.

Table 3.5 Example network state assignment γ

s	$\gamma(B_1)(s)$		$\gamma(B_2)(s)$	
	b_1	b_2	b_1	b_2
s_1	0	1	0	1
s_2			1	0
s_3	1	0		

3.4.4 Input Information Supply

A PLAM not only requires information about the state of sequential machine M, but also about the input of M. Every PLAM should have enough input information at its disposal to check for each of its molecules whether it is active or not. A molecule may only become active if the current input minterm of sequential machine M is covered by the input term of the molecule. If for a certain molecule with input term i, it has to be checked whether i is covered by the current input minterm of machine M, it suffices to compare the term i with the input minterm

of M only for the care bits of i. Remember that we have made the choice that the needed input information will be supplied directly to each PLAM, i.e. each PLAM derives its input information directly from its external input bits, which belong to the input bits of the sequential machine M. Thus each PLAM receives enough input information, if we make the external input bit set of each PLAM equal to the union of all care bits occuring in the input terms of its molecules.

Definition 3.10 Let B be a block of a molecule distribution. Then the *external input bit set* $EIB(B)$ of the PLAM of block B is given by

$$EIB(B) = \bigcup_{c \in B} B_0(c.i) \cup B_1(c.i).$$

Example 3.5 Consider the molecule distribution of example 3.3. For this molecule distribution, the external input bit sets of the PLAMs of both blocks are given by

$$\begin{aligned} EIB(B_1) &= \{x_1, x_2, x_4, x_7\}; \\ EIB(B_2) &= \{x_3, x_4, x_5, x_6\}. \end{aligned}$$

3.4.5 Processing of Next State Information

We have already noted that the PLAMs have to communicate to keep their current state information up-to-date. Now we will work out this thought precisely. Suppose that at a certain moment, the state information of the PLAMs is correct. We must achieve that at the next moment, the state information of all PLAMs is still correct. For each PLAM, next state information of machine M must be available with which the PLAM can determine what its next state is. Since each PLAM makes a transition to its wait state by default, it only has to take care of its next state if one of its current states is the next state of M. Thus, when the next state of M is s, all PLAMs that have s among their current states should be informed.

Before we answer the question how we inform PLAMs about the next state of machine M, we should know where this next state information is computed. According to the chosen distribution of tasks, the next state of M is computed by the molecules which contain a δ-operation. Consider such a molecule, which contains operation δ_s. When this molecule becomes active, it must inform some PLAMs about the fact that the next state of M equals s. The PLAMs that must be informed are the PLAMs that have s in their set of current states, and that don't compute the next state information theirselves. We make the choice here to inform all PLAMs that have s among their current states, regardless of whether or not they already compute the next state information themselves. This can lead to redundant communications, but it simplifies the model. Note that the PLAM that contains the molecule can be one of the PLAMs that have to be informed. For informing this PLAM, no

communications are needed. In order to inform the other PLAMs, messages need to be sended to them. The data which is sent in the message is the next state of the sequential machine M. So, an active molecule with operation δ_s sends state s to all PLAMs that contain current state s, except the PLAM to which the active molecule belongs.

From the previous paragraph we conclude that a state s of M is transmitted from a PLAM p to another PLAM p' if and only if p contains a molecule that contains operation δ_s, and p' contains a molecule that has s as current state. Define the next states of a PLAM as the states that occur in a δ-operation of at least one molecule of the PLAM. Then, we can say that given a PLAM p and another PLAM p', the states that have to be transmitted from the PLAM p to the PLAM p' are the states of M that are next state of p and current state of p'. This is formally described in the next two definitions. Both definitions are followed by an example.

Definition 3.11 Let B be a block of a molecule distribution. Then the *next state set* $NS(B)$ of the PLAM of block B is given by

$$NS(B) = \{s \mid \exists_{c \in B}\ \delta_s \in c.OP\}.$$

Example 3.6 Consider the molecule distribution of example 3.3. For this molecule distribution function, the next state sets of both blocks are given by

$$\begin{aligned} NS(B_1) &= \{s_1, s_3\}; \\ NS(B_2) &= \{s_1, s_2, s_3\}. \end{aligned}$$

Definition 3.12 Let π be a molecule distribution, and let B and B' be blocks of π. Then the *transmitted state set* $TS_{B-B'}(\pi)$ from the PLAM of block B to the PLAM of block B' is given by

$$TS_{B-B'}(\pi) = \begin{cases} NS(B) \cap CS(B') & \text{if } B \neq B'; \\ \emptyset & \text{otherwise.} \end{cases}$$

Example 3.7 Consider the molecule distribution of example 3.3. For this molecule distribution, the transmitted state sets are given by

$$\begin{aligned} TS_{B_1 \to B_1}(\pi) &= \emptyset; \\ TS_{B_1 \to B_2}(\pi) &= \{s_1\}; \\ TS_{B_2 \to B_1}(\pi) &= \{s_1, s_3\}; \\ TS_{B_2 \to B_2}(\pi) &= \emptyset. \end{aligned}$$

Knowing which communications have to take place, we can investigate how we should implement these communications. We have considered two alternative implementations.

The first alternative goes as follows. Since we know that the messages that are transmitted between PLAMs are states of the machine M, we can introduce a separate communication channel for each state s of M that is transmitted at least between one couple of PLAMs. This

communication channel can be implemented with one internal bit z_s of the network, which is one if the next state of M is s and zero otherwise. All PLAMs that have s among their next states have z_s as internal output bit. Each of these PLAMs makes z_s one if it wants to transmit state s, and zero otherwise. All PLAMs that have s among their current states have z_s as internal input bit. Each of these PLAMs makes a transition to its copy of state s if a one appears at its internal input bit z_s.

The second alternative goes as follows. Let p and p' be two different PLAMs, and let S be the set of states that have to be transmitted from p to p'. Then all states of S, and a special idle symbol, are coded on a minimal number of internal bits, and these bits become internal output bits of PLAM q and internal input bits of PLAM q'. By repeating this process for all pairs of different PLAMs, we can implement all needed communications.

The first alternative has the characteristic that each state s of M that has to be transmitted between PLAMs, receives an own internal bit z_s. Because this resembles one-hot encoding of the states of M, the first alternative will be called the *one-hot coding method.* The second alternative communication method has the characteristic that it minimally encodes the states communicated between each pair of PLAMs. Therefore, the second alternative will be called the *minimal coding method.*

Let us compare the one-hot coding method with the minimal coding method. The one-hot coding method is good in the following cases:

- when one PLAM p must transmit the same state s to many other PLAMs (in this case, p uses only one internal output bit z_s to transmit state s to many blocks);
- when many PLAMs transmit the same state s to one PLAM p (in this case, p uses only one internal input bit z_s to receive all states s coming from many other PLAMs).

The minimal coding method is good in the following cases:

- when one PLAM p must transmit a set S containing many states to PLAM p' and there are no other PLAMs to which states from S have to be transmitted (in this case, p only needs $\lceil^2\log(|S|+1)\rceil$ internal output bits to send the states of S to p');
- when one PLAM p' must transmit a set S containing many states to PLAM p and no other PLAMs have to transmit states of S to p (in this case, p only needs $\lceil^2\log(|S|+1)\rceil$ internal input bits to receive the states of S from p').

When we observe the advantages of both methods, we see that we should choose the best method depending on the communications that have to take place between PLAMs. It may also be a good idea to combine the two methods into a general method that can use the advantages of both methods. These suggestions will be left for future investigations, and

for the time being we will make use of the one-hot encoding method because of its simplicity.

The next three definitions precisely describe the chosen first alternative. Each definition is followed by an example.

Definition 3.13 Let π be a molecule distribution, and let B and B' be blocks of π. Then the *transmitted internal bit set* $TIB_{B\rightarrow B'}(\pi)$ from the PLAM of block B to the PLAM of block B' is given by

$$TIB_{B\rightarrow B'}(\pi) = \{z_s \mid s \in TS_{B\rightarrow B'}(\pi)\}.$$

Example 3.8 Consider the molecule distribution of example 3.3. For this molecule distribution, the transmitted internal bit sets are given by

$$\begin{aligned} TIB_{B_1-B_2}(\pi) &= \{z_{s_1}\}; \\ TIB_{B_2-B_1}(\pi) &= \{z_{s_1}, z_{s_3}\}. \end{aligned}$$

Definition 3.14 Let π be a molecule distribution. Then the *network internal bit set* $EB(\pi)$ of molecule distribution π is given by

$$EB(\pi) = \bigcup_{B,B'\in\pi} TIB_{B-B'}(\pi).$$

Example 3.9 Consider the molecule distribution of example 3.3. For this molecule distribution, the network internal bit set is given by

$$EB(\pi) = \{z_{s_1}, z_{s_3}\}.$$

Definition 3.15 Let π be a molecule distribution, and let B be a block of π. Then the *internal input bit set* $IIB_\pi(B)$ and the *internal output bit set* $IOB_\pi(B)$ of the PLAM of block B are given by

$$\begin{aligned} IIB_\pi(B) &= \bigcup_{B''\in\pi} TIB_{B''\rightarrow B}(\pi); \\ IOB_\pi(B) &= \bigcup_{B''\in\pi} TIB_{B-B''}(\pi). \end{aligned}$$

Example 3.10 Consider the molecule distribution of example 3.3. For this molecule distribution, the internal input and output bits of both blocks are given by

$$\begin{aligned} IIB_\pi(B_1) &= \{z_{s_1}, z_{s_3}\}; \\ IIB_\pi(B_2) &= \{z_{s_1}\}; \\ IOB_\pi(B_1) &= \{z_{s_1}\}; \\ IOB_\pi(B_2) &= \{z_{s_1}, z_{s_3}\}. \end{aligned}$$

3.4.6 Processing of Output Information

The output bits of M are computed as follows by the PLAMs of the network. The external output bit set of the network is made equal to the output bit set of the sequential machine M. An external output bit *ob* of the network becomes one if there exists a PLAM that has *ob*

among its set of external output bits, and this PLAM assigns one to *ob*. In all other cases, the value of *ob* is zero. So, a PLAM needs to have external output bit *ob* if and only if it contains a molecule *c* such that *c* contains operation λ_{ob}. The next definition and example work out these thoughts.

Definition 3.16 Let B be a block of a molecule distribution. Then the *external output bit set* $EOB(B)$ of the PLAM of block B is given by

$$EOB(B) = \{ob \mid \exists_{c \in B}\ \lambda_{ob} \in c.OP\}.$$

Example 3.11 Consider the molecule distribution of example 3.3. For this molecule distribution, the external output bits of the PLAMs of both blocks are given by

$$\begin{aligned} EOB(B_1) &= \{y_1, y_2, y_3, y_5, y_6, y_9\}; \\ EOB(B_2) &= \{y_2, y_4, y_5, y_7, y_8\}. \end{aligned}$$

3.4.7 Primary Term Lines

Now that we know how the state bits, input bits and output bits of each PLAM are chosen, and how the external input bits, the internal bits and the external output bits of the PLAM network are chosen, the only thing that remains is to define the term lines of each PLAM precisely. The term lines are separated into two groups: primary and secondary term lines. The primary term lines of a PLAM perform the tasks of the molecules in the corresponding block, and the secondary term lines are used to keep the state information of the PLAM up-to-date.

This section describes how the primary term lines are determined. The primary term lines of a PLAM are based on the molecules of the corresponding block. For every group of molecules that are active under the same condition, one term line is reserved. One group of molecules that are active under the same condition is represented with a pair (cs, i), where cs is the current state and i is the input term for which the molecules become active. Such a pair is called an activation condition, and the corresponding group of molecules is called the activated molecule set.

Definition 3.17 Let B be a block of a molecule distribution. Then the *activation condition set* $AC(B)$ of block B is given by

$$AC(B) = \{(cs, i) \mid \exists_{c \in B}\ c.cs = cs \wedge c.i = i\}.$$

Consider an activation condition (cs, i). The term line corresponding with the group of molecules which are active under that condition, is constructed as follows.

The current state and the input term of the term line are chosen such that the term line is active if and only if the current state of the prototype

machine is cs and the input minterm is covered by i. Thus, the current state of the term line must be the copy of state cs, and the input term of the term line must be the same as i for all care bits of i. Remember that the care bits of i are surely contained in the external input bit set of the PLAM. For all other input bits of the PLAM, the input term of the term line must be don't care.

The next state and the output term of the term line are chosen such that all operations of the group of molecules are performed when the term line becomes active.

Firstly, we describe how δ-operations are performed.

- When there are δ-operations among the operations that have to be performed, we choose one of these operations. Let the chosen δ-operation be $\delta_{s'}$. If s' is a current state of the PLAM, the next state of the term line is the encoded copy of the state s', otherwise it is the encoded wait state, which consists of all zeros. If the PLAM has an internal output bit $z_{s'}$, this bit $z_{s'}$ is made one in the output term of the term line. All other internal output bits have the value zero.
- If there are no δ-operations among the operations that have to be performed, the term line may not influence the next state. Therefore, we choose the next state of the term line equal to the all-0-term, and we choose the value 0 for all internal output bits of the term line. This choice can be explained as follows. The next state of a PLAM is determined by OR-ing the next states of all active term lines, and the value of each internal output bit is determined by OR-ing the values that are assigned to this internal output bit by all active term lines. Thus, an active term line that has an all-0-term as next state and zeros for the internal output bits, does not influence the next state of the PLAM nor does it influence the states that are transmitted to other PLAMs.

Secondly, we describe how λ-operations are performed. For each λ-operation λ_{ob} that has to be performed, the term line must make output bit ob equal to one. All output bits which must be made equal to 1, are external output bits of the PLAM to which the term line belongs, so the only thing which has to be done is making these external output bits one in the output term of the term line. All other external output bits of the term line are assigned the value 0, so that the value of these external output bits is not influenced by the term line.

The following two definitions formalize the previously explained term line choice.

Definition 3.18 Let π be a molecule distribution, let γ be a network state assignment of π, and let B be a block of π. Then the *primary term line function* of B, which is denoted with $ptl_{\pi,\gamma,B}$, is a function that maps activation conditions of B to term lines of the PLAM of B. This function is defined as follows. Let $(cs, i) \in AC(B)$ be an activation

condition, and let OP denote the set of operations that are activated when this activation condition is fulfilled:

$$OP = \{op \mid \exists_{c \in B}\ c.cs = cs \wedge c.i = i \wedge op \in c.OP\}.$$

If OP contains δ-operations, then choose one δ-operation of OP, and let $\delta_{s'}$ be this chosen δ-operation. Then, the term line to which the activation condition (cs, i) is mapped, is the term line l which is specified below, where $ob \in EOB(B)$, and $z_s \in IOB_\pi(B)$:

$$l.cs = \gamma(B)(cs);$$

$$l.i = [i]_{EIB(B)} \cdot t_\times(IIB_\pi(B));$$

$$l.ns = \begin{cases} \gamma(B)(s') & \text{if } OP \text{ has } \delta\text{-operations and } s' \in CS(B); \\ t_0(SB(B)) & \text{otherwise;} \end{cases}$$

$$l.o(ob) = \begin{cases} 1 & \text{if } \lambda_{ob} \in OP; \\ 0 & \text{otherwise;} \end{cases}$$

$$l.o(z_s) = \begin{cases} 1 & \text{if } OP \text{ has } \delta\text{-operations and } s = s'; \\ 0 & \text{otherwise.} \end{cases}$$

Definition 3.19 Let π be a molecule distribution, let γ be a network state assignment of π, and let B be a block of π. Then the *primary term line set* $PTL_{\pi,\gamma}(B)$ of the PLAM of block B is defined by

$$PTL_{\pi,\gamma}(B) = \{ptl_{\pi,\gamma,B}(cs, i) \mid (cs, i) \in AC(B)\}.$$

Example 3.12 Consider the molecule distribution π of example 3.3 and the network state assignment γ of example 3.4. Then the primary term lines for the PLAMs of block B_1 and block B_2 are presented in table 3.6 and 3.7, respectively.

Table 3.6 Primary term lines of the PLAM of block B_1

cs		i						ns		o					
b_1	b_2	x_1	x_2	x_4	x_7	z_{s_1}	z_{s_3}	b_1	b_2	y_1	y_2	y_3	y_5	y_6	y_9	z_{s_1}
0	1	0	1	1	–	–	–	0	1	1	0	0	0	0	0	1
0	1	1	1	1	–	–	–	1	0	0	0	0	0	0	0	0
0	1	–	0	1	–	–	–	1	0	0	0	1	1	0	0	0
1	0	–	–	–	1	–	–	0	1	0	1	0	0	1	0	1
1	0	–	–	–	0	–	–	1	0	0	0	0	0	0	1	0

Table 3.7 Primary term lines of the PLAM of block B_2

cs		i					ns		o						
b_1	b_2	x_3	x_4	x_5	x_6	z_{s_1}	b_1	b_2	y_2	y_4	y_5	y_7	y_8	z_{s_1}	z_{s_3}
0	1	–	1	–	–	–	0	0	0	0	0	1	0	0	0
0	1	0	0	–	–	–	0	1	1	0	1	0	0	1	0
0	1	1	0	–	–	–	1	0	1	0	0	1	0	0	0
1	0	–	–	1	1	–	0	1	0	0	0	0	1	1	0
1	0	–	–	0	0	–	1	0	1	1	0	0	1	0	0
1	0	–	–	0	1	–	0	0	0	1	0	1	0	0	1

3.4.8 Secondary Term Lines

The primary term lines are not the only term lines needed. As we have observed earlier, each PLAM must refresh its state information again and again. To keep the state information up-to-date, the PLAM must react on 1 values at its internal input bits. If an internal input bit z_s is 1, the PLAM must make a transition to its copy of s. Therefore, we introduce a secondary term line in the PLAM for each internal input bit z_s of the PLAM. This term line is made as follows:

- The current state of the term line is an all-×-term.
- The input term of the term line assigns 1 to input bit z_s and × to all other input bits.
- The next state of the term line is the coded state corresponding with the copy of s. The correspondence is given by the state assignment function.
- The output term is an all-0-term.

The following definition formally describes the construction of these secondary term lines. For each block, it gives a function that assigns a term line to each internal input bit of the block.

Definition 3.20 Let π be a molecule distribution, let γ be a network state assignment of π, and let B be a block of π. Then the *secondary term line function* of B, which is denoted with $stl_{\pi,\gamma,B}$, is a function that maps internal input bits of the PLAM of B to term lines of the PLAM of B. This function is defined as follows. Let $z_s \in IIB_\pi(B)$ be an internal input bit of the PLAM of B. The term line to which this internal input bit is mapped, is is the term line l which is specified

below, where $ib \in EIB(B) \cup IIB_\pi(B)$:

$$l.cs = t_\times(SB(B));$$

$$l.i(ib) = \begin{cases} 1 & \text{if } ib = z_s; \\ \times & \text{otherwise;} \end{cases}$$

$$l.ns = \gamma(B)(s);$$

$$l.o = t_0(EOB(B)).$$

Definition 3.21 Let π be a molecule distribution, let γ be a network state assignment of π, and let B be a block of π. Then the *secondary term line set* $STL_{\pi,\gamma}(B)$ of the PLAM of block B is defined by

$$STL_{\pi,\gamma}(B) = \{stl_{\pi,\gamma,B}(z_s) \mid z_s \in IIB_\pi(B)\}.$$

Example 3.13 Consider the molecule distribution π of example 3.3 and the network state assignment γ of example 3.4. Then the secondary term lines for the PLAMs of block B_1 and block B_2 are presented in table 3.8 and 3.9, respectively.

Table 3.8 Secondary term lines of the PLAM of block B_1

cs		i						ns		o						
b_1	b_2	x_1	x_2	x_4	x_7	z_{s_1}	z_{s_3}	b_1	b_2	y_1	y_2	y_3	y_5	y_6	y_9	z_{s_1}
–	–	–	–	–	–	1	–	0	1	0	0	0	0	0	0	0
–	–	–	–	–	–	–	1	1	0	0	0	0	0	0	0	0

Table 3.9 Secondary term lines of the PLAM of block B_2

cs		i					ns		o						
b_1	b_2	x_3	x_4	x_5	x_6	z_{s_1}	b_1	b_2	y_2	y_4	y_5	y_7	y_8	z_{s_1}	z_{s_3}
–	–	–	–	–	–	1	0	1	0	0	0	0	0	0	0

3.4.9 The Complete PLAM Network

Now we have completely described how a network of PLAMs must be constructed, provided that we dispose of a molecule distribution of the sequential machine M and a network state assignment for this molecule distribution. The following two definitions precisely define this PLAM network, in terms that have been introduced previously in the definitions of the previous subsection.

Definition 3.22 Consider a molecule distribution π, a network state assignment γ of π, and a block B of π. Then the *induced PLAM* of block B is the PLAM (IB, SB, OB, TL) given by

$$\begin{aligned} IB &= EIB(B) \cup IIB_\pi(B); \\ SB &= SB(B); \\ OB &= IOB_\pi(B) \cup EOB(B); \\ TL &= PTL_{\pi,\gamma}(B) \cup STL_{\pi,\gamma}(B). \end{aligned}$$

This induced PLAM will be denoted with $PLAM_{\pi,\gamma,B}$.

Example 3.14 Consider the molecule distribution π of example 3.3 and the state assignment function γ of example 3.4. Then the induced PLAMs of block B_1 and block B_2 are given by

$$\begin{aligned} PLAM_{\pi,\gamma,B_1}.IB &= \{x_1, x_2, x_4, x_7, z_{s_1}, z_{s_3}\}; \\ PLAM_{\pi,\gamma,B_1}.SB &= \{b_1, b_2\}; \\ PLAM_{\pi,\gamma,B_1}.OB &= \{y_1, y_2, y_3, y_5, y_6, y_9, z_{s_1}\}; \\ \\ PLAM_{\pi,\gamma,B_2}.IB &= \{x_3, x_4, x_5, x_6, z_{s_1}\}; \\ PLAM_{\pi,\gamma,B_2}.SB &= \{b_1, b_2\}; \\ PLAM_{\pi,\gamma,B_2}.OB &= \{y_2, y_4, y_5, y_7, y_8, z_{s_1}, z_{s_3}\}. \end{aligned}$$

The term lines of both PLAMs are given in table 3.10 and 3.11.

Table 3.10 Term lines of the PLAM induced by B_1

cs		*i*						*ns*		*o*						
b_1	b_2	x_1	x_2	x_4	x_7	z_{s_1}	z_{s_3}	b_1	b_2	y_1	y_2	y_3	y_5	y_6	y_9	z_{s_1}
0	1	0	1	1	–	–	–	0	1	1	0	0	0	0	0	1
0	1	1	1	1	–	–	–	1	0	0	0	0	0	0	0	0
0	1	–	0	1	–	–	–	1	0	0	0	1	1	0	0	0
1	0	–	–	–	1	–	–	0	1	0	1	0	0	1	0	1
1	0	–	–	–	0	–	–	1	0	0	0	0	0	0	1	0
–	–	–	–	–	–	1	–	0	1	0	0	0	0	0	0	0
–	–	–	–	–	–	–	1	1	0	0	0	0	0	0	0	0

Table 3.11 Term lines of the PLAM induced by B_2

cs		*i*					*ns*		*o*						
b_1	b_2	x_3	x_4	x_5	x_6	z_{s_1}	b_1	b_2	y_2	y_4	y_5	y_7	y_8	z_{s_1}	z_{s_3}
0	1	–	1	–	–	–	0	0	0	0	0	1	0	0	0
0	1	0	0	–	–	–	0	1	1	0	1	0	0	1	0
0	1	1	0	–	–	–	1	0	1	0	0	1	0	0	0
1	0	–	–	1	1	–	0	1	0	0	0	0	1	1	0
1	0	–	–	0	0	–	1	0	1	1	0	0	1	0	0
1	0	–	–	0	1	–	0	0	0	1	0	1	0	0	1
–	–	–	–	–	–	1	0	1	0	0	0	0	0	0	0

Definition 3.23 Consider a sequential machine M, a molecule distribution π of M, and a network state assignment γ of π. Then, the *induced PLAM network* is the PLAM network $n = (IB, EB, OB, P)$ given by

$$\begin{aligned} IB &= M.IB; \\ EB &= EB(\pi); \\ OB &= M.OB; \\ P &= \{PLAM_{\pi,\gamma,B} \mid B \in \pi\}. \end{aligned}$$

Example 3.15 Consider the molecule distribution π of example 3.3 and the state assignment function γ of example 3.4. Then the induced PLAM network is the PLAM network (IB, EB, OB, P) given by

$$\begin{aligned} IB &= \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}; \\ EB &= \{z_{s_1}, z_{s_3}\}; \\ OB &= \{y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9\}; \\ P &= \{PLAM_{\pi,\gamma,B_1}, PLAM_{\pi,\gamma,B_2}\}. \end{aligned}$$

The PLAMs $PLAM_{\pi,\gamma,B_1}$ and $PLAM_{\pi,\gamma,B_2}$ have already been presented in example 3.14. A schematic view of this induced PLAM network is given in figure 3.1.

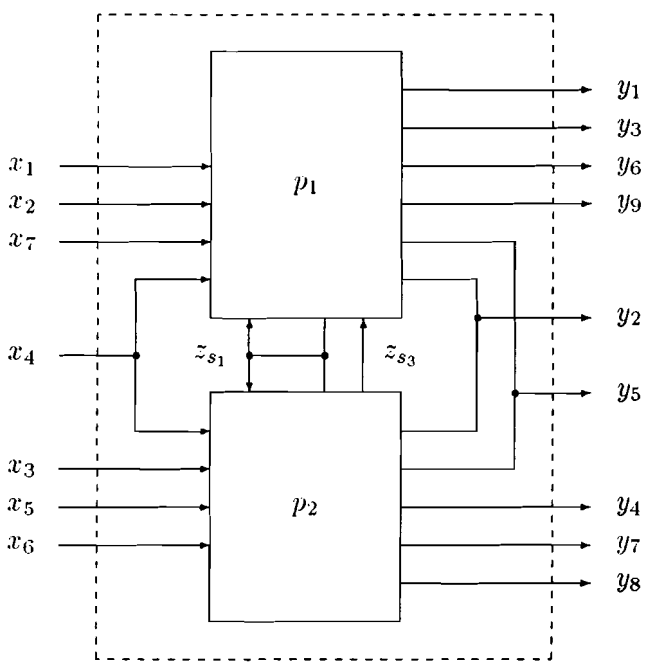

Figure 3.1 Induced PLAM network

3.4.10 Decomposition Theorem

It should be clear that the way in which a PLAM network is constructed,

based on a molecule distribution, is such that the following theorem holds.

Theorem 3.1 Consider a sequential machine M and a molecule distribution π of this sequential machine. Let n be a PLAM network that is induced by π. Then n is an output independent PLAM network decomposition of M.

The formal framework makes it possible to really prove this theorem, but we had not got the time to do so. It should be clear that the theorem can be proved by proving that n is an output independent PLAM network, and that the implemented machine $n.M$ of n realizes M, because there exists a state homomorphism $\phi : M.S \rightarrow S_{impl}(n)$, which is defined, for all $s \in M.S$ and $B \in \pi$, by

$$\phi(s)(PLAM_{\pi,\gamma,B}) = \begin{cases} \gamma(B)(s) & \text{if } s \in CS(B); \\ t_0(SB(B)) & \text{otherwise.} \end{cases}$$

The next example shows, for one sequential machine and molecule distribution, that theorem 3.1 holds.

Example 3.16 Consider the machine M of example 2.1. A molecule distribution π of this machine is given by $\pi = \{B_1, B_2\}$, where B_1 and B_2 are specified in table 3.12 and 3.13, respectively. A network state assignment γ of π is given in table 3.14. The PLAM network that is induced by π and γ is the network of example 2.9. In example 2.12, we have proved that this network is an output independent PLAM network decomposition of M.

Table 3.12 Molecules that are contained in block B_1

cs	i		OP
	x_1	x_2	
s_1	0	0	$\{\delta_{s_1}, \lambda_{y_1}\}$
	0	1	$\{\delta_{s_1}, \lambda_{y_2}\}$
	1	0	$\{\delta_{s_2}, \lambda_{y_1}\}$
s_2	0	0	$\{\delta_{s_1}\}$
	–	1	$\{\lambda_{y_1}\}$
	1	0	$\{\delta_{s_2}\}$

Table 3.13 Molecules that are contained in block B_2

cs	i		OP
	x_1	x_2	
s_2	–	0	$\{\lambda_{y_2}\}$
	–	1	$\{\delta_{s_3}, \lambda_{y_3}\}$
s_3	–	0	$\{\delta_{s_2}, \lambda_{y_2}\}$
	–	1	$\{\delta_{s_3}\}$

Table 3.14 Network state assignment γ

s	$\gamma(B_1)(s)$		$\gamma(B_2)(s)$	
	b_1	b_2	b_1	b_2
s_1	0	1		
s_2	1	0	0	1
s_3			1	0

4 Decomposition Method

The decomposition model which has been derived in the previous chapter, describes possible PLAM-network decompositions of a given prototype machine M in terms of task distributions. This chapter gives a method for finding a decomposition among these described decompositions, which satisfies the soft and hard constraints.

4.1 Overview of the Method

Because the decomposition task is a very complex, it is performed in a number of consecutive steps, each of which solves only a part of the problem. Figure 4.1 gives an overview of the decomposition method. The method consists of two top-down steps (macromolecule construction and macromolecule splitting), followed by two bottom-up steps (packaging and network construction). In the top down steps, a macromolecule cover is constructed. A macromolecule cover is a set of molecule sets, called macromolecules, such that the union of these molecule sets forms a molecule cover. In the bottom up steps, the macromolecules of the macromolecule cover are packaged into bigger molecule sets, which form the blocks of the molecule distribution which is used to construct the final PLAM network.

Macromolecule construction starts with comparing the transition table to the PLAM-size constraint, in order to determine which of the four parts of the PLAM-size constraint are most difficult to satisfy. Afterwards, macromolecules are constructed in such a way that each macromolecule minimally claims the PLAM spaces for which the constraint is most difficult to satisfy. This increases the probability that finally a PLAM network is obtained that satisfies the hard and soft constraints.

The probability of obtaining a PLAM network that satisfies the constraints is further increased by *macromolecule splitting*. In the macromolecule splitting step, macromolecules that surely don't fit in one PLAM, are splitted into smaller macromolecules, which claim a smaller part of the PLAM spaces and thus have greater probability of fitting in one PLAM.

Packaging consists of constructing a molecule distribution by placing macromolecules into blocks, such that the constraints are satisfied by the PLAM network that is induced by this molecule distribution. Each block can be seen as a packet in which molecules can be placed such that the corresponding PLAM does not violate the PLAM-size constraint.

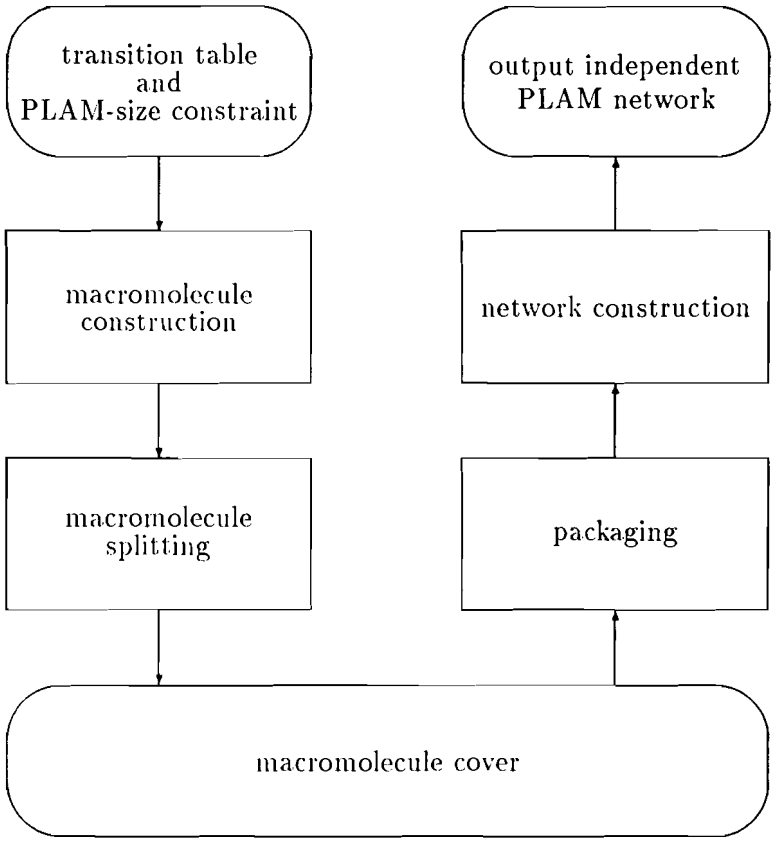

Figure 4.1 Overview of the decomposition method

The number of blocks (or, equivalently, the number of packages) which is needed in order to obtain a molecule distribution for which the induced PLAM network satisfies all constraints, is predicted before the packaging process begins. During the first packaging it is tried to put all macromolecules in this number of blocks. If the first packaging succeeds, the number of blocks is repeatedly decremented until the number of blocks has become too small to package all macromolecules. If the first packaging fails, the number of blocks is repeatedly incremented until the number of blocks has become large enough to contain all macromolecules. This increment/decrement process can be seen as a search for the decomposition with the least number of PLAMs. The minimization of the number of connections between the PLAMs is done during the packaging algorithm.

The packaging algorithm will not be explained further in this report, because there are packaging methods available, for example the methods described in [4] and [5], that are efficient and effective, and that can be adapted for this specific packaging problem.

The *network construction* step is straightforward. It consists of the translation of a molecule distribution into its induced PLAM network, that has been described extensively in the previous chapter.

4.2 Modeling Hard and Soft Constraints

Because we want to make a molecule distribution that leads to a PLAM network that satisfies the hard and soft constraints, we must have a good view of the relation between molecule distributions and the hard and soft constraints on their induced PLAM network. This relation is described in this section.

First, we will look at the hard constraint, which is given by the PLAM-size constraint (see definition 2.22). Every PLAM in the network has only a limited number of input bits, state bits, output bits and term lines. The limited number of state bits can also be expressed as a limited number of states, taking into account that one of these states must be reserved for the wait state. We introduce the concepts of PLAM spaces and positions in the next definition.

Definition 4.1 The set of input bits, states, output bits and term lines of a PLAM are respectively called the *input bit space*, *state space*, *output bit space*, and *term line space* of the PLAM. The elements of these spaces are called *positions*. So, the input bit space contains *input bit positions*, the state space contains *state positions*, the output bit space contains *output bit positions* and the term line space contains *term line positions*.

The cardinality of the four PLAM spaces can be expressed in terms of the PLAM-size constraint c: the input bit space contains $c.nr_{IB}$ positions, the state space contains $2^{c.nr_{SB}} - 1$ positions, the output bit space contains $c.nr_{OB}$ positions, and the term line space contains $c.nr_{TL}$ positions.

When molecules are placed in a PLAM, they claim positions in the PLAM spaces. The following definition introduces the terminology that will be used to describe the effect of molecules on the spaces of the PLAM in which they are placed.

Definition 4.2 A set of molecules that is placed in a certain PLAM, claims positions of this PLAM. A *claim* for a position is characterized by the space to which the position belongs, and the demanded value for this position. We distinguish 7 types of claims. These types are summarized in table 4.1.

The next two definitions define the claims of a set of molecules on the PLAM in which they are placed. The first definition handles claims that are independent of the whole molecule cover and the second definition handles claims that are dependent of the whole molecule cover.

Definition 4.3 Let B be a set of molecules. Then the *set of external input bit claims*, the *set of state claims*, the *set of external output bit claims* and the *set of primary term line claims* of the molecules of B are

Table 4.1 Overview of the seven claim types

claim type	space	claimed value
External Input Bit (EIB) claim	input bit space	input bit of the prototype machine
Internal Input Bit (IIB) claim	input bit space	z_s, where s is a state of the prototype machine
State (S) claim	state space	state of the prototype machine
External Output Bit (EOB) claim	output bit space	output bit of the prototype machine
Internal Output Bit (IOB) claim	output bit space	z_s, where s is a state of the prototype machine
Primary Term Line (PTL) claim	term line space	primary term line for activation condition (cs, i), where cs is a state of the prototype machine and i is an input term of the prototype machine
Secondary Term Line (STL) claim	term line space	secondary term line for a bit z_s, where s is a state of the prototype machine

respectively given by

$$\begin{aligned}
\mathcal{C}_{EIB}(B) &= \{ib \mid \exists_{c\in B}\ c.i(ib) \neq \times\};\\
\mathcal{C}_{S}(B) &= \{s \mid \exists_{c\in B}\ c.cs = s\};\\
\mathcal{C}_{EOB}(B) &= \{ob \mid \exists_{c\in B}\ c.o(ob) = 1\};\\
\mathcal{C}_{PTL}(B) &= \{(cs, i) \mid \exists_{c\in B}\ c.cs = cs \wedge c.i = i\}.
\end{aligned}$$

These claims are called *independent*.

Example 4.1 Consider the molecule distribution of example 3.3. For the blocks B_1 and B_2 of this molecule distribution, the external input bit claims, state claims, external output bit claims and primary term line claims are given below.

$$\begin{aligned}
\mathcal{C}_{EIB}(B_1) &= \{x_1, x_2, x_4, x_7\};\\
\mathcal{C}_{S}(B_1) &= \{s_1, s_3\};\\
\mathcal{C}_{EOB}(B_1) &= \{y_1, y_2, y_3, y_5, y_6, y_9\};\\
\mathcal{C}_{PTL}(B_1) &= \{(s_1, \overline{x_1}\, x_2\, x_4), (s_1, x_1\, x_2\, x_4), (s_1, \overline{x_2}\, x_4),\\
&\quad (s_3, x_7), (s_3, \overline{x_7})\};
\end{aligned}$$

$$\begin{aligned}
\mathcal{C}_{EIB}(B_2) &= \{x_3, x_4, x_5, x_6\};\\
\mathcal{C}_{S}(B_2) &= \{s_1, s_2\};\\
\mathcal{C}_{EOB}(B_2) &= \{y_2, y_4, y_5, y_7, y_8\};\\
\mathcal{C}_{PTL}(B_2) &= \{(s_1, x_4), (s_1, \overline{x_3}\,\overline{x_4}), (s_1, x_3\,\overline{x_4}), (s_2, x_5\, x_6),\\
&\quad (s_2, \overline{x_5}\,\overline{x_6}), (s_2, \overline{x_5}\, x_6)\}.
\end{aligned}$$

Definition 4.4 Let C be a molecule cover, and let B be a subset of C. Then the *set of internal input bit claims*, the *set of internal output bit claims* and the *set of secondary term line claims* of the molecules of B

in molecule cover C are respectively given by

$$\begin{aligned}
\mathcal{C}_{IIB}(B,C) &= \{z_s \mid \exists_{c\in B}\ c.cs = s \wedge \exists_{c\in C\setminus B}\ \delta_s \in c.OP\};\\
\mathcal{C}_{IOB}(B,C) &= \{z_s \mid \exists_{c\in B}\ \delta_s \in c.OP \wedge \exists_{c\in C\setminus B}\ c.cs = s\};\\
\mathcal{C}_{STL}(B,C) &= \mathcal{C}_{IIB}(B,C).
\end{aligned}$$

These claims are called *dependent.*

Example 4.2 Consider the molecule distribution of example 3.3. For the blocks B_1 and B_2 of this molecule distribution, the internal input bit claims, internal output bit claims and secondary term line claims are given below.

$$\begin{aligned}
\mathcal{C}_{IIB}(B_1, B_1 \cup B_2) &= \{z_{s_1}, z_{s_3}\};\\
\mathcal{C}_{IOB}(B_1, B_1 \cup B_2) &= \{z_{s_1}\};\\
\mathcal{C}_{STL}(B_1, B_1 \cup B_2) &= \{z_{s_1}, z_{s_3}\};\\
\\
\mathcal{C}_{IIB}(B_2, B_1 \cup B_2) &= \{z_{s_1}\};\\
\mathcal{C}_{IOB}(B_2, B_1 \cup B_2) &= \{z_{s_1}, z_{s_3}\};\\
\mathcal{C}_{STL}(B_2, B_1 \cup B_2) &= \{z_{s_1}\}.
\end{aligned}$$

For checking whether a certain molecule distribution leads to a PLAM network that satisfies the hard constraints, we can look at the claims of each block of the molecule distribution, and determine whether these claims cause a violation of the PLAM-size constraint.

Definition 4.5 Let C be a molecule cover, let B be a subset of C, and let c be a PLAM-size constraint. Then B *violates the PLAM-size constraint* c if one of the following conditions is true:

$$\begin{aligned}
c.nr_{IB} &< |\mathcal{C}_{EIB}(B)| + |\mathcal{C}_{IIB}(B,C)|;\\
2^{c.nr_{SB}} - 1 &< |\mathcal{C}_S(B)|;\\
c.nr_{OB} &< |\mathcal{C}_{EOB}(B)| + |\mathcal{C}_{IOB}(B,C)|;\\
c.nr_{TL} &< |\mathcal{C}_{PTL}(B)| + |\mathcal{C}_{STL}(B,C)|.
\end{aligned}$$

Definition 4.6 Let π be a molecule distribution for a molecule cover C, and let c be a PLAM-size constraint. Then π *violates the PLAM-size constraint* c iff there is a block of π that violates the PLAM-size constraint c.

It should be clear that an induced PLAM network of a molecule distribution violates the PLAM-size constraint iff the molecule distribution violates the PLAM-size constraint.

Example 4.3 Consider the molecule distribution π of example 3.3, and consider the constraint c given by

$$\begin{aligned}
c.nr_{IB} &= 6;\\
c.nr_{SB} &= 2;\\
c.nr_{OB} &= 7;\\
c.nr_{TL} &= 7
\end{aligned}$$

For each block of molecule distribution π, a schematic view of the induced PLAM is given in figure 4.2. The schematic PLAM on the left side represents the induced PLAM of block B_1, and the one on the right side represents the induced PLAM of block B_2. The positions in the input bit space of the PLAMs are the ingoing arrows, the positions in the state space are the circles, the positions in the output bit space are the outgoing arrows and the positions in the term line space are the rectangles. In example 4.1 and example 4.2, the claims of blocks B_1 and B_2 have been given. The honouring of these claims is shown in the schematic PLAMs by placing the values that are claimed in a certain space at positions in this space. It can easily be seen that PLAM-size constraint c is not violated by B_1 nor by B_2. Thus, a PLAM network that is induced by molecule distribution π surely satisfies the PLAM-size constraints c.

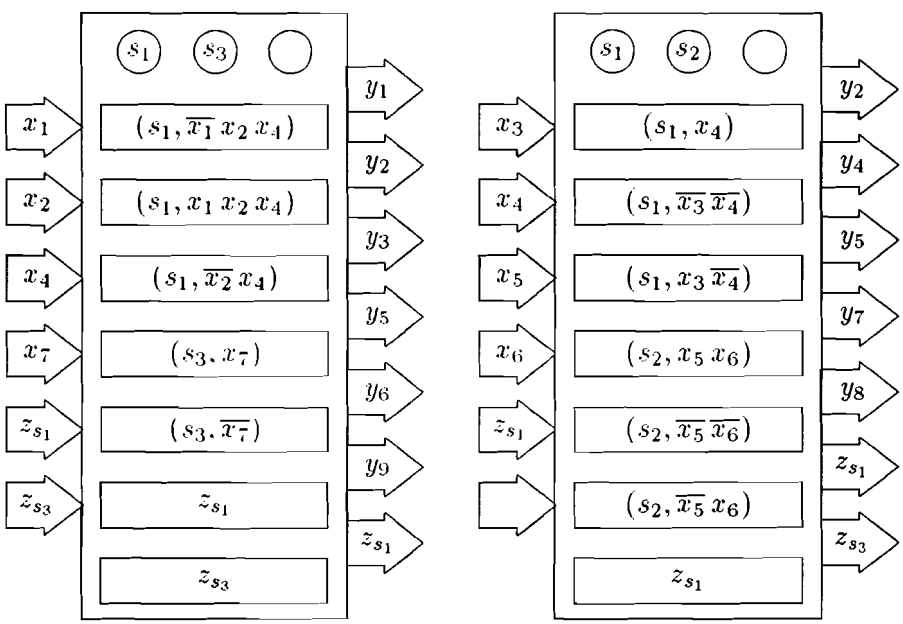

Figure 4.2 Claims of the blocks of an example molecule distribution

It is also possible to relate the soft constraints directly to a molecule distribution, using claims. The number of PLAMs in the PLAM network simply equals the number of blocks of the molecule distribution. The number of connections in a PLAM network that is induced by molecule distribution π is equal to

$$\Sigma_{B \in \pi} |\mathcal{C}_{EIB}(B)| + |\mathcal{C}_{IIB}(B, C)| + |\mathcal{C}_{EOB}(B)| + |\mathcal{C}_{IOB}(B, C)|, \qquad (2)$$

where $C = \bigcup \pi$.

Example 4.4 Consider the molecule distribution of example 3.3. Filling

in the claim sets, that have been determined in example 4.1 and 4.2, in expression (2) results in a number of connections equal to 25.

4.3 Macromolecule Construction

This section explains precisely the first step of the decomposition method.

Definition 4.7 Let M be a sequential machine. A *macromolecule* of M is a set containing molecules of M. A set A of macromolecules of M is a *macromolecule cover* of M iff $\bigcup A$ is a molecule cover of M.

The objective of macromolecule cover construction is to find a macromolecule cover which maximizes the probability that the final PLAM network satisfies the hard and soft constraints. This is a difficult task, because it is hard to predict the effect of a certain macromolecule cover on the final PLAM network.

We introduce four different ways of constructing a macromolecule cover. Which of the four is chosen depends on the part of the PLAM-size constraint that seems most difficult to satisfy. All four ways of macromolecule cover construction are based on two-level minimization of a task table, which is derived from the transition table. Therefore, we first explain what has to be precisely understood by two-level minimization and task tables, and afterwards we explain how the macromolecule construction method works.

4.3.1 Two-Level Minimization

Let B and B' be sets of bits. Then, a multiple output binary function from B to B' is a function that maps complete assignments of the bits of B to possibly incomplete assignments of the bits of B'. This can be formalized as follows.

Definition 4.8 Let B, B' be two sets of bits. Then a *multiple output binary function* f from B to B' is a function $f : \mathcal{T}_0(B) \rightarrow \mathcal{T}(B')$. f is called *completely specified* iff $f(t)$ is a minterm for all $t \in \mathcal{T}_0(B)$.

Example 4.5 Let $B = \{x_1, x_2\}$ and $B' = \{y_1, y_2\}$ be two sets of bits. Then the function $f : \mathcal{T}_0(B) \rightarrow \mathcal{T}(B')$, which is defined by table 4.2, is a multiple output binary function from B to B'. For example, this function expresses that, when x_1 is 0 and x_2 is 0, y_2 is assigned the value 1, and nothing is said about the value of y_1.

The next definition introduces the covering concept for multiple output binary functions, based on the previously defined covering of terms.

Table 4.2 Multiple output binary function f from B to B'

t		$f(t)$	
x_1	x_2	y_1	y_2
0	0	–	1
0	1	0	0
1	0	0	1
1	1	1	–

Definition 4.9 Let B, B' be sets of bits, and let f, f' be two multiple output binary functions from B to B'. Then, f *covers* f' iff

$$\forall_{t \in \mathcal{T}_0(B)}\ f(t) \geq f'(t).$$

Example 4.6 Let B, B' be the sets of bits of example 4.5. Let $g : \mathcal{T}_0(B) \rightarrow \mathcal{T}(B')$ be the multiple output binary function given by table 4.3. Then g is covered by the multiple output binary function f of example 4.5.

Table 4.3 Multiple output binary function g from B to B'

t		$f(t)$	
x_1	x_2	y_1	y_2
0	0	0	1
0	1	0	0
1	0	0	1
1	1	1	1

When we aim at realizing a multiple output boolean function with a PLA, we express the function with a PLA table. The definition of a PLA table, along with the definition of the function that is described by the PLA table, are given now.

Definition 4.10 A *PLA table* T with input bit set B and output bit set B' is a subset of $\mathcal{T}(B) \times \mathcal{T}(B')$. T is called *completely specified* if t' is a minterm for all $(t, t') \in T$. A PLA table T has *separated outputs* if

$$\forall_{(t,t') \in T}\ |B_1(t')| \leq 1.$$

Definition 4.11 Let T be a PLA table with input bit set B and output bit set B'. Then the *described function* of T, which is denoted by f_T, is a multiple output binary function from B to B'. For any $t \in \mathcal{T}_0(B)$ and $b \in B'$, $f_T(t)(b)$ is equal to

$$\begin{cases} 1 & \text{if } \exists_{(t_1,t_2) \in T}\ t \leq t_1 \wedge t_2(b) = 1; \\ \times & \text{if } \exists_{(t_1,t_2) \in T}\ t \leq t_1 \wedge t_2(b) = \times \wedge \forall_{(t_1,t_2) \in T}\ t \leq t_1 \Rightarrow t_2(b) \neq 1; \\ 0 & \text{otherwise.} \end{cases}$$

Example 4.7 In table 4.4, three PLA tables are given, labeled A, B and C. The described function of A and B is the multiple output binary function f of table 4.2, and the described function of C is the multiple output binary function g of table 4.3.

Table 4.4 PLA tables

A				B				C			
t		t'		t		t'		t		t'	
x_1	x_2	y_1	y_2	x_1	x_2	y_1	y_2	x_1	x_2	y_1	y_2
0	0	–	1	–	0	0	1	0	0	0	1
1	0	0	1	0	0	–	0	1	–	0	1
1	1	1	–	1	1	1	–	1	1	1	0

The covering concept for multiple output binary functions can be translated to the covering concept for PLA tables.

Definition 4.12 Let T and T' be PLA tables. Then T *covers* T' iff f_T covers $f_{T'}$.

Example 4.8 Let A, B and C be the PLA tables that are presented in table 4.4. Then, A covers C and B covers C.

It should be clear that, when a multiple output boolean function f from B to B' can be described by a PLA table T, there exists a PLA with $|B|$ input bits, $|B'|$ output bits and $|T|$ terms such that this PLA realizes the function f.

There are many tools available that can find a minimal PLA table that is covered by a given PLA table. The minimality criterion depends on the type of minimization. The following list summarizes the most important types of PLA table minimization, and gives their minimality criteria.

- *Multiple output row minimization*
  The minimized PLA table is a completely specified PLA table. The number of rows in the minimized table is as small as possible.
- *Single output row minimization*
  The minimized PLA table is a completely specified PLA table with separated outputs. The minimized table satisfies the condition that for each output bit, the number of rows that assign 1 to this output bit is as small as possible.
- *Multiple output support minimization*
  The minimized PLA table is a completely specified PLA table. The number of care input bits in the minimized table is as small as possible.
- *Single output support minimization*
  The minimized PLA table is a completely specified PLA table with separated outputs. The minimized table satisfies the condition that for each output bit, the number of care input bits for this

output bit is as small as possible.

4.3.2 Task Tables

A task table is a PLA table that is derived from the transition table, and it is used in order to construct a molecule cover. The following definition describes precisely how a task table is defined.

Definition 4.13 Let T be a transition table. The *task table* of T is a PLA table with input bit set B and output bit set B', where B and B' are given by

$$\begin{aligned} B &= \{x_s \mid s \in T.S\} \cup \{x_{ib} \mid ib \in T.IB\}; \\ B' &= \{y_{\delta_s} \mid s \in T.S\} \cup \{y_{\lambda_{ob}} \mid ob \in T.OB\}. \end{aligned}$$

For each horizontal $h \in T.H$, the task table contains a row (t, t'), where t and t' are defined as follows, for $s \in T.S$, $ib \in T.IB$ and $ob \in T.OB$:

$$t(x_s) = \begin{cases} 0 & \text{if } h.cs \neq s; \\ 1 & \text{if } h.cs = s; \end{cases}$$

$$t(x_{ib}) = h.i(ib);$$

$$t'(y_{\delta_s}) = \begin{cases} 0 & \text{if } h.ns \neq s; \\ 1 & \text{if } h.ns = s; \end{cases}$$

$$t'(y_{\lambda_{ob}}) = h.o(ob).$$

For each $(cs, x) \in T.S \times \mathcal{T}_0(T.IB)$ such that there is no horizontal $h \in H$ with $h.cs = cs$ and $x \leq h.i$, the task table contains a row (t, t') which is defined as follows, for $s \in T.S$, $ib \in T.IB$ and $ob \in T.OB$:

$$t(x_s) = \begin{cases} 0 & \text{if } cs \neq s; \\ 1 & \text{if } cs = s; \end{cases}$$

$$t(x_{ib}) = x(ib);$$

$$t'(y_{\delta_s}) = \times;$$

$$t'(y_{\lambda_{ob}}) = \times.$$

Example 4.9 Consider the transition table T, which has been presented in example 2.5. Then the task table of T is given in table 4.5.

If we construct a completely specified PLA table which is covered by the task table, for example by two-level minimizing the task table, this completely specified PLA table is called a task cover table.

Definition 4.14 Let T be a transition table. A *task cover table* of T

Table 4.5 Task table of transition table T

t					t'					
x_{s_1}	x_{s_2}	x_{s_3}	x_{x_1}	x_{x_2}	$y_{\delta_{s_1}}$	$y_{\delta_{s_2}}$	$y_{\delta_{s_3}}$	$y_{\lambda_{y_1}}$	$y_{\lambda_{y_2}}$	$y_{\lambda_{y_3}}$
1	0	0	0	0	1	0	0	1	0	0
1	0	0	0	1	1	0	0	–	1	0
1	0	0	1	0	0	1	0	1	0	0
1	0	0	1	1	–	–	–	–	–	–
0	1	0	0	0	1	0	0	–	–	0
0	1	0	–	1	0	0	1	1	0	1
0	1	0	1	0	0	1	0	0	1	0
0	0	1	–	0	0	1	0	0	1	0
0	0	1	–	1	0	0	1	–	0	0

is a completely specified PLA table that is covered by the task table of T.

Example 4.10 Reconsider the transition table T of example 4.9. A task table cover of this transition table is given in table 4.6.

Table 4.6 Task cover table of transition table T

t					t'					
x_{s_1}	x_{s_2}	x_{s_3}	x_{x_1}	x_{x_2}	$y_{\delta_{s_1}}$	$y_{\delta_{s_2}}$	$y_{\delta_{s_3}}$	$y_{\lambda_{y_1}}$	$y_{\lambda_{y_2}}$	$y_{\lambda_{y_3}}$
1	0	0	0	0	1	0	0	1	0	0
1	0	0	0	1	1	0	0	0	1	0
1	0	0	1	0	0	1	0	1	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	0	–	1	0	0	1	1	0	1
0	1	0	1	0	0	1	0	0	1	0
0	0	1	–	0	0	1	0	0	1	0
0	0	1	–	1	0	0	1	0	0	0

There is a simple mapping β from task cover tables of a transition table to molecule covers of the described machine of this transition table. This mapping is defined as follows. Let T be a transition table, and let K be a task cover table of T. Then $\beta(K)$ contains a molecule for a row (t, t') of K, if t satisfies the condition that there exists an $s \in T.S$ such that $t(x_s) = 1$, and $t(x_{s'}) = 0$ for all $s' \in T.S \setminus \{s\}$. The molecule that corresponds with this row (t, t') is the molecule c which is specified below, for $ib \in T.IB$:

$$
\begin{aligned}
cs &= s; \\
i(ib) &= t(x_{ib}); \\
OP &= \{op \mid y_{op} \in B_1(t')\}.
\end{aligned}
$$

It can be shown that this mapping is correct, and that each molecule cover of the prototype machine is the image of at least one task cover

table.

Example 4.11 Let K be the task cover table of example 4.10. This task table cover K is mapped to the molecule cover which is presented as table 4.7.

Table 4.7 Molecule cover corresponding with task cover table

cs	i		OP
	x_1	x_2	
s_1	0	0	$\{\delta_{s_1}, \lambda_{y_1}\}$
	0	1	$\{\delta_{s_1}, \lambda_{y_2}\}$
	1	0	$\{\delta_{s_2}, \lambda_{y_1}\}$
s_2	0	0	$\{\delta_{s_1}\}$
	–	1	$\{\delta_{s_3}, \lambda_{y_1}, \lambda_{y_3}\}$
	1	0	$\{\delta_{s_2}, \lambda_{y_2}\}$
s_3	–	0	$\{\delta_{s_2}, \lambda_{y_2}\}$
	–	1	$\{\delta_{s_3}\}$

4.3.3 Macromolecule Cover Construction Based on Two-Level Minimization of Task Tables

Depending on the specific characteristics of the problem instance, which consists of a transition table of the prototype machine together with a PLAM-size constraint, we choose one of four possible ways of constructing macromolecules. The choice is based on an estimation of the relative difficulty of satisfying each part of the PLAM-size constraint. This estimation is made by comparing the transition table with the PLAM-size constraint in four dimensions: the input bit dimension, the state bit dimension, the output bit dimension and the term line dimension.

Definition 4.15 Let T be a transition table, and let c be a PLAM-size constraint. Then the corresponding *constraint exceeding factors* for the input bit, state, output bit and term line space are denoted with $\varepsilon_{IB}(T,c), \varepsilon_S(T,c), \varepsilon_{OB}(T,c), \varepsilon_{TL}(T,c)$ respectively, and defined by

$$\begin{aligned}
\varepsilon_{IB}(T,c) &= |T.IB|/c.nr_{IB};\\
\varepsilon_S(T,c) &= |T.S|/(2^{c.nr_{SB}}-1);\\
\varepsilon_{OB}(T,c) &= |T.OB|/c.nr_{OB};\\
\varepsilon_{TL}(T,c) &= |T.H|/c.nr_{TL}.
\end{aligned}$$

The macromolecules are constructed as follows. First, we determine which of the four constraint exceeding factors is the largest. When this is the factor for the input bit, the state or the term line space, we choose for *state based macromolecule construction*, otherwise we choose for *operation based macromolecule construction*. An explanation of these

two possible ways of macromolecule construction is given below.

- *state based macromolecule construction* means that the rows of the task table are partitioned, such that rows with the same one-hot encoded current state belong to the same block. Then, a separate multiple output minimization is performed on each block of the partition, and afterwards the resulting rows for each block are merged into a complete minimized task table, which forms a task cover table. For each block of this task table cover, a macromolecule is made that contains precisely the molecules that correspond with the rows of this block. **State based macromolecule construction guarantees that each macromolecule claims precisely one state, at most one internal input bit, and at most one secondary term line.**
- *operation based macromolecule construction* means that the task table is minimized in its entirety, but using single output minimization. For each output bit of the resulting task cover table, a macromolecule is made that contains exactly the molecules which correspond to rows that assign one to this output bit. **Operation based macromolecule construction guarantees that each macromolecule claims at most one external or internal output bit.**

For state based as well as for operation based macromolecule construction, we can still choose for *row based macromolecule construction* or *support based macromolecule construction.* For row based macromolecule construction, the two-level minimization step minimizes the number of rows, and for support based construction, the two-level minimization step minimizes the number of input bits that are care. **Row based macromolecule construction results in a minimal number of primary term line claims for each macromolecule, and support based macromolecule construction results in a minimal number of external input bit claims for each macromolecule.**

Now we have 4 possible ways of macromolecule construction, because we must choose between state and operation based, and between row and support based. The effects of each alternative on the claims made by macromolecules have been given in the bold-faced parts of the previous paragraphs. In table 4.8, the alternatives are compared with respect to the claims that are made by the macromolecules. When an alternative results in few claims of a certain type, a bullet is placed.

The choice between the alternatives is made with the following basic idea in mind. When a certain PLAM space probably causes the most difficulties, we try to minimize for each macromolecule the number of its claims in the problem-causing spaces. The problem causing spaces are the spaces for which the constraint exceeding factor is high. Table 4.9 shows which alternative should be chosen, depending on the values of the constraint exceeding factors. This table is made using table 4.8. For example, when the constraint exceeding factor for the state space is the largest constraint exceeding factor, we try to minimize the state

Table 4.8 Comparison of alternatives with respect to claims

claim type	alternative			
	state row	state support	operation row	operation support
EIB		•		•
IIB	•	•		
S	•	•		
EOB			•	•
IOB			•	•
PTL	•		•	
STL	•	•		

claims for each macromolecule. Table 4.8 shows that in this case, state and row based, or state and support based macromolecule construction should be chosen. For making the choice between the remaining two alternatives, we look at the second to largest constraint exceeding factor. From table 4.8, it follows that it is better to choose for state and row based macromolecule construction if the constraint exceeding factor for the term line space is larger than that of the input bit space, and that it is better to choose for state and support based macromolecule construction otherwise.

Table 4.9 Choice of macromolecule construction method

largest constraint exceeding factor	macromolecule construction method
$\varepsilon_{IB}(T,c)$	state & support
$\varepsilon_S(T,c)$	state & support if $\varepsilon_{IB}(T,c) \geq \varepsilon_{TL}(T,c)$ state & row if $\varepsilon_{TL}(T,c) \geq \varepsilon_{IB}(T,c)$
$\varepsilon_{OB}(T,c)$	operation & support if $\varepsilon_{IB}(T,c) \geq \varepsilon_{TL}(T,c)$ operation & row if $\varepsilon_{TL}(T,c) \geq \varepsilon_{IB}(T,c)$
$\varepsilon_{TL}(T,c)$	state & row

Example 4.12 Let the prototype machine be given by the transition table T, which has been presented in example 2.5. The task table of T is given in table 4.5. Let the constraint c be given by

$$
\begin{aligned}
c.nr_{IB} &= 2;\\
c.nr_{SB} &= 2;\\
c.nr_{OB} &= 4;\\
c.nr_{TL} &= 4.
\end{aligned}
$$

Then, the constraint exceeding factors are as follows:

$$\begin{aligned}
\varepsilon_{IB}(T,c) &= 2/2 = 1; \\
\varepsilon_{S}(T,c) &= 3/3 = 1; \\
\varepsilon_{OB}(T,c) &= 3/4 = 0.75; \\
\varepsilon_{TL}(T,c) &= 8/4 = 2.
\end{aligned}$$

In this case we choose for state and row based macromolecule construction (see table 4.9). The resulting task cover table is given in table 4.10 and the resulting macromolecule cover is presented in table 4.11.

Now consider another constraint:

$$\begin{aligned}
c.nr_{IB} &= 4; \\
c.nr_{SB} &= 2; \\
c.nr_{OB} &= 1; \\
c.nr_{TL} &= 6.
\end{aligned}$$

Now the constraint exceeding factors are given by

$$\begin{aligned}
\varepsilon_{IB}(T,c) &= 2/4 = 0.5; \\
\varepsilon_{S}(T,c) &= 3/3 = 1; \\
\varepsilon_{OB}(T,c) &= 3/1 = 3; \\
\varepsilon_{TL}(T,c) &= 8/6 \approx 1.33.
\end{aligned}$$

In this case we choose for operation and row based macromolecule construction (see table 4.9). The resulting task cover table is given in table 4.12 and the resulting macromolecule cover is presented in table 4.13.

Table 4.10 Task cover table for constraint $(2, 2, 4, 4)$

t					t'					
x_{s_1}	x_{s_2}	x_{s_3}	x_{x_1}	x_{x_2}	$y_{\delta_{s_1}}$	$y_{\delta_{s_2}}$	$y_{\delta_{s_3}}$	$y_{\lambda_{y_1}}$	$y_{\lambda_{y_2}}$	$y_{\lambda_{y_3}}$
1	0	0	0	–	1	0	0	1	0	0
1	0	0	–	1	0	0	0	0	1	0
1	0	0	1	–	0	1	0	1	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	0	–	1	0	0	1	1	0	1
0	1	0	1	0	0	1	0	0	1	0
0	0	1	–	0	0	1	0	0	1	0
0	0	1	–	1	0	0	1	0	0	0

Table 4.11 Macromolecule cover for constraint $(2, 2, 4, 4)$

cs	i		OP
	x_1	x_2	
s_1	0	–	$\{\delta_{s_1}, \lambda_{y_1}\}$
	–	1	$\{\lambda_{y_2}\}$
	1	–	$\{\delta_{s_2}, \lambda_{y_1}\}$
s_2	0	0	$\{\delta_{s_1}\}$
	–	1	$\{\delta_{s_3}, \lambda_{y_1}, \lambda_{y_3}\}$
	1	0	$\{\delta_{s_2}, \lambda_{y_2}\}$
s_3	–	0	$\{\delta_{s_2}, \lambda_{y_2}\}$
	–	1	$\{\delta_{s_3}\}$

Table 4.12 Task cover table for constraint $(4, 2, 1, 6)$

t					t'					
x_{s_1}	x_{s_2}	x_{s_3}	x_{x_1}	x_{x_2}	$y_{\delta_{s_1}}$	$y_{\delta_{s_2}}$	$y_{\delta_{s_3}}$	$y_{\lambda_{y_1}}$	$y_{\lambda_{y_2}}$	$y_{\lambda_{y_3}}$
0	1	0	0	0	1	0	0	0	0	0
1	0	0	0	–	1	0	0	0	0	0
1	0	0	1	–	0	1	0	0	0	0
0	1	0	1	0	0	1	0	0	0	0
0	0	1	–	0	0	1	0	0	0	0
0	1	0	–	1	0	0	1	0	0	0
0	0	1	–	1	0	0	1	0	0	0
0	1	0	–	1	0	0	0	1	0	0
1	0	0	–	–	0	0	0	1	0	0
1	0	0	–	1	0	0	0	0	1	0
0	1	0	–	0	0	0	0	0	1	0
0	0	1	–	0	0	0	0	0	1	0
0	1	0	–	1	0	0	0	0	0	1

Table 4.13 Macromolecule cover for constraint $(4, 2, 1, 6)$

cs	i		OP
	x_1	x_2	
s_2	0	0	$\{\delta_{s_1}\}$
s_1	0	–	$\{\delta_{s_1}\}$
s_1	1	–	$\{\delta_{s_2}\}$
s_2	1	0	$\{\delta_{s_2}\}$
s_3	–	0	$\{\delta_{s_2}\}$
s_2	–	1	$\{\delta_{s_3}\}$
s_3	–	1	$\{\delta_{s_3}\}$
s_2	–	1	$\{\lambda_{y_1}\}$
s_1	–	–	$\{\lambda_{y_1}\}$
s_1	–	1	$\{\lambda_{y_2}\}$
s_2	–	0	$\{\lambda_{y_2}\}$
s_3	–	0	$\{\lambda_{y_2}\}$
s_2	–	1	$\{\lambda_{y_3}\}$

4.4 Macromolecule Splitting

When the macromolecule cover has been constructed, it is not guaranteed that it is possible to make a molecule distribution, based on this macromolecule cover, that does not violate the PLAM-size constraint. Particularly, when one of the macromolecules violates the PLAM-size constraint, then no molecule distribution can be made that satisfies the PLAM-size constraint.

Definition 4.16 Let A be a macromolecule cover, let $a \in A$ be a macromolecule, and let c be a PLAM-size constraint. Then *a violates the PLAM-size constraint c* if for each set of macromolecules $A' \subseteq A$ such that $a \in A'$, the molecule set $\bigcup A'$ violates the PLAM-size constraint c.

Using the fact that an independent claim of a set of molecules in a molecule cover can not be eliminated by merging it with other sets of molecules, we can easily derive that a macromolecule a violates the PLAM-size constraints if one of the following conditions is true

$$\begin{array}{lcl} c.nr_{IB} & < & |\mathcal{C}_{EIB}(a)|; \\ 2^{c.nr_{SB}} - 1 & < & |\mathcal{C}_{S}(a)|; \\ c.nr_{OB} & < & |\mathcal{C}_{EOB}(a)|; \\ c.nr_{TL} & < & |\mathcal{C}_{PTL}(a)|. \end{array} \tag{3}$$

Remark that it is not true that one of these conditions must be true when a violates the PLAM-size constraint.

If a macromolecule cover contains a macromolecule that violates the PLAM-size constraint, then it is not possible to make a molecule distribution, based on this macromolecule cover, such that this molecule distribution satisfies the PLAM-size constraint. Therefore, we check for each macromolecule whether one of the conditions of (3) is true. If this is the case, then the macromolecule is splitted in the hope that the resulting smaller macromolecules each make fewer claims than the original macromolecule. Each molecule in the macromolecule then becomes a macromolecule on its own. Trivially, a macromolecule with cardinality one cannot be splitted in this way. Such a macromolecule can be splitted in another way. Suppose that the macromolecule is given by $\{(cs, i, OP)\}$. Then, for each $op \in OP$, we introduce a new macromolecule $\{(cs, i, \{op\})\}$. When OP has a cardinality of 1, this way of splitting does not have effect, and the macromolecule is called unsplittable.

Algorithm 4.1 gives a method for doing the splitting as has been described above, for a given macromolecule cover and a given PLAM-size constraint. Remark that the algorithm may report that the macromolecule cover is not usable. This happens when a macromolecule violates the PLAM-size constraint, but is not splittable in the way that we described. In this case, it does not make sense to build a molecule

distribution based on the macromolecule cover, because this molecule distribution surely violates the PLAM-size constraint. The only possibility is to try another macromolecule cover as input for the splitting algorithm.

```
function SplitMamos( A: set of macromolecules;
                     c: PLAM-size constraint)
                     : set of macromolecules;
  var
    MamosToCheck: set of macromolecules;
    MamosChecked: set of macromolecules;
    a: macromolecule;

  function Violation( a: macromolecule;
                      c: PLAM-size constraint)
                      : boolean;
  begin
    Violation := |C_EIB(a)| > c.nr_IB ∨
                 |C_S(a)| > 2^(c.nr_SB) − 1 ∨
                 |C_EOB(a)| > c.nr_OB ∨
                 |C_PTL(a)| > c.nr_TL;
  end;

begin
  MamosToCheck := A;
  while MamosToCheck ≠ ∅ do begin
    Choose macromolecule a ∈ MamosToCheck;
    MamosToCheck := MamosToCheck \ {a};
    if Violation(a,c) then
      if a is splittable then
        Split a;
        Add resulting macromolecules to MamosToCheck;
      else
        Report that A isn't usable as macromolecule cover
    else
      MamosChecked := MamosChecked ∪ {a}
  end;
  SplitMamos := MamosChecked;
end;
```

Algorithm 4.1 Splitting algorithm

4.5 Usage of Beam Search

The decomposition method which has been described can be seen as a sequence of transformations. These transformations are the rectangles

of figure 4.1. Each transformation maps single elements of its domain to single elements of its codomain. For example, the macromolecule splitting transformation maps a macromolecule cover to another macromolecule cover.

In each transformation, some choices are made that can not be reversed during future transformations. Therefore, good choices must be made during all transformations. However, most choices must be made using incomplete information. It is very difficult to estimate the impact of choices on the final result, especially at the beginning of the search process. One way of coping with this lack of information is the use of beam search [4] [5]. The idea behind beam search is the following. When it is hard to choose between alternatives, because there is too few information to see which of the alternatives is the best, the choice is postponed until enough information is available to choose the best alternative. All alternatives are explored in parallel. These thoughts can be put into correspondence with the transformations that take place during the search process, by allowing multiple results of each transformation. When a transformation is applied to a certain element of its domain, multiple elements of its codomain result, and each element corresponds with an alternative. For example, for macromolecule construction there were four different ways, and we have given a table for choosing one of these four ways, depending on the constraint exceeding factors. Using beam search, we can choose more than one of the four possible ways, and consider all resulting macromolecule covers in parallel as alternatives.

When beam search is integrated in the decomposition method, each stage of the process results in a set of objects instead of one object. These objects can be macromolecule covers, molecule distributions or PLAM networks. In each stage, the following is done. First, the transformation is applied to each object. This results in a new generation of objects, all belonging to the codomain of the transformation. Secondly, a selection of the most promising objects in the new generation is made. This is called 'pruning'. When the pruning has taken place, the objects in the new generation are used for the next stage of the search process.

5 Conclusions

Few years ago, in the Section of Digital Information Systems of the Eindhoven University of Technology, a method was developed for decomposition of sequential machines into PLAM networks, with constraints on the PLAM-size as hard constraints, and with constraints on the number of PLAMs and number of connections in the PLAM network as soft constraints [4]. The aim of the reported graduation project was to develop a modified method which would be able to process effectively sequential machines with large states and which would use direct communications between PLAMs to reduce the delay. This aim has been realized.

We have developed a new decomposition model, that describes PLAM network decompositions in terms of task distributions. The tasks that are distributed can be smaller than the tasks composed of all the computations related to a single state, as it was in [4]. Therefore, the new decomposition model eliminates problems with large states. The model describes PLAM network decompositions that use direct communications between PLAMs. This results in delay reduction. A characteristic of the decompositions that are described by the model, is that they are simultaneous: for one current state and input, multiple PLAMs can be active simultaneously.

We have also developed a modified decomposition method, based on the new model. This method finds, for a certain sequential machine and PLAM-size constraint, among all decompositions that are described in the model, a decomposition that satisfies the hard and soft constraints. The proposed decomposition method determines subtasks of the prototype machine in such a way that distributing these subtasks over PLAMs probably leads to a decomposition that satisfies the hard and soft constraints. Afterwards, the subtasks are distributed using a packaging algorithm, and finally the subtask distribution is translated into a PLAM network. The method is described precisely, except the packaging part, which is not explained here because the packaging methods given in [4] and [5] can be applied to this specific packaging problem after some slight modifications.

We have succeeded in describing the problem statement, the decomposition model and the decomposition method in a formal framework. The formal treatment not only facilitates the final implementation of the method in a software tool, but also makes it possible to prove that the method yields correct decompositions under all circumstances.

Notation

Boolean expressions

$\neg P$	not P
$P \wedge Q$	P and Q
$P \vee Q$	P or Q
$P \Rightarrow Q$	P implies Q
$P \Leftarrow Q$	P is a consequence of Q
$P \Leftrightarrow Q$	P is equivalent with Q

Terms

$\times$	don't care
$\mathcal{T}(B)$	all terms over set of bits B
$\mathcal{T}_k(B)$	k-terms over set of bits B
$\mathcal{T}_0(B)$	minterms over set of bits B
$B_a(t)$	set of a-bits of term t
$t_a(B)$	all-a-term over set of bits B
$t \cdot t'$	concatenation of term t and t'
$[t]_B$	projection of term t on set of bits B
$t \geq t'$	term t covers term t'
$t \leq t'$	term t is covered by term t'
$\mathcal{C}_k(t)$	set of covered k-terms of term t
$\mathcal{C}_0(t)$	set of covered minterms of term t

Sequential Machines

$\mathcal{H}(IB, S, OB)$	horizontals over set IB of input bits, set S of states and set OB of output bits

PLAMs and PLAM Networks

$\mathcal{TL}(IB, SB, OB)$	termlines over set IB of input bits, set SB of state bits and set OB of output bits
$M_{impl}(p)$	sequential machine implemented by PLAM p
$M_{impl}(n)$	sequential machine implemented by PLAM network n
$S_{impl}(n)$	states that are implemented by PLAM network n
$EIB_n(p)$	external input bits of PLAM p in network n
$IIB_n(p)$	internal input bits of PLAM p in network n
$IOB_n(p)$	internal output bits of PLAM p in network n
$EOB_n(p)$	external output bits of PLAM p in network n
$nr_{conn}(n)$	number of connections of network n

Operations, Atoms and Molecules

$Operations(M)$	operations of sequential machine M
$Atoms(M)$	atoms of sequential machine M
$RequiredAtoms(M)$	required atoms of sequential machine M
$AllowedAtoms(M)$	allowed atoms of sequential machine M
$ForbiddenAtoms(M)$	forbidden atoms of sequential machine M
$CoveredAtoms(c)$	atoms that are covered by cube c

Decomposition Model

$CS(B)$	current states of the PLAM of block B
$SB(B)$	state bits of the PLAM of block B
$NS(B)$	next states of the PLAM of block B
$TS_{B \rightarrow B'}(\pi)$	transmitted states from the PLAM of block B to the PLAM of block B' in a molecule distribution π
$TIB_{B \rightarrow B'}(\pi)$	transmitted internal bits from the PLAM of block B to the PLAM of block B' in a molecule distribution π
$EIB(B)$	external input bits of the PLAM of block B
$IIB_\pi(B)$	internal input bits of the PLAM of block B in molecule distribution π

$IOB_\pi(B)$	internal output bits of the PLAM of block B in molecule distribution π
$EOB(B)$	external output bits of the PLAM of block B
$EB(\pi)$	network internal bits of molecule distribution π
$AC(B)$	activation conditions of block B
$ptl_{\pi,\gamma,B}$	primary term line function for block B of molecule distribution π, under network state assignment γ
$stl_{\pi,\gamma,B}$	secondary term line function for block B of molecule distribution π, under network state assignment γ
$PTL_{\pi,\gamma}(B)$	primary term lines of the PLAM of block B of molecule distribution π, under network state assignment γ
$STL_{\pi,\gamma}(B)$	secondary term lines of the PLAM of block B of molecule distribution π, under network state assignment γ
$PLAM_{\pi,\gamma,B}$	PLAM that is induced by block B of molecule distribution π, under network state assignment γ

Claims

$\mathcal{C}_x(B)$	values that are claimed by the molecules of B for a certain claim type x
$\mathcal{C}_x(B,C)$	values that are claimed by the molecules of B for a certain claim type x in a molecule cover C

Multiple Output Binary Functions

f_T	multiple output binary function described by PLA table T

Constraint exceeding factors

$\varepsilon_{IB}(T,c)$	constraint exceeding factor for input space corresponding with transition table T and constraint c
$\varepsilon_S(T,c)$	constraint exceeding factor for state space corresponding with transition table T and constraint

	c
$\varepsilon_{OB}(T, c)$	constraint exceeding factor for output space corresponding with transition table T and constraint c
$\varepsilon_{TL}(T, c)$	constraint exceeding factor for term line space corresponding with transition table T and constraint c

Quantifications

$\bigcup_R T$	union quantification with range R and term T
$\exists_R T$	existential quantification with range R and term T
$\exists!_R T$	'there is exactly one' quantification with range expression R and term expression T
$\forall_R T$	universal quantification with range R and term T
$\Sigma_R T$	addition quantification with range R and term T

Sets

$\{\ldots\}$	set
2^S	powerset of set S
$\{p \mid Q\}$	set containing elements p such that Q is true
$\lvert S \rvert$	cardinality of set S
$S \cup S'$	union of set S and set S'
$S \cap S'$	intersection of set S and set S'
$\bigcup \mathcal{S}$	union of all elements of $\mathcal{S}$, where $\mathcal{S}$ is a set of sets
$\bigcap \mathcal{S}$	intersection of all elements of $\mathcal{S}$, where $\mathcal{S}$ is a set of sets

Sequences

$\langle\ldots\rangle$	sequence
ε	empty sequence

$\mathcal{Q}_0(S)$	all sequences over S
$\mathcal{Q}(S)$	all non-empty sequences over S

Functions

$f : A \multimap B$	function f with domain A and codomain B
$f\|_A$	restriction of function f to subset A of its domain

Miscellaneous

$s.f$	field f of structure s
$\lceil x \rceil$	ceiling of x
$\lfloor x \rfloor$	floor of x
$\square$	undefined

Bibliography

[1] Baranov, S.I. and L. Bregman, Automata Decomposition and Synthesis with PLAM. *Microprocessing and Microprogramming. Proc. of 19th EUROMICRO Symposium on Microprocessing and Microprogramming*, Barcelona, Spain, 6-9 September 1993. Vol. 38 (1993), No. 1-5. P. 759-766.

[2] Hartmanis, J. and R.E. Stearns, *Algebraic Structure Theory of Sequential Machines.* Englewood Cliffs, NJ, USA: Prentice-Hall, 1966.

[3] Jóźwiak, L., Simultaneous Decompositions of Sequential Machines. *Microprocessing and Microprogramming. Proc. of EUROMICRO Symposium on Microprocessing and Microprogramming.* Amsterdam, Netherlands, 27-30 August 1990. Vol. 30 (1990), No. 1-5. P. 305-312.

[4] Jóźwiak, L. and J.C. Kolsteren, An Efficient Method For the Sequential General Decomposition of Sequential Machines. *Microprocessing and Microprogramming. Proc. of EUROMICRO Symposium on Microprocessing and Microprogramming.* Vienna, Austria, 2-5 September 1991. Vol. 32 (1991), No. 1-5. P. 657-664.

[5] Jóźwiak, L. and F.A.M. Volf, An Efficient Method for Decomposition of Multiple Output Boolean Functions and Assigned Sequential Machines. In: *Proc. of European Design Automation Conference EDAC*, Brussels, Belgium, 16-19 March 1992. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 1992. P. 114-122.

[6] Levin, I.S., A Hierarchical Model of the Interaction of Microprogrammed Automata. *Avtomatika i Vychislitel'naya Tekhnika (Automatic Control and Computer Sciences).* Vol. 21 (1987), No. 3. P. 77-83.