Abstract

A well known problem when reasoning about concurrent systems is that of state explosion. One of the strategies that has been proposed to alleviate this problem is to make use of the symmetries which a concurrent system may exhibit to construct a symmetry-reduced model that reflects the behaviour of the system. The main contribution of this thesis is an investigation into the theoretical foundations of the method by considering symmetries in the context of category theory. It seems natural to do so since the morphisms that characterise each category may be thought of as a kind of simulation of behaviour.

A new category of language systems is presented, together with several subcategories. Morphisms in this category are defined to preserve structure. The notion of a symmetry of a language system is defined and the quotient structure of the language system is given. The important question of behaviour preservation between the system and its symmetry-reduced model is generalised to the notion of morphism in the category. The conditions required on the morphism to ensure that it preserves behaviour are identified. These results are extended to the projection morphism that define the symmetry-reduced model by constructing a split morphism. Two specific behaviours, namely absence of deadlock and extensibility, are considered.

The second contribution of this thesis is to establish a categorical relationship between the language system model and elementary nets. A vector language semantics for elementary nets is given. Functors between these categories are defined and the existence of an adjunction is proved.
Acknowledgements

I would like to thank my supervisors, David Pitt and Mike Shields, for advising, supporting and guiding me in my research.
# Contents

1 Introduction .......................................................... 1

2 Symmetry Reductions ........................................... 7
   2.1 Groups and Symmetry ......................................... 7
   2.2 State Spaces .................................................... 8
   2.3 Symmetry Reduced State Spaces ......................... 11
   2.4 Detection of Symmetries ....................................... 15
   2.5 Preserved Properties .......................................... 16
   2.6 Choices of the group ......................................... 20

3 Symmetry Reductions in the Context of Category Theory 23
   3.1 Category Theory .................................................. 23
   3.2 Language Systems and the Category LS .................. 25
   3.3 Wide Subcategories of LS ...................................... 29
   3.4 Permutations of Language Systems ....................... 32
   3.5 Quotient Structure of the Language System .......... 34

4 Behaviour Preservation ........................................... 40
   4.1 Monoid Morphisms and the Commutativity Condition 41
   4.2 The Category of Vector Languages ....................... 45
   4.3 A Proof of Behaviour Preservation ....................... 55
   4.4 The Split Morphism .......................................... 60

5 Properties of Language Systems ............................... 66
   5.1 Absence of Deadlock ......................................... 67
   5.2 Extensibility .................................................... 71
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6 Categorical Relationship Between Nets and Language Systems</td>
<td>73</td>
</tr>
<tr>
<td>6.1 Elementary Nets and the category EN</td>
<td>73</td>
</tr>
<tr>
<td>6.2 The Functor from EN to LS</td>
<td>77</td>
</tr>
<tr>
<td>6.3 The Functor from LS to EN</td>
<td>82</td>
</tr>
<tr>
<td>6.4 An Adjunction</td>
<td>92</td>
</tr>
<tr>
<td>7 Conclusions and Future Work</td>
<td>104</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This thesis is concerned with reasoning about the behaviour of complex concurrent systems. As computer systems become increasingly complex, verifying the behaviour of such systems becomes increasingly difficult. Nevertheless as these systems become more widely used it is increasingly important to perform these verifications. A barrier to the use of formal methods for verifying behavioural properties is the difficulty of performing proofs at a global level for large complex systems. One of the ways that the analysis of complex computer systems may be simplified is by exploiting their symmetry properties. This is an established method which uses group theory to abstract away from the details of a large complex system to obtain a smaller and simpler one. This smaller system, called the quotient system, is constructed in such a way that it reflects the behavioural properties of the large system and so may be used to check its correctness. This thesis aims to investigate the general conditions under which a quotient or symmetry-reduced model embodies the behaviour of the original model.

Concurrency is the simultaneous progression of two or more processes/programs which can interact with one another. Examples of concurrent systems include computer networks and flight-control systems. Each component (i.e. program/process) in a concurrent system is continuously interacting with its environment. The environment of one component being made up of the other components in the system.

It is the reactive nature of concurrent systems that differentiates them from traditional sequential programs. Since such traditional programs terminate, they can be modelled
as mappings from inputs to outputs. Such a program can then be verified, which is to say that the formal description of the program meets the properties expected of that program, by checking that the output expected following a certain input does indeed result. This input/output method of checking can only be used for concurrent programs that represent a 'parallelised' version of a sequential program, but cannot be used for reactive nonterminating concurrent systems.

There are two methods of verifying a finite state (nonterminating) concurrent system - state space exploration and theorem proving. The exploitation of symmetries to simplify the analysis of concurrent systems, as presented in this thesis, is applicable to both methods. State space methods examine each of the possible states a system may reach, starting from a given initial state, by executing all enabled transitions in each state. This involves the construction of a structure (the 'state space') representing all the possible states of the system, which is normally done automatically. Model checking is an automatic state space exploration technique which can verify temporal properties of a finite state system. If the system does not satisfy a property the model checking tool gives a counter-example.

The property of exploring all possible states a system can reach differentiates state space exploration techniques from more traditional software engineering techniques for finding errors such as simulation and testing. Although simulation and testing are useful they may not expose an error of a concurrent system because they only explore part of the possible behaviours of a system.

An alternative to state space exploration for verifying a system is theorem proving, which uses axioms and inference rules to verify a system, either manually or using a theorem proving tool. However, even the theorem proving tools require the user to define invariants of the system (these are properties which hold at every reachable state) whereas the construction of a state space can be fully automatic. As the emphasis of theorem proving methods is to establish the correctness (or otherwise) of a system, they are not as good as state space methods at determining the nature and location of errors in an incorrect system. Also, the construction of a partial state space may be sufficient to identify an error.
Clearly then state space verification methods have many advantages. Unfortunately, they have one major practical disadvantage - the state explosion problem. An appreciation of the extent of the state explosion can be found in [17], where empirical results are presented that can be applied to models with in excess of $10^{20}$ states. Even for a simple concurrent system, the number of states in the state space can be huge, for example the classic dining philosophers system with $n$ 4-state philosophers has $3^n - 1$ states [32]. This problem is caused, in part, by the modelling of concurrency by interleaving. It is necessary to explore all possible interleavings of concurrent events to verify the system. Since the execution of $n$ concurrent events entails $n!$ interleavings, it is clear why the verification of concurrent systems entails the examination of such a huge number of states. Consequently, generating all the possible states is not possible for most concurrent systems. But the advantages of state space methods have prompted researchers to investigate methods which reduce the number of states to be constructed, one of which makes use of the symmetries in the system to be verified.

A concurrent system that is composed of components that are essentially indistinguishable and interchangeable exhibits considerable symmetry. The extent to which components are indistinguishable and can be interchanged depends on the specifics of the system and the level of abstraction at which it is modelled. A simple example of symmetry is the mutual exclusion problem for two processes, $A$ and $B$. The state where $A$ is in the critical section and $B$ waiting is equivalent to the state where $B$ is in the critical section and $A$ is waiting. For a cache coherence protocol that considers all processors to be identical, the states resulting from permuting the processors are equivalent. So to verify the model it is sufficient to check just one of these symmetric (or equivalent) states, allowing for the generation of a reduced state space. This symmetry-reduced or quotient state space is constructed in such a way that it has the same behavioural properties as the full state space.

Formal models for concurrent systems may be either a system level model or a behaviour level model. System models such as Petri nets [22] and transition systems give an explicit representation of the states the system may be in. Behaviour models such as Mazurkiewicz traces [18], CSP [12] and vector languages [27] focus on what an outside observer sees, representing the behaviour of the system in terms of sequences.
of occurrences together with extra structure (for example refusal sets in CSP) to cope with non-determinism. The model which is used in chapter 3, called a language system, is a behaviour level model. A fundamental class of Petri nets, called elementary nets, is featured in chapter 6.

This thesis presents symmetries in the context of category theory. Category theory is a generalised mathematical theory of structures which, by abstracting away from the details of the structure, reveals relationships between them.

A category is a collection of objects together with a related set of morphisms which typically are structure-preserving maps between objects. In this thesis, the objects of the category will be a model of concurrency. Morphisms demonstrate how each object relates to the others in the same category, i.e. how different systems represented by the same type of model are related. This could be a (full) system and its symmetry reduction (or quotient) if the function(s) which define the symmetry reduction are in the category. Hence morphisms can be thought of as a kind of simulation of behaviour making the categorical approach an appropriate choice for studying the foundations of the symmetry method. Since it is the morphisms which characterise the category it is possible to have more than one category of a given model, as will be demonstrated in section 3.2.

The categorical approach also allows for the formal relationships between models to be established and for the translation of results between them. A functor is a map between categories. If a such a map can be defined in both directions between two categories then it may be possible to define an adjunction which is an important categorical tool. An adjunction preserves limits and colimits. An example of a limit is the categorical product between two objects in a category. This product can correspond to the parallel composition of the systems that the objects represent. An example of a colimit is the co-product of two objects in a category which often corresponds to a non-deterministic choice between the systems the objects represent.

The language system model used in this thesis is a simple algebraic model that represents behaviour using the vector languages introduced by Shields [27] [28] [29]. The quotient structure of a language system is defined and several new categories of lan-
guage systems are presented and those which contain that functions which define the quotient are identified. The important question of which morphisms preserve behaviour is studied. The relationship between elementary nets (a class of low level nets) and the language system model will is established by proving the existence of an adjunction between the two models.

The structure of this report is as follows.

Chapter 2 describes the state explosion problem that limits the usefulness of state exploration techniques. The basic principles of group theory needed to exploit the symmetries of a concurrent system and hence alleviate the state explosion problem are presented. A review of how symmetries have been exploited, particularly in the area of model checking, is also given.

Chapter 3 sets up the machinery needed for investigating the theoretical foundations of the symmetry method, which is the main contribution of this thesis. An introduction to category theory is given and the language system model is defined. A new category of language systems together with several wide subcategories are presented. A symmetry of a language system is defined and the quotient structure of the model is given. Which of the language system categories contain the functions that define the quotient is determined.

The question of preservation of behaviour between two language systems in the same category is addressed in chapter 4. The circumstances under which a morphism between two language systems preserves behaviour is determined. The construction of a split morphism is used to extend the results to the projection morphisms that define the quotient system.

Having identified the circumstances under which behaviour is preserved, we then investigate whether specific properties are preserved. Two properties of a language system, namely absence of deadlock and extensibility are defined in chapter 5. For each of these properties we identify the conditions required under which a language system has that property if and only if its quotient has that property.

Chapter 6 presents the second contribution of this thesis: establishing the categorical relationship between language systems and elementary nets. A vector language seman-
tics for elementary nets is given. The existence of an adjunction between the categories representing these different models of concurrency is proved.

Concluding remarks and ideas for future research are given in the final chapter.
Chapter 2

Symmetry Reductions

This chapter is a review of the use of groups of symmetries in the verification of concurrent systems by exhaustive state space exploration. The state space of a concurrent system can be very large and this problem of state explosion complicates verification techniques such as model checking that search the entire state space. The symmetries of a concurrent system can be exploited to reduce the size of the state space to be searched. We describe how model checking with symmetry is performed. The chapter begins by giving the principles of group theory that form the theoretical basis for using symmetry-reduced state spaces in model checking.

2.1 Groups and Symmetry

This section presents a very brief introduction to some basic principles of group theory, concentrating on those aspects that will be used in this thesis. For a fuller introduction to group theory see [14]. It will be shown that the structure-preserving symmetries of an object form a group, called a permutation group. That it can further be shown (using Cayley's theorem) that all finite groups are isomorphic (i.e. structurally identical) to such a permutation group gives rise to the idea that group theory can be considered as the study of symmetry.
Definition 2.1.1 A group is a set $G$ on which a binary operation $*$ is defined such that:

1. $G$ is closed under $*$, i.e. to each ordered pair $(a, b) \in G$, $*$ assigns an element that is again in $G$

2. $*$ is associative

3. there exists an identity, i.e. an element $e \in G$ such that
   \[ e \ast x = x \ast e = x \quad \text{all} \quad x \in G \]

4. for each $a \in G$, there is an inverse, i.e. an element $a' \in G$ such that
   \[ a \ast a' = a' \ast a = e \]

A permutation of a set $A$ is a one-to-one function of $A$ onto $A$. A set of permutations closed under function composition meets the definition above and hence forms a group, as will now be demonstrated. The composition of permutations is closed, function composition is associative, the identity permutation $I$ such that $I(a) = a$ acts as the identity and the inverse of any permutation $\sigma$ is simply $\sigma^{-1}$ which exists and is unique as a consequence of the definition of a permutation. An example is the symmetric group $S_n$ on $n$ objects which is the set of all $n!$ rearrangements of the $n$ objects.

For a permutation group $G$ on a set $X$, the orbit of $x \in X$ is defined to be the set $\{\sigma(x) | \sigma \in G\}$. This is an example of a group acting on a set. The relation on $X$ defined by $x_1 \sim x_2$ if and only if $x_1 = \sigma(x_2)$ (some $\sigma \in G$) is an equivalence relation. An equivalence relation has the property that it partitions the set on which it is defined into cells which are disjoint and exhaustive subsets of that set. Under this equivalence relation two elements of $X$ are related if they are in the same orbit. Hence the orbits partition $X$.

2.2 State Spaces

The first step in verifying a system is specifying the properties that the system should satisfy, e.g. absence of deadlock. In model checking this specification is stated using
2.2. State Spaces

a temporal logic. The next step is to construct a formal model of the system. This model must capture the important features of the system. Two such features are states and transitions. A state is an instantaneous description of the system that includes all relevant information such as the values of variables. A change in the system resulting from the execution of an action results in a new state. This new state and its preceding state form a pair that defines a transition of the system.

Figure 2.1 shows the state space of a system that consists of two processes, one of which executes the sequence of actions \( a_1 a_2 \) and the other executes the sequence of actions \( b_1 b_2 \). The numbers 0 to 8 represent the states of the system, the initial state is labelled with 0. The arrows represent state changes, i.e. transitions. For example, from the initial state 0 the execution of action \( b_1 \) results in the system being in state 2. This system can be used to illustrate the state explosion problem. With two processes executing two actions the number of states is nine. If a third process was introduced that also performed a sequence of two actions then in the worst case the state space would have \( 3^3 = 27 \) states. The addition of a fourth similar process could result in a state space with \( 3^4 = 81 \) states.

Figure 2.1: The state space of a concurrent system with two processes each executing a sequence of two actions

These ideas of state and transition can be formalised as a Kripke structure which is the model used in model checking. Given a first order logic description of a concurrent system, it is possible to extract the Kripke structure that models the system using techniques detailed in [5]. A Kripke structure comprises a set of states, a set of transitions between states and a function that labels each state with the set of atomic propositions
2.2. State Spaces

that are true in that state.

**Definition 2.2.1** A Kripke structure $M$ over a set of atomic propositions $AP$ is a triple $M = (S, R, L)$ where

- $S$ is a finite set of states
- $R \subseteq S \times S$ is the set of transitions
- $L : S \rightarrow 2^{AP}$ is the labelling function

A path in a Kripke structure $M$ from a state $s$ is an infinite sequence of states $\pi = s_0s_1s_2\ldots$ such that $s_0 = s$ and $(s_i, s_{i+1}) \in R$ for all $i \geq 0$. A path models a computation of the system.

Such a system can be represented pictorially as a state transition graph which is a directed graph with a node for each state the system can reach and an arc for each possible state change.

The definition of a Kripke structure can be modified if it is necessary to distinguish between transitions, for example if partial order reduction is used. The transition relation $R$ is replaced by a set of transition relations $T$ such that for each $\alpha \in T$, $\alpha \in S \times S$.

A Kripke structure model of a state space has semantic transitions. Some state space models require structural transitions in addition. Structural transitions represent changes to the underlying system and semantic transitions which represent changes in the state of the system relate to the behaviour of the system. These terms can be represented using a Petri net.

A Petri net [22] has a set of places $P$ (represented graphically represented by circles) and a set of transitions $T$ (represented by rectangles) with directed edges between them. Each transition is connected to its associated set of input places by a directed edge from place to transition. Similarly, each transition is connected to its associated set of output places by a directed edge from transition to place. For the Petri net model, states of the system are represented by an allocation of tokens (drawn as '•') at places. Such an allocation is called a marking and is usually denoted $M$. A transition can only
occur (or fire) if each of its input places has at least one token. When the transition occurs, one token is removed from each of the input places and one token is deposited at each of its output places. For a Petri net, structural transitions are the transitions \( T \) of the actual net and the semantic transitions are the transitions from one marking to the next (denoted \( M[t]M' \)) in the state space representing all possible markings of the underlying net.

### 2.3 Symmetry Reduced State Spaces

A full survey of methods which alleviate the state explosion problem is given by Valmari in [32], of which only symmetry reduction will be considered in this thesis. The idea of using symmetries to reduce the state space has been used since the early 1980's, see [13] for an early application of the method to Petri nets.

A permutation is a symmetry of a Kripke structure \( M \) if it preserves the transition relation \( R \). Such a permutation would also preserve the structure of the state space and hence also preserve the behaviour of the underlying model. The symmetries form a group and the action of this group partitions the (full) set of states into sets of symmetric states. It is then only necessary to examine the future behaviour of one from each of the sets of symmetric states, thus greatly reducing the number of states to be examined when model checking. In effect, symmetric states are collapsed to a single state for the purpose of checking the model. These ideas will now be presented more formally.

A permutation \( \sigma \) on the state space \( S \) of a Kripke structure \( M \) is a symmetry of the structure \( M \) if \( \sigma \) satisfies the following condition

\[
\forall s_1 \in S, \forall s_2 \in S, \quad ((s_1, s_2) \in R \implies (\sigma(s_1), \sigma(s_2)) \in R)
\]

Permutations which preserve the structure of the state space are called automorphisms. Since they are also permutations, a set of automorphisms (providing it satisfies definiton 2.1.1) forms a group, which will be denoted \( G \). The action of \( G \) on a state \( s \in S \) gives the set \( \theta(s) = \{ t | \exists \sigma \in G : \sigma(s) = t \} \), which is the orbit of \( s \). Two states are
2.3. Symmetry Reduced State Spaces

in the same orbit if \( s = \sigma(s') \) for some \( \sigma \in G \). A relation on \( S \) defined to be \( s \sim s' \) if and only if \( s = \sigma(s') \) (some \( \sigma \in G \)) is an equivalence relation. Hence the orbits form a partition of \( S \). The properties of the automorphisms in \( G \) mean that states in the same orbit have equivalent future behaviour. So for the purposes of model checking we can store just one state from each reachable orbit when constructing the state space, rather than examining the future behaviour of all states in the orbit. The state stored is called the representative state and is denoted \( \text{rep}(\theta(s)) \). Hence each node in the symmetry reduced or quotient state space represents an equivalence class of states. We now give the definition of the quotient structure as presented in [5].

**Definition 2.3.1** For a Kripke structure \( M = (S, R, L) \) and group of automorphisms \( G \) acting on the state space \( S \), the quotient structure is \( M_G = (S_G, R_G, L_G) \) where

\[
S_G = \{ \theta(s) | s \in S \} \quad \text{is the set of states}
\]

\[
R_G = \{ (\theta(s_1), \theta(s_2)) | (s_1, s_2) \in R \} \quad \text{is the transition relation}
\]

\[
L_G \quad \text{is the labelling function given by } L_G(\theta(s)) = L(\text{rep}(\theta(s)))
\]

To illustrate these ideas we present the example of a state transition graph and its quotient given in [8] as a solution to the mutual exclusion problem. The system has \( n \) processes each of which has a noncritical section corresponding to the location \( N_i \) and a critical section corresponding to location \( C_i \). Each process moves between its two sections subject to the mutual exclusion property which requires that no two processes are ever in their critical section simultaneously. The state transition graph for an \( n \) process mutual exclusion problem is given in figure 2.2. It has \( n + 1 \) states.

The automorphism group \( G \) for this example is the symmetric group \( S_n \). Quotienting by \( G \) and taking the states \( (N_1, N_2, \ldots, N_n) \) and \( (C_1, N_2, \ldots, N_n) \) as the representative states, the quotient model shown in figure 2.3 is obtained. It is possible to model check over this two state model, which is smaller than the original model which had \( n + 1 \) states.

The quotient state space may be computed without the need to construct the full state space, since this was the motivation for exploiting symmetry. The quotient state space
2.3. Symmetry Reduced State Spaces

Figure 2.2: The state transition graph for the two state \( n \) process mutual exclusion [8]

Figure 2.3: The quotient for the two state \( n \) process mutual exclusion problem [8]
2.3. Symmetry Reduced State Spaces

can be constructed algorithmically using an explicit state representation or symbolically using a BDD (binary decision diagram).

On-the-fly algorithms construct only as much of state space as is required to established whether the property of interest holds, i.e. they may cease before the entire state space has been constructed. An on-the-fly algorithm which constructs the symmetry reduced or quotient state space from a set of initial states is given in [15]. If \( s \) has been stored and \( (s, s') \in R \) then an edge is stored that starts from \( s \) and ends in the state \( s_r \), where \( s_r \) is the representative state from the orbit containing \( s' \). Although time is saved by only storing the unique representative \( s_r \), it is still necessary to examine each of the states that succeed \( s \). The unique representative for each orbit is determined by the canonicalisation function \( \zeta \). Unfortunately, finding this \( \zeta \) function is as least as hard as the Graph Isomorphism problem, as [15] proves. The Graph Isomorphism problem determines whether two graphs are isomorphic, which is to say that they contain the same number of vertices connected in the same way. Although the graph isomorphism problem has not been proved to be NP-hard [3] it is thought to be an NP-hard problem. To overcome the difficulty of finding \( \zeta \), it is suggested in [15] that a less strictly defined function may be used. It is shown that any function \( \zeta \) that maps a state to an equivalent state (not necessarily the unique representative state of the equivalence class) can be used in the algorithm. Using this definition of \( \zeta \), the algorithm will check each equivalence class at least once, which is clearly sufficient to check the model.

Symbolic model checking that makes use of symmetries is discussed in [4], [6] and [3]. The difficulty with symbolic model checking, i.e. when the transition relation is represented by a BDD, is that the orbit relation has to be calculated. The orbit relation determines whether two states, say \( s_1 \) and \( s_2 \), are in the same orbit, i.e. if there exists a permutation \( \sigma \) such that \( \sigma(s_1) = s_2 \). This information is needed to determine the transition relation in the quotient structure and hence the BDD of the quotient. The orbit relation is used to derive the representative function \( \zeta \) which maps each state to a unique representative. However, the orbit relation presents the same problem as the canonicalisation function - determining it is as hard as the Graph Isomorphism problem. Also in common with the canonicalisation function, a solution is offered in [4].
and [3] which uses multiple representatives. Instead of a unique choice from each orbit we now have several, thus avoiding the problem of constructing the orbit relation. As the steps in this model checking procedure always maintain only subsets of each orbit substantial savings are made.

The symmetry reduced state space of a Petri net contains one marking from each equivalence class of markings, the equivalence relation on the markings having been determined by the symmetries of the underlying net. A symmetry reduced state space for a class of high level nets called coloured Petri nets is given by Jensen in [16].

Symmetry reduction methods can also be combined with partial order reduction techniques, [7]. Partial order based methods exploit the independence of actions. It is because the two methods exploit different feature of the systems that allows them to be used simultaneously. Sequences of actions are defined to be equivalent if they are the same up to the reordering of independent actions occurring occurring in the sequences. Only a subset of each class of equivalent actions is considered.

### 2.4 Detection of Symmetries

In order to make use of the symmetry reduction techniques described, the (behaviour-preserving) symmetries of the system have to be detected. Clearly, this needs to be done without constructing the entire state space since this is the very thing we are trying to avoid doing.

As an alternative to the user describing the symmetries of the system, there are automatic methods of detecting symmetries.

Ip and Dill [15] present an automatic method of detecting symmetries by inspecting the system description. Instead of the user identifying symmetries, this is done automatically by including in the description language a new data type called scalarsets. A scalarset is a finite set and is usually used in place of a subrange, so that a scalarset of size \( n \) represents a subrange from 0 to \( n - 1 \). If the description of the system has identical elements that can be permuted without changing any verification properties this can
be identified by converting a subrange to a scalarset. The presence of a scalarset indicates that elements of the subrange can be permuted without affecting the verification properties.

Restrictions on the use of scalarsets preclude symmetry-breaking constructs. Hence a scalarset is a fully symmetric type, i.e. states are guaranteed to have the same future behaviours, up to permutation of the elements of the scalarset. Therefore the presence of a scalarset indicates that certain symmetries hold on the state space. It is proved that scalarsets induce an automorphism on the state space. Indeed every permutation of a scalarset is an automorphism on the state space, since scalarsets are fully symmetric. So for a scalarset of size n, the group used to obtain the quotient structure will be the symmetry group $S_n$, giving savings approaching a factor of n!

There are algorithms that compute symmetries automatically. In [25], Schmidt presents an algorithm that computes the symmetries of a given place/transition net. These symmetries are all bijections from places and transitions to places and transitions that respect places, transitions and arcs. In [33], Varpaaniemi points out that in practice it is often sufficient to know just the place symmetries (i.e. the bijections on the set of places that correspond to symmetries) and presents an algorithm, similar to Schmidt's, but which computes just the place symmetries. Schmidt extends his algorithm in [26] to cover other types of nets, including stochastic nets. Algorithms such as these form the basis of the state space generation tools for nets, examples of which are PROD [34] and INA [23].

2.5 Preserved Properties

Only certain types of verification questions can be answered using a symmetry-reduced state space - those concerning properties which if true in the reduced state space are true in the full state space. The existence of a bi-directional correspondence between paths in the full state space and paths in the reduced state space is used to determine those properties which are true in both. A state $s'$ is said to be reachable from the state $s$ if and only if it occurs in a path starting at $s$. This Correspondence Theorem ([4], [3], [8], [10], [15]) states that for every path starting from $s_0$ is $M$ there exists a
2.5. Preserved Properties

corresponding path in the quotient $M_G$ starting from $\theta(s_0)$, and for every path starting from $\theta(s_0)$ in $M_G$ there is a corresponding path starting from $s_0$ in $M$ (where $G$ is a subgroup of $M$). Two paths $s_0s_1 \ldots$ in $M$ and $\theta(t_0)\theta(t_1)\ldots$ in $M_G$ are said to correspond if and only if $s_i \in \theta(t_i)$ for all $i$. This theorem tells us that state $s$ is reachable in the full state space if and only if $\theta(s)$ is reachable in the reduced state space, so if any state in the orbit of $s$ can be reached, then $s$ can be reached.

In [4] and [3], Clarke et al use the Correspondence Theorem to prove that a property expressed in a temporal logic such as $\text{CTL}^*$ is true in the full structure $M$ if and only if it is true in the quotient $M_G$, provided that the group $G$ comprises automorphisms which preserve both the structure of the model and the structure of the $\text{CTL}^*$ formula to be verified. Before describing automorphisms of $\text{CTL}^*$ formulas, we give a description of the the $\text{CTL}^*$ logic which starts by considering a restricted version of $\text{CTL}^*$ called $\text{CTL}$.

A Computation Tree Logic (CTL) formula is interpreted over a computation tree of a Kripke model, unlike a Linear Time Temporal Logic (LTL) formula which is interpreted over each path of a Kripke structure. CTL uses the temporal operators X (nexttime) and U (until) preceded by the existential path quantifier E. For example, the CTL formula $E \ X p$ indicates that in some of the paths starting in the current state $s_i$, $X p$ is true in $s_i$ if and only if $p$ is true in $s_{i+1}$, where $p$ is an atomic proposition. The set of CTL formulas is generated by the following rules, [8]:

- every atomic proposition such as $p$ is a CTL formula
- if $f$, $g$ are CTL formulas then so are $f U g$, $E f$, $X f$, $f \land g$ and $\neg f$

In addition to the basic symbols already described, CTL uses the additional temporal operators F (sometime) and G (always), the propositional connectives $\lor$, $\Rightarrow$ and the universal path quantifier A. The path quantifier $A$ indicates that the temporal formula following it is true in all paths starting in the current state. All of these additional operators and connectives can be derived from those already described, for example $A f$ has the same meaning as $\neg E \neg f$. 
The basic modalities of CTL are of the form $A$ or $E$ followed by a single temporal operator $F$, $G$, $X$ or $U$. CTL* is an extension of CTL, where formulas can allow for boolean combinations and nestings of these basic modalities.

Recall that the Correspondence Theorem ([4],[3]), is used to prove the preservation of the correctness of a property expressed in CTL*, provided that the group $G$ comprises automorphisms which preserve both the structure of the model and the structure of the CTL* formula to be verified. Hence for a model $M$ and a formula $f$, we require that $G = Aut M \cap Auto f$ where $Aut M$ is the set of automorphisms of $M$ and $Auto f = Aut f_1 \cap \ldots \cap Aut f_n$ for $f$ comprising $f_1 \ldots f_n$, where $f_1 \ldots f_n$ is the list of atomic formulae occurring in $f$. Using $Auto f$ instead of $Aut f$ ensures that the internal symmetry of $f$ is captured. The subgroup $Auto f$ is defined as follows, [8]:

1. for a propositional formula $f$, $Auto f = Aut f$. For example, if $f = p_1 \land p_2$ then $Aut f$ is the group comprising the identity and the permutation that transposes the process indices 1 and 2

2. for a CTL* formula $f$, $Auto f$ is defined inductively as follows:

    (a) if $f = X g$ or $f = E g$ then $Auto f = Auto g$

    (b) if $f = g U h$ then $Auto f = Auto g \cap Auto h$

    (c) otherwise $f$ is a boolean combination of atomic propositions and subformulas of the form $X g$, $g U h$ and $E g$. Let $f = b(e_1, e_2, \ldots, e_k, f_1, f_2, \ldots, f_l)$ where $b$ is a boolean formula over the atomic propositions $e_1, e_2, \ldots, e_k$ and the subformulas $f_1, f_2, \ldots, f_l$. Now replace each $f_i$ by an unindexed proposition $F_i$ and define

    $Auto f = Auto b(e_1, e_2, \ldots e_k, F_1, F_2, \ldots, F_l) \cap Auto f_1 \cap \ldots \cap Auto f_l$. Note that $Auto b(e_1, e_2, \ldots e_k, F_1, F_2, \ldots, F_l)$ is a propositional formula so may be calculated as in 1. above, and the form of each of the $f_i$ is such that $Auto f_i$ may be calculated as in 2(a) or 2(b)

This construction of $Auto f$ is exemplified in [8] using the CTL* formula

$f = p_1 \land EX (q_1 \lor q_2) \lor p_2 \land EX (q_1 \lor q_2)$, where $p_1, p_2, q_1, q_2$ are atomic propositions.
2.5. *Preserved Properties*

If the set of process indices is \( I = \{1, 2\} \), the application of part 2.(c) of the definition above yields \( \text{Auto} f = \text{Auto} (p_1 \land B \lor p_2 \land B) \cap \text{Auto} (\text{EX}(q_1 \lor q_2)) \) where \( B \) is an unindexed proposition. By part 1. of the definition we have \( \text{Auto} (p_1 \land B \lor p_2 \land B) = \text{Sym} I \) where \( \text{Sym} I \) is the set of all permutations on \( I \). Applying part 2.(a) twice yields \( \text{Auto} (\text{EX}(q_1 \lor q_2)) = \text{Sym} I \), hence \( \text{Auto} f = \text{Sym} I \).

We now sketch the role of group \( \text{Auto} f \) in the proof [4],[8] that a property expressed in CTL* is true in the full structure \( M \) if and only if it is true in the quotient \( M_G \), provided that the group \( G \) is a subgroup of \( \text{Aut} M \cap \text{Auto} f \). Assume that \( f \) is true in \( M \), then there is a path \( s_0 s_1 \ldots \) in \( M \) satisfying \( f \). By the Correspondence Theorem there exists a corresponding path \( \theta(s_0)\theta(s_1)\ldots \) in \( M_G \). Although \( s_i \) and \( \theta(s_i) \) are not in general the same state they are in the same orbit. It is the action of \( G \) on \( S \) that produces these orbits and since \( G \) is a subgroup of \( \text{Auto} f \) the path \( s_0 s_1 \ldots \) in \( M_G \) also satisfies \( f \). Hence \( f \) is also true in \( M_G \).

Another interesting example of the calculation of automorphisms of a CTL* formula is given in [9]. Here \( f = \text{E} (\text{GF} \, \text{ex}_1 \land \text{GF} \, \text{ex}_2) \) where the proposition \( \text{ex}_i \) is such that it holds in a state \( s \) if and only if all transitions leading to \( s \) are the result of a single execution step of the process \( i \). If \( I = \{1, 2, 3, 4\} \) then \( \text{Auto} f \) comprises those permutations on \( I \) that leave 1 and 2 fixed. Now consider the formula \( \Phi = \text{GF} \, \text{ex}_1 \land \ldots \land \text{GF} \, \text{ex}_n \).

A computation path is said to be unconditionally fair if it satisfies \( \Phi \). Such fairness constraints are often used in conjunction with interleaving semantics to avoid process starvation by ensuring that individual processes 'make progress'. Unconditional fairness means that each process is executed infinitely often. If \( f = \text{E} \Phi \) and \( I = \{1, \ldots, n\} \) then \( \text{Auto} f \) is simply the identity permutation as this is the only permutation on \( I \) that leaves each of the \( \text{ex}_i \) invariant. Consequently \( G = \text{Aut} M \cap \text{Auto} f \) comprises only the identity permutation and no reduction of the state space is achieved. The conclusion [9] is that the pure group-theoretic methods can not exploit symmetry when dealing with fairness constraints. In [9] and [11] automata-theoretic methods are described which do allow symmetries to be used when model checking under fairness assumptions.

Further reductions can be achieved by exploiting the local symmetries of the path subformulas CTL* formula, as presented in [1] using the framework of LTL (linear
The Correspondence Theorem is also used by Jensen in [16] to prove that it sufficient to check dynamic properties of nets such as reachability, liveness and boundedness in the quotient state space.

A corollary of the Correspondence Theorem concerns deadlock detection. In a concurrent system the presence of multiple threads of control can lead to two or more processes competing for resources. The competition can lead to deadlock, which is peculiar to concurrent systems. A process is said to have a deadlock if there exists a reachable state in which it is impossible for the process to be executed any further. Deadlock can be more precisely defined in terms of enabled transitions. A structural transition \( t \) is enabled in a state \( s \) if and only if there is a state \( s' \) such that \( (s, t, s') \in T \) (recall \( T \) is the set of transition relations which replaces the transition relation \( R \) in the definition of a Kripke structure when it is necessary to distinguish between transitions). A state is said to be a deadlock if no structural transition is enabled in it. As a result of the Correspondence Theorem we have that if a (full) state space has a reachable deadlock state \( s_d \) then the quotient state space also has a reachable deadlock state \([s_d]\). Although this is only a necessary condition, a lemma in [15] extends the condition to allow deadlock to be determined without inspecting the full state space.

2.6 Choices of the group

In section 2.3 the group \( G \), which determined the quotient state space, was defined to be the set of those permutations that preserve the structure of the model. \( G \) may be any subgroup of the group of automorphisms, but to achieve maximum reduction in the size of the state space we require the largest possible group \( G \). Hence the most desirable case arises when all the permutations are also automorphisms, in which case \( G \) will be of size \( n! \), where \( n \) is the number of processes.

In [8] and [10] other possibilities for \( G \) are compared. It has already been commented on that when checking a CTL* formula \( f \), the group must be a subgroup of \( Aut M \cap Auto f \) in order to preserve the correctness of \( f \) in the quotient state space. If \( f \) is a complex
formula composed of \( i \) subformulæ and has little symmetry then \( Auto \ f \) may be small in which case it is often better to decompose \( f \) into smaller subformulæ which are then checked individually (i.e. form \( i \) smaller quotients and check each in turn).

If the quotient is endowed with additional structure then the model checking of any \( \text{CTL}^* \) formula \( f \) may be carried out on a quotient structure determined by \( Aut \ M \) only. This removes the need to compute both \( Auto \ f \) and its intersection with \( Aut \ M \). This is called the annotated quotient structure \([8][10]\). The annotated quotient is \( M_G = (S_G, AR) \) where \( G \) is any subgroup of \( Aut \ M \). \( S_G \) is the set of representatives of the partition of \( S \) into equivalence classes and \( AR \) is the annotated relation. The annotated relation labels each transition with additional information denoting how co-ordinates are permuted from one state to the next. For each transition \( ([s], t) \in R \) the triple \( ([s], \sigma, [t]) \) is contained in \( AR \), for some \( \sigma \in G \) such that \( t = \sigma([t]) \). A transition from a representative state \([s]\) to a representative state \([t]\) is included in \( AR \) with \( \sigma \) the identity. Also transitions from a representative state \([s]\) to a non-representative state \( t \) is included in \( AR \) with \( \sigma \) not the identity. Transitions from non-representative states are not included.

An example of an annotated quotient structure will now be given. This example, taken from \([11]\) concerns a simplified resource controller, which is a program consisting of one server and three client processes. Each process is in exactly one of the following states: idle, request or critical, denoted by \( I \), \( R \) and \( C \) respectively. A process can move from the idle to the request state freely. The server can grant the resource to a process that is in the request state by moving it to the critical state, providing that no other process is in the critical state.

Any combination of \( I \)'s, \( R \)'s and one \( C \) is allowed which means the model for this example has twenty states. However its annotated quotient structure, shown in figure 2.4, has only seven (representative) states.

Some explanation of figure 2.4 will now be given. The initial state is marked III, where all three processes are in the idle state. The state RII represents the states IRI, IIR and itself. Similarly, RRI represents RIR, IRR and itself, CRI represents CIR, IRC, ICR, RIC, RCI and itself, and so on. Any of the processes can move from the initial
2.6. Choices of the group

Figure 2.4: The annotated quotient structure for the resource controller [11]

state to a request state. Hence the initial state has three successors in the original model, namely RII if process 0 moves to R, IRI if process 1 moves to R and IIR if process 2 moves to R. These three successor states are all represented by RII in the annotated quotient structure. The three edges leading from the initial state in figure 2.4 correspond to the three enabled transitions from III in the original model which have just been described. The transition from the representative state III to the non-representative state IRI is given by the edge labelled \( \pi_{01}, 1 \) which indicates that process 1 has an enabled transition and its execution leads to a state in the same orbit as RII. Since the initial state and RII are both representative states the edge indicating that process 0 has an enabled transition is annotated with the identity.
Chapter 3

Symmetry Reductions in the Context of Category Theory

The model which will be used in the remainder of this thesis, called a language system \([27][28][29]\), will be described in this chapter. The rest of the chapter presents new material. A new category of language systems, denoted \(\text{LS}\), is presented. The morphisms in category \(\text{LS}\) are defined in a non-restrictive way and consequently are consistent with the definition of net morphism which will be studied in chapter 6. Restrictions on the definition of morphism are given which result in several wide subcategories of \(\text{LS}\). We then consider permutations of language systems which define the notion of symmetry for the model. A symmetry-reduction, or quotient structure, of a language system can then be described. We start with the definition of a category.

3.1 Category Theory

This section establishes the terminology of the category theory used in this thesis. An explanation of all the category theory concepts relevant to computer science may be found in \([2]\).

A category consists of a class of objects together with a class of morphisms. Typically a morphism is a structure preserving map between two objects. Category theory studies
structural aspects of mathematics and most examples of a mathematical structure
together with the appropriate morphisms yield a category. For example, sets with
functions constitute a category, as do groups with group homomorphisms and vector
spaces with linear maps. It is important to note that what characterises a category is
its morphisms and not only its objects, as will be demonstrated in section 3.2 where
several categories will be defined which have the same objects. The formal definition
of a category will now be given.

Definition 3.1.1 A category $\mathcal{C}$ consists of a class of objects and in addition for any
pair $a, b$ of objects there is a set of morphisms from $a$ to $b$, i.e. having $a$ as domain and
$b$ as codomain, denoted $\text{Hom}(a, b)$, and

1. given a morphism $g : a \to b$ in $\text{Hom}(a, b)$ and a morphism $f : b \to c$ in $\text{Hom}(b, c)$
   there is a morphism $f \circ g : a \to c$ in $\text{Hom}(a, c)$. (This binary operation $\circ$ is called
   composition.)
2. composition of morphisms is associative: $f \circ (h \circ g) = (f \circ h) \circ g$.
3. for each object $a$ there is a morphism $\text{Id}_a : a \to a$ called the identity on $a$.
4. given any $f$ in $\text{Hom}(a, b)$, $f \circ \text{Id}_a = \text{Id}_{b} \circ f = f$

Isomorphism in the categorical sense will now be defined.

Definition 3.1.2 A morphism $f : b \to c$ is an isomorphism in a category $\mathcal{C}$

$\iff \exists g : c \to b$ such that $g \circ f = \text{Id}_b$ and $f \circ g = \text{Id}_c$

Category theory can be used to show how different kinds of structures are related to
each other. This is achieved using functors which are, essentially, structure preserving
maps between categories. Given categories $\mathcal{C}$ and $\mathcal{D}$, a functor $H$ from $\mathcal{C}$ to $\mathcal{D}$ sends
objects of $\mathcal{C}$ to objects of $\mathcal{D}$ and morphisms of $\mathcal{C}$ to morphisms of $\mathcal{D}$ in such a way
that it preserves the composition of morphisms and identity morphisms. A functor is
formally defined as follows:
Definition 3.1.3 A functor $H$ from $C$ to $D$ consists of a pair of functions $H_{\text{obj}}$ (mapping objects to objects) and $H_{\text{mor}}$ (mapping morphisms to morphisms) for which

1. if $g : a \rightarrow b$ in $C$ then $H_{\text{mor}}(g) : H_{\text{obj}}(a) \rightarrow H_{\text{obj}}(b)$ in $D$

2. for any object $a$ of $C$, $H_{\text{mor}}(\text{Id}_a) = \text{Id}_{H_{\text{obj}}(a)}$

3. if $g \circ f$ is defined in $C$, then $H_{\text{mor}}(g) \circ H_{\text{mor}}(f)$ is defined in $D$ and $H_{\text{mor}}(g \circ f) = H_{\text{mor}}(g) \circ H_{\text{mor}}(f)$.

3.2 Language Systems and the Category LS

In this section a category of language systems is defined in which the mappings between the objects are (language) homomorphisms.

Vector languages were introduced by Shields [27] [28] [29]. They represent the behaviour of a system by an $n$-tuple of sequences, each of which records the actions performed by one of its constituent processes. The vector language is the set of all behaviour representations of the system. Some preliminaries are required before a precise definition of a vector language can be given.

Definition 3.2.1 Define a language system to be a tuple $LS = (I, A, L, \alpha)$ where

$I = [1 : n]$ is a set of processes

$A$ is the set of actions

$\alpha : I \rightarrow \mathcal{P}(A)$ such that $\bigcup_{i \in I} \alpha(i) = A$

$L : I \rightarrow \mathcal{P}(A^*)$ such that $L_i \subseteq (\alpha(i))^*$

The set $\alpha(i)$ represents the actions associated with the process $i$ and is called the alphabet of $i$. The set $L_i$ comprises strings over the alphabet of $i$ and is called the string language of $i$. $L_i$ is process $i$'s view of the behaviour of the system, and is determined by the constraints associated with process $i$. 
Definition 3.2.2 For each action \( a \in A \), the \( \alpha \)-event vector of \( a \), denoted \( a_\alpha \), is defined as

\[
a_\alpha(i) = \begin{cases} 
a & \text{if } a \in \alpha(i) \\
\Lambda & \text{if } a \notin \alpha(i)\end{cases}
\]

where \( \Lambda \) is the empty sequence. The set of event vectors is denoted by \( A_\alpha = \{ a_\alpha \mid a \in A \} \).

An event vector represents the simultaneous execution of the event by each participating process. Event vectors may be concatenated. If \( a_1, \ldots, a_n \in A \) then \( a_1 \ldots a_n \) is the function \( \varphi : I \to A^* \) such that \( \varphi(i) = a_1(i) \ldots a_n(i) \). Denote the set of all such \( \varphi \) by \( A^*_\alpha \).

For example, if \( A = \{a, b, c, d\} \) and \( I = \{1, 2, 3\} \) with alphabets \( \alpha(1) = \{a, b\} \), \( \alpha(2) = \{b, c\} \) and \( \alpha(3) = \{a, d\} \) the set of event vectors is

\[
A_\alpha = \left\{ \begin{pmatrix} a \\ \Lambda \end{pmatrix}, \begin{pmatrix} b \\ \Lambda \end{pmatrix}, \begin{pmatrix} \Lambda \\ c \end{pmatrix}, \begin{pmatrix} \Lambda \\ \Lambda \end{pmatrix}, \begin{pmatrix} \Lambda \\ d \end{pmatrix} \right\} = \{a, b, c, d\}
\]

Then \( \varphi = a.b.c.a.d \in A^*_\alpha \) and \( \varphi = \begin{pmatrix} aba \\ bc \\ aad \end{pmatrix} \)

Note that \( A^*_\alpha \) is a monoid, i.e. a set with an associative binary operation under which it is closed (in this case \( \cdot \) ) together with an identity element (the vector such that \( \Lambda(i) = \Lambda \forall i \in I \)). The operation \( \cdot \) is defined as follows. If \( \varphi, \psi \in A^*_\alpha \), then their concatenation \( \varphi \cdot \psi \) is defined by \( (\varphi \cdot \psi)(i) = \varphi(i) \cdot \psi(i) \), for each \( i \in I \). If \( X \) is a set of sequences or a set of event vectors then \( \downarrow X \) denotes the prefix closure of \( X \).

Two actions \( a \) and \( b \) are potentially concurrent if no process can execute both of them. Formally this may be stated as \( a \in \alpha(i) \implies b \notin \alpha(i) \), all \( i \). Note that this is the case if and only if \( a.b = b.a \), so, algebraically, two events are potentially concurrent exactly when they commute in the \( \cdot \) operation. This actually defines the relation \( \cdot \) on \( A \) which is the independency relation on the set of Mazurkiewicz traces over \( A \). Indeed, there is an isomorphism between the monoid \( A^*_\alpha \) and the monoid \( A^*_\alpha \) [19] [29].
In the last example the actions $a$ and $c$ are potentially concurrent since

$$a \cdot c = \begin{pmatrix} a \\ \Lambda \\ a \end{pmatrix} \cdot \begin{pmatrix} \Lambda \\ c \\ \Lambda \end{pmatrix} = \begin{pmatrix} a \\ c \\ a \end{pmatrix} \cdot \begin{pmatrix} \Lambda \\ c \\ \Lambda \end{pmatrix} = c \cdot a$$

The actions $b$ and $d$ in the same example are also potentially concurrent, as are $c$ and $d$.

**Definition 3.2.3** The vector language of a system is

$$\mathcal{L} = \{ \underline{x} \in A^*_i \forall i \in I : \underline{x}(i) \in L_i \}$$

Note that it is not possible to simplify the definition 3.2.1 of a language system by assuming that $\alpha(i)$ is exactly those elements of $A$ that appear in the strings of $L_i$ since this simplification may change the behaviour of the language system. This can be seen by considering a simple language system that has only two processes, with alphabets $\alpha(1) = \{a, b\}$ and $\alpha(2) = \{b\}$ and string languages $L_1 = a^*$ and $L_2 = b^*$. If we were to make the simplification of removing $b$ from $\alpha(1)$ so as to make it comprise only the actions that appear in $L_1$, the vector language of the system would be $\downarrow (a b)^*$. However the true behaviour of the system is $\downarrow (a)^*$ since

$$a \cdot b = \begin{pmatrix} a \\ \Lambda \\ a \end{pmatrix} \cdot \begin{pmatrix} b \\ b \end{pmatrix} = \begin{pmatrix} ab \\ b \end{pmatrix}$$

and $ab \not\in L_1$. The appearance of $b$ in $\alpha(1)$ effectively ‘stops’ $b$ ever occurring when the two processes run in parallel.

It is also worth noting here that, in common with many other language based models, vector languages cannot cope with non-determinism. This problem has been overcome in other models by adding extra structure, e.g. refusal sets in CSP [12]. The vector language model may also be similarly enhanced, [30].

Consider two language systems $LS$ and $LS'$, where $LS' = (I', A', L', \alpha')$. (Note that in this thesis the elements of any tuple will similarly inherit the subscript or dash.) A morphism between these two language systems is simply a function on their alphabets.
and a relation on the indexing set which send strings in one language to strings in another. If we define a morphism in the following way it is possible to demonstrate the existence of a category of language systems.

**Definition 3.2.4** A LS homomorphism $\phi : LS \rightarrow LS'$ is a pair $(\phi_I, \phi_A)$ comprising a relation

$$\phi_I : I \leftrightarrow I'$$

and a function

$$\phi_A : A \rightarrow A'$$

such that $\forall i \in I$ and any $i' \in I'$ for which $(i, i') \in \phi_I$:

$$\phi_A(\alpha(i)) \subseteq \alpha'(i') \quad (3.2.1)$$

$$\phi_A(L_i) \subseteq L_i' \quad (3.2.2)$$

Note that in (3.2.1) the function $\phi_A$ is extended to a function mapping $P(A) \rightarrow P(A')$ in the natural way, and to a function mapping $P(A^*) \rightarrow P(A'^*)$ in (3.2.2).

**Proposition 3.2.1** For any two LS homomorphisms $\theta = (\theta_I, \theta_A)$ and $\phi = (\phi_I, \phi_A)$:

$$\theta : LS' \rightarrow LS''$$

and

$$\phi : LS \rightarrow LS'$$

we have the compositions

$$\theta_I \circ \phi_I : I \leftrightarrow I'' \quad \text{and} \quad \theta_A \circ \phi_A : A \rightarrow A''$$

and we define the composition $\theta \circ \phi = (\theta_I \circ \phi_I, \theta_A \circ \phi_A)$. Then $\theta \circ \phi : LS \rightarrow LS''$ is a LS homomorphism.

**Proof** The composition of two functions/relations is a function/relation so it is sufficient to show that conditions (3.2.1) and (3.2.2) hold for the composition $\theta \circ \phi$.

Let $(i, i'') \in \theta_I \circ \phi_I$, then there exists $i'$ such that $(i, i') \in \phi_I$ and $(i', i'') \in \theta_I$.

$$\theta_A \circ \phi_A(\alpha(i)) = \theta_A(\phi_A(\alpha(i)))$$

$$\subseteq \theta_A(\alpha'(i')) \quad (3.2.1) \text{ for } \phi$$

$$\subseteq \alpha''(i'') \quad (3.2.1) \text{ for } \theta$$
i.e. the composition satisfies (3.2.1).

\[ \theta_A \circ \phi_A(L_i) = \theta_A(\phi_A(L_i)) \]
\[ \subseteq \theta_A(L_i') \quad (3.2.2) \text{ for } \phi \]
\[ \subseteq L_i'' \quad (3.2.2) \text{ for } \theta \]

i.e. the composition satisfies (3.2.2).

\[\square\]

**Proposition 3.2.2** Language systems and their homomorphisms form a category.

**Proof** LS homomorphisms are closed under composition (proposition 3.2.1) and their compositions are associative. The identity is simply the pair \( Id_{LS} = (Id_I, Id_A) \) comprising the identity functions on \( I \) and \( A \).

\[\square\]

**Definition 3.2.5** Let \( LS \) be the category of language systems described above.

### 3.3 Wide Subcategories of LS

**Definition 3.3.1** A LS isomorphism \( \phi : LS \to LS' \) comprises two bijections:

\[ \phi_I : I \to I' \quad \text{and} \quad \phi_A : A \to A' \]

such that, \( \forall i \in I, (3.2.1) \) and (3.2.2) hold for \( \phi \) and \( \phi^{-1} \),

where \( \phi^{-1} = (\phi^{-1}_I : I' \to I, \phi^{-1}_A : A' \to A) \). As \( \phi_I \) is now a bijective function, \( i' = \phi_I(i) \) and \( i = \phi^{-1}_I(i') \).

**Proposition 3.3.1** A morphism in \( LS \) is an isomorphism in the usual categorical sense, see definition 3.1.2, if and only if it is a LS isomorphism, definition 3.3.1.
Proof

If a morphism $\phi : LS \to LS'$ in $LS$ is an isomorphism in the categorical sense then there exists a morphism $\phi^{-1} : LS' \to LS$ satisfying the conditions of $\phi$. Therefore $\phi$ is an LS isomorphism.

If $\phi = (\phi_I, \phi_A)$ is a LS isomorphism then

$$\forall i' \in I', \quad \phi_I \circ \phi_I^{-1}(i') = \phi_I(i) = i'$$

$$\forall i \in I, \quad \phi_I^{-1} \circ \phi_I(i) = \phi_I^{-1}(i') = i$$

hence $\phi_I \circ \phi_I^{-1} = Id_{I'}$ and $\phi_I^{-1} \circ \phi_I = Id_I$. A similar result can be demonstrated for the bijection $\phi_A$ hence $\phi$ is an isomorphism in the categorical sense.

\[\square\]

Proposition 3.3.2 If $\phi : LS \to LS'$ is a LS isomorphism then, $\forall i \in I$:

$$\phi_A(\alpha(i)) = \alpha'(i') \quad (3.3.3)$$

$$\phi_A(L_i) = L_{i'}' \quad (3.3.4)$$

Proof

Since $\phi$ is a LS isomorphism, (3.2.1) and (3.2.2) are true for $\phi^{-1}$, i.e.

$$\phi_A^{-1}(\alpha'(i')) \subseteq \alpha(i) \quad (3.3.5)$$

$$\phi_A^{-1}(L_{i'}) \subseteq L_i \quad (3.3.6)$$

Now

$$\alpha'(i') = Id_{A'}(\alpha'(i'))$$

$$= \phi_A \circ \phi_A^{-1}(\alpha'(i'))$$

$$\subseteq \phi_A(\alpha(i)) \quad \text{applying (3.3.5)}$$

(3.3.3) then follows from combining this result with (3.2.1), for $\phi$ a LS isomorphism.

Also

$$L_{i'} = Id_{A'}(L_{i'})$$

$$= \phi_A \circ \phi_A^{-1}L_{i'}$$

$$\subseteq \phi_A(L_i) \quad \text{applying (3.3.6)}$$
(3.3.4) follows from combining this result with (3.2.2).

\[ \square \]

**Remark 3.3.1** This result will not hold for every LS homomorphism for which the \( \phi \)'s are bijections, the inverse \( \phi^{-1} \) must also be a LS homomorphism. In the following example of a bijective LS homomorphism, (3.3.3) and (3.3.4) do not hold because \( \phi^{-1} \) is not a LS homomorphism.

**Example 3.3.1** Let

\[
I = \{1, 2\} = I' \quad \text{and} \quad A = \{a_1, a_2, a_3\} = A'
\]

and

\[
\begin{align*}
\alpha(1) &= \{a_1\} & \alpha'(1) &= \{a_1, a_2\} \\
\alpha(2) &= \{a_2, a_3\} & \alpha'(2) &= \{a_2, a_3\}
\end{align*}
\]

If

\[
\psi : I, A \rightarrow I', A' \quad \text{is such that} \quad \psi_I = \text{Id}_I : I \rightarrow I \quad \text{and} \quad \psi_A = \text{Id}_A : A \rightarrow A
\]

then \( \psi \) is a LS homomorphism, since \( \psi_A(\alpha(i)) \subseteq \alpha'(i') \) for \( i = 1, 2 \). But the inverse \( \psi^{-1} : I', A' \rightarrow I, A \) does not satisfy (3.2.1) since \( \{a_1, a_2\} = \psi_A^{-1}(\alpha'(1)) \not\subseteq \alpha(1) = \{a_1\} \), so is not a LS homomorphism.

**Definition 3.3.2** A LS automorphism is a LS isomorphism which maps from LS to LS. In this case

\[
\phi_I : I \rightarrow I \quad \text{and} \quad \phi_A : A \rightarrow A
\]

are both permutations since they are one-one and onto functions of a set to itself.

**Remark 3.3.2** LS automorphisms satisfy definition 2.1.1 and hence form a group.

The changes to the definition of the LS homomorphism which resulted in the LS isomorphisms and LS automorphisms allow two further categories to be defined. These
are both wide subcategories of LS. A wide subcategory D of a category C comprises the same objects as C but the arrows of D are subsets of the sets of arrows of C. The set of LS isomorphisms is closed under function composition, as is the set of LS automorphisms. In both cases the identity is the same as the identity in LS. This allows two further categories to be defined. Both of which have language systems as objects, one has LS isomorphisms as arrows and the other has LS automorphisms. These are both wide subcategories of LS.

3.4 Permutations of Language Systems

A permutation $g$ of a set $X$ is by definition a bijection $g : X \rightarrow X$. The quotient structure, which is defined in the next section, is determined by group $G$, elements of which are permutations of the sets $I$ and $A$. The next proposition proves that if $\sigma : LS \rightarrow LS$ is an LS homomorphism and $\sigma_I$ and $\sigma_A$ are both permutations of the finite sets $I$ and $A$ then $\sigma$ is an LS automorphism, definition 3.3.2. A lemma is needed first.

**Lemma 3.4.1** Let the sets $I$ and $A$ be finite and let $\sigma = (\sigma_I, \sigma_A)$ be pair of permutations of $I$ and $A$ respectively, then

$$\sigma_I^{m!} = Id_I \quad \text{and} \quad \sigma_A^{m!} = Id_A$$

where $m = \max\{|I|, |A|\}$ and $|S|$ denotes the cardinality of $S$.

**Proof** Let $S_I$ and $S_A$ denote the set of permutations of $I$ and $A$ respectively. $|S_I| = |I|!$ and $|S_A| = |A|!$. Hence $|S_I|$ divides $m!$ and $|S_A|$ divides $m!$, so by a corollary of Lagrange's Theorem, $\sigma_I^{m!} = Id_I$ and $\sigma_A^{m!} = Id_A$.

$\Box$

**Proposition 3.4.1** Let the sets $I$ and $A$ be finite and let $\sigma : (I, A, \alpha, L) \rightarrow (I, A, \alpha, L)$ be a LS homomorphism (definition 3.2.4) such that $\sigma_I$ and $\sigma_A$ are both permutations then $\sigma$ is an LS automorphism.
3.4. Permutations of Language Systems

Proof From lemma 3.4.1 we have

\[ \sigma \circ (\sigma^{m|-1}) \big|_I = Id_I \quad \text{and} \quad \sigma \circ (\sigma^{m|-1}) \big|_A = Id_A \]

hence \( \sigma^{m|-1} = \sigma^{-1} \) on both \( I \) and \( A \). But \( \sigma^{m|-1} \) is an LS homomorphism since the composition of morphism in a category is again a morphism in that category (definition 3.1.1), thus \( \sigma^{-1} \) is also an LS homomorphism. Since \( \sigma \) and \( \sigma^{-1} \) are both LS homomorphisms, \( \sigma \) is an LS automorphism.

\[ \square \]

Thus by proposition 3.3.2 we have:

**Corollary 3.4.1** Let the sets \( I \) and \( A \) be finite and let \( \sigma : (I, A, \alpha, L) \rightarrow (I, A, \alpha, L) \) a permutation of \( I \) and \( A \) be a LS homomorphism, then

\[ \sigma_A(\alpha(i)) = \alpha(\sigma_I(i)) \quad (3.4.1) \]

\[ \sigma_A(L_i) = L_{\sigma_I(i)} \quad (3.4.2) \]

It is now worth reconsidering example 3.3.1 in light of corollary 3.4.1. In example 3.3.1, \( \psi = (\psi_I, \psi_A) \) where \( \psi_I \) and \( \psi_A \) are permutations on the finite sets \( I \) and \( A \). However, in this instance \( \psi \) is not an LS isomorphism since \( \psi^{-1} \) is not an LS homomorphism. The difference to notice is that \( \psi : (I, A, \alpha, L) \rightarrow (I', A', \alpha', L') \) and although \( I = I' \) and \( A = A' \), the two language systems have different structure, in particular \( \alpha(1) \neq \alpha'(1) \).

The next example shows that proposition 3.4.1 cannot be extended for \( I \) and \( A \) infinite sets. In the following example \( \sigma : (I, A, \alpha, L) \rightarrow (I, A, \alpha, L) \) is an LS homomorphism (definition 3.2.4) such that \( \sigma_I \) and \( \sigma_A \) are both permutations but the sets \( I \) and \( A \) are infinite, then \( \sigma \) is an not an LS automorphism.

**Example 3.4.1** Let \( I = \{u, v\} \), \( A = \mathbb{Z} \), \( \alpha(u) = 3\mathbb{Z} \cup 3\mathbb{Z} + 1 \) and \( \alpha(v) = 3\mathbb{Z} \cup 3\mathbb{Z} + 2 \), where, for example, \( 3\mathbb{Z} \) is the set of all integer multiples of 3. The permutations \( \sigma_I : I \rightarrow I \) and \( \sigma_A : \mathbb{Z} \rightarrow \mathbb{Z} \) are defined as follows:

\[ \sigma_I(u) = u \quad \text{and} \quad \sigma_I(v) = v \]
\[
\begin{align*}
\sigma_A(3x) &= 3(3x + 2) \\
\sigma_A(6x + 1) &= 3x + 1 \\
\sigma_A(6x + 2) &= 3x + 2 \\
\sigma_A(6x + 4) &= 9x \\
\sigma_A(6x + 5) &= 3(3x + 1)
\end{align*}
\]

where \( x \in \mathbb{Z} \).

Notice that \( \sigma \) satisfies 3.4.1, i.e. \( \sigma_A(\alpha(u)) = \alpha(\sigma_I(u)) \) and \( \sigma_A(\alpha(v)) = \alpha(\sigma_I(v)) \).

However this is not the case for \( \sigma^{-1} \), e.g. take \( 12 \in \alpha(u) \) and notice that \( \sigma_A^{-1}(12) = 11 \notin \alpha(\sigma_I(u)) = \alpha(u) \).

### 3.5 Quotient Structure of the Language System

In this section, the quotient structure of a language system will be defined and some examples given. First, we consider what represents a symmetry of a language system.

The basic premise of the use of symmetries to alleviate the state explosion problem is that the quotient or symmetry-reduced system is constructed in such a way that it embodies the behaviours of the large system from which it was derived. Where this is the case the quotient system preserves the behaviour of the full system and it is possible to infer behavioural properties of the full system by considering the (smaller) quotient system. The next example gives an intuitive idea of a symmetry of a language system.

**Example 3.5.1** Consider a language system with just two processes. The string languages for this system are \( L_1 = \downarrow (ab)^* \) and \( L_2 = \downarrow (ac)^* \). If we were to swap \( c \) for \( b \) in the string language of \( L_2 \) then we would have the string language \( L_1 \). Similarly if we were to swap \( b \) for \( c \) in the string language of \( L_1 \) we would have the string language of \( L_2 \). So, if we additionally swap 1 and 2 then we are left with a system that behaves in the same way as the original system.

What has just been described in example 3.5.1 is a permutation on the set of processes \( I \) and the set of actions \( A \) that leaves the behaviour of the system unchanged. A
symmetry of a language system is therefore a permutation which leaves the string languages unchanged and hence preserves the behaviour of the language system. This is significantly different from the notion of a symmetry presented in chapter 2 which was a permutation that preserved the structure of the state space.

When the sets $I$ and $A$ are finite these permutations are LS automorphisms, proposition 3.4.1. The set of all such LS automorphisms forms a group. We will also be considering subgroups, denoted $G$, of this group. The orbit of $i \in I$ under the action of $G$ is defined to be $[i] = \{\phi_I(i) | \phi \in G\}$. The relation $i_1 \sim i_2$ if and only if $i_2 = \phi_I(i_1)$, some $\phi \in G$, can be shown to be an equivalence relation; each equivalence class being the corresponding orbit. The orbits of $a \in A$ are similarly defined.

**Definition 3.5.1** The quotient structure of a language system LS is $LS_G = (I_G, A_G, \alpha_G, L_G)$ where

- $I_G = \{[i] | i \in I\}$
- $A_G = \{[a] | a \in A\}$
- $\alpha_G : I_G \rightarrow \mathcal{P}(A_G)$ such that $\alpha_G([i]) = \{[a] | a \in \alpha(i)\}$
- $L_G : I_G \rightarrow \mathcal{P}(A_G)$ such that $L_G([i]) = \{[a][b] \ldots | ab\ldots \in L_i\}$

**Proposition 3.5.1** The functions $\alpha_G$ and $L_G$ are well defined.

**Proof**

Each $\phi = (\phi_I, \phi_A) \in G$ is a LS automorphism and so (3.3.3) and (3.3.4) hold. So for $i_1 \sim i_2$ we have $i_2 = \phi_I(i_1)$ and

$$\sigma_A(\alpha(i_1)) = \alpha(i_2) \quad (3.5.3)$$
$$\sigma_A(L_i) = L_j \quad (3.5.4)$$

Considering $\alpha_G$ first it must be proved that

$$\{[a_1] | a_1 \in \alpha(i_1)\} = \{[a_2] | a_2 \in \alpha(i_2)\}$$

Let $a_1 \in \alpha(i_1)$. We have $i_2 = \phi_I(i_1)$, some $\phi \in G$, and $\phi_A(a_1) \in \phi_A(\alpha(i_1)) = \alpha(i_2)$ by (3.5.3). But $a_1 \sim \phi_A(a_1)$, so $[a_1] = [\phi_A(a_1)] \in \{[a_2] | a_2 \in \alpha(i_2)\}$. 


3.5. Quotient Structure of the Language System

It has been shown that \( \{[a_1]a_1 \in \alpha(i_1)\} \subseteq \{[a_2]a_2 \in \alpha(i_2)\} \), the converse inclusion follows by symmetry. Hence \( \alpha_G \) is well defined.

Turning to \( L_G \), it must be proved that

\[
\{[a_1][b_1] \ldots [a_1 b_1 \ldots \in L_{i_1}\} = \{[a_2][b_2] \ldots [a_2 b_2 \ldots \in L_{i_2}\}
\]

Let \( a_1 b_1 \ldots \in L_{i_1} \). We have \( i_2 = \phi_l(i_1) \), some \( \phi \in G \), and

\[
\phi_A(a_1)\phi_A(b_1) \ldots \in \phi_A(L_{i_1}) = L_{i_2} \text{ by (3.5.4)}. \text{ But } a_1 \sim \phi_A(a_1), b_1 \sim \phi_A(b_1), \ldots \text{ so } [a_1][b_1] \ldots = [\phi_A(a_1)][\phi_A(b_1)] \ldots \in \{[a_2][b_2] \ldots [a_2 b_2 \ldots \in L_{i_2}\}.
\]

It has been shown that \( \{[a_1][b_1] \ldots [a_1 b_1 \ldots \in L_{i_1}\} \subseteq \{[a_2][b_2] \ldots [a_2 b_2 \ldots \in L_{i_2}\}, \) the converse inclusion follows by symmetry and hence \( L_G \) is well defined.

The next proposition is required to ensure that the string languages of the quotient language system are correctly defined.

**Proposition 3.5.2** \( L_G([i]) \subseteq (\alpha_G([i]))^* \)

**Proof** Follows from the definitions of \( \alpha_G \) and \( L_G \).

**Definition 3.5.2** For each action \([a] \in A_G\), the \( \alpha_G \)-event vector of \([a]\), denoted \([a]\), is defined as

\[
[a](i) = \begin{cases} [a] & \text{if } [a] \in \alpha_G([i]) \\ [A] & \text{if } [a] \notin \alpha_G([i]) \end{cases}
\]

The set of event vectors is denoted by \( A_{\alpha_G} = \{[a] \mid [a] \in A_G\} \).

**Definition 3.5.3** The vector language of a quotient language system is

\[
L_G = \{[x] \in A_{\alpha_G}^* \mid \forall [i] \in I_G : [x](i) \in L_G([i])\}
\]
3.5. Quotient Structure of the Language System

Some examples of simple language systems and their quotients will now be given.

**Example 3.5.2** Consider a language system $LS$ in which $I = \{1, 2\}$ and $A = \{a, b, c\}$, with alphabets $\alpha(1) = \{a, b\}$, $\alpha(2) = \{a, c\}$ and string languages

$L_1 = \downarrow(ab)^*$, $L_2 = \downarrow(ac)^*$. The only permutation (apart from the identity permutation) which preserves behaviour is

$$\sigma = \begin{pmatrix}
1 & 2 & a & b & c \\
2 & 1 & a & c & b
\end{pmatrix}$$

So $G$ comprises $\sigma$ and the identity permutation. The orbits are

$$[1] = \{1, 2\}, \quad [a] = \{a\} \quad \text{and} \quad [b] = \{b, c\}$$

hence

$$I_G = \{[1]\}, \quad A_G = \{[a], [b]\}, \quad \alpha_G([1]) = \{[a], [b]\} \quad \text{and} \quad L_{G[1]} = \downarrow([a][b])^*$$

**Example 3.5.3** Consider a language system $LS$ in which $I = \{1, 2, 3\}$ and $A = \{a, b, c, d\}$, with alphabets $\alpha(1) = \{a, b\}$, $\alpha(2) = \{a, c\}$, $\alpha(3) = \{c, d\}$ and string languages

$L_1 = \downarrow(ab)^*$, $L_2 = \downarrow(ac)^* \cup \downarrow(ca)^*$, $L_3 = \downarrow(cd)^*$. A permutation which preserves behaviour is

$$\sigma = \begin{pmatrix}
1 & 2 & 3 & a & b & c & d \\
3 & 2 & 1 & c & d & a & b
\end{pmatrix}$$

So $G$ comprises $\sigma$ and the identity permutation. The orbits resulting from the group action of $G$ are

$$[1] = \{1, 3\}, \quad [2] = \{2\}, \quad [a] = \{a, c\} \quad \text{and} \quad [b] = \{b, d\}$$

hence

$$I_G = \{[1], [2]\}, \quad A_G = \{[a], [b]\}, \quad \alpha_G([1]) = \{[a], [b]\}, \quad \alpha_G([2]) = \{[a]\}$$

and

$$L_{G[1]} = \downarrow([a][b])^*, \quad L_{G[2]} = ([a])^*$$
Definition 3.5.4 A quotient morphism $\pi : LS \rightarrow LS_G$ comprises the two functions

$$\pi_I : I \rightarrow I_G \quad \text{and} \quad \pi_A : A \rightarrow A_G$$

such that

$$\pi_I(i) = [i] \quad \text{and} \quad \pi_A(a) = [a]$$

The next proposition proves that the quotient morphism $\pi = (\pi_A, \pi_I)$ is in the category LS since it satisfies properties (3.2.1) and (3.2.2) of an LS homomorphism:

$$\pi_A(\alpha(i)) \subseteq \alpha_G(\pi_I(i)) \quad \text{and} \quad \pi_A(L(i)) \subseteq L_G(\pi_I(i))$$

In fact we have a stronger result, as stated.

Proposition 3.5.3 A quotient morphism $\pi = (\pi_A, \pi_I)$ satisfies

$$\pi_A(\alpha(i)) = \alpha_G(\pi_I(i)) \quad \text{and} \quad \pi_A(L(i)) = L_G(\pi_I(i))$$

hence the following diagram commutes

$$
\begin{array}{ccc}
I & \xrightarrow{\pi_I} & I_G \\
\downarrow & & \downarrow \alpha_G \\
\mathcal{P}(A) & \xrightarrow{\pi_A} & \mathcal{P}(A_G)
\end{array}
$$

Proof

$$\pi_A(\alpha(i)) = \{[a] | a \in \alpha(i)\}$$

$$= \alpha_G([i]) \quad \text{by definition of } \alpha_G$$

$$= \alpha_G(\pi_I(i)) \quad \text{by definition of } \pi$$

$$\pi_A(L(i)) = \{[a][b] \ldots | ab \ldots \in L(i)\}$$

$$= \ L_G([i]) \quad \text{by definition of } L_G$$

$$= \ L_G(\pi(i)) \quad \text{by definition of } \pi$$
Remark 3.5.1 The quotient morphism $\pi = (\pi_I, \pi_A)$ is not in the wide subcategories of $LS$ that were defined in section 3.2 because $\pi$ is not a bijection.

From Proposition 3.5.3, we can conclude that a language system and its quotient are both in the category $LS$. An important question is whether a system and its quotient have similar behaviour, as this is at the heart of using symmetries to relieve the state explosion problem. This question of whether behaviour is preserved between two language systems in the same category (similarly a language system and its quotient) is the subject of the next chapter.
Chapter 4

Behaviour Preservation

The objective of this chapter is to determine the circumstances under which a morphism between two language systems (or a language system and its quotient) preserve behaviour. The morphisms in the categories presented in the last chapter preserve structure, we will now investigate if they preserve behaviour.

Since the behaviour of a language system is given by its vector language, we begin by extending the function $\phi_A$ to a well defined function from vectors in the language of $LS$ to vectors in the language of $LS'$. This function is then used to define a category of vector languages and in order to determine which morphisms preserve behaviour we attempt to define a functor from the category $LS$, or one of its wide subcategories, to the category of vector languages. Counter-examples are presented which show that of all the categories considered so far, only the category in which morphisms are $LS$ isomorphisms is a candidate for the domain of this functor. This then motivates a direct proof of the circumstances under which a morphism preserves behaviour. The construction of a split morphism extends this proof to the projection morphisms which define the quotient system.
4.1 Monoid Morphisms and the Commutativity Condition

Given a LS homomorphism \( \phi : LS \rightarrow LS' \) as presented in definition 3.2.4 we extend the function \( \phi_A \) as follows:

\[
\phi_A : A_\alpha \rightarrow A'_\alpha \quad \text{such that} \quad \phi_A(a) = \phi_A(a)
\]

Recall that the elements of \( A_\alpha \) generate the monoid \( A_\alpha^* \) with the \('\)' operation defined by \((x,y)(i) = x(i) . y(i)\). We may extend \( \phi_A \) to a relation

\[
\phi_A^* : A_\alpha^* \leftrightarrow (A'_\alpha)^*
\]

such that

\[
\prod x \cdot \prod y \quad \text{if} \quad \prod x \cdot \prod y \quad \text{and} \quad \phi_A(a) = b
\]

for \( x, y \in A_\alpha^* \) and \( a, b \in A_\alpha \).

Having defined the relation \( \phi_A^* \), we now give an example.

**Example 4.1.1** For

\[
A_\alpha = \left\{ \begin{pmatrix} a \\ \Lambda \\ \end{pmatrix}, \begin{pmatrix} \Lambda \\ b \end{pmatrix} \right\} \quad \text{and} \quad A'_\alpha = \left\{ \begin{pmatrix} p \\ \Lambda \\ \end{pmatrix}, \begin{pmatrix} q \\ p \end{pmatrix} \right\}
\]

and \( \phi_A(a) = p, \phi_A(b) = q \) the relation gives:

\[
a \cdot b \quad \phi_A^* \quad p \cdot q \quad \text{and} \quad b \cdot a \quad \phi_A^* \quad q \cdot p
\]

Notice that in this example \( a \cdot b = b \cdot a \) so from \( b \cdot a \quad \phi_A^* \quad q \cdot p \) we have \( a \cdot b \quad \phi_A^* \quad p \cdot q \). Hence we have \( a \cdot b \quad \phi_A^* \quad p \cdot q \) and \( a \cdot b \quad \phi_A^* \quad q \cdot p \), which stops us from extending \( \phi_A \) to a function on vectors. The problem arises because \( a \) and \( b \) commute but \( \phi_A(a) \) and \( \phi_A(b) \) do not commute. This problem motivates the next proposition.

**Proposition 4.1.1** If \( \phi_A^* : A_\alpha^* \rightarrow (A'_\alpha)^* \) is a monoid morphism then \( \phi_A \) preserves the commutativity condition:

\[
a \cdot b = b \cdot a \implies \phi_A(a) \cdot \phi_A(b) = \phi_A(b) \cdot \phi_A(a)
\]

(4.1.1)
4.1. Monoid Morphisms and the Commutativity Condition

Proof Since \( a \cdot b = b \cdot a \), we have \( \phi_A^*(a \cdot b) = \phi_A^*(b \cdot a) \). As \( \phi_A^* \) is a monoid morphism, we further have that \( \phi_A^*(a \cdot b) = \phi_A^*(a) \cdot \phi_A^*(b) = \phi_A(a) \cdot \phi_A(b) \). Similarly \( \phi_A^*(b \cdot a) = \phi_A^*(b) \cdot \phi_A^*(a) = \phi_A(b) \cdot \phi_A(a) \). It follows that \( \phi_A(a) \cdot \phi_A(b) = \phi_A(b) \cdot \phi_A(a) \). □

When \( \phi_A \) satisfies the commutativity condition (4.1.1) then \( \phi_A^* \) is a well defined function. We now define a category in which this condition holds.

Definition 4.1.1 A commutativity preserving LS homomorphism \( \phi : LS \rightarrow LS' \) is a pair \((\phi_I, \phi_A)\) comprising a relation

\[
\phi_I : I \leftrightarrow I'
\]

and a function

\[
\phi_A : A \rightarrow A'
\]

such that \( \forall i \in I \) and any \( i' \in I' \) for which \( (i, i') \in \phi_I \):

\[
\phi_A(\alpha(i)) \subseteq \alpha'(i') \quad (4.1.2)
\]

\[
\phi_A(L_i) \subseteq L'_{i'} \quad (4.1.3)
\]

\[
a \cdot b = b \cdot a \quad \Rightarrow \quad \phi_A(a) \cdot \phi_A(b) = \phi_A(b) \cdot \phi_A(a) \quad (4.1.4)
\]

Proposition 4.1.2 For any two commutativity preserving LS homomorphisms \( \theta = (\theta_I, \theta_A) \) and \( \phi = (\phi_I, \phi_A) \):

\[
\theta : LS' \rightarrow LS'' \quad \text{and} \quad \phi : LS \rightarrow LS'
\]

we have the compositions

\[
\theta_I \circ \phi_I : I \leftrightarrow I'' \quad \text{and} \quad \theta_A \circ \phi_A : A \rightarrow A''
\]

and we define the composition \( \theta \circ \phi = (\theta_I \circ \phi_I, \theta_A \circ \phi_A) \). Then \( \theta \circ \phi : LS \rightarrow LS'' \) is a commutativity preserving LS homomorphism.
4.1. Monoid Morphisms and the Commutativity Condition

**Proof** The composition of two functions/relations is a function/relation so it is sufficient to show that conditions (4.1.2), (4.1.3) and (4.1.4) hold for the composition $\theta \circ \phi$.

It follows from proposition 3.2.1 that the composition satisfies (4.1.2) and (4.1.3).

If $a \cdot b = b \cdot a$ then $\phi_A(a) \cdot \phi_A(b) = \phi_A(b) \cdot \phi_A(a)$ by (4.1.4) for $\phi_A$ and so $\theta_A(\phi_A(a)) \cdot \theta_A(\phi_A(b)) = \theta_A(\phi_A(b)) \cdot \theta_A(\phi_A(a))$ by (4.1.4) for $\theta_A$ i.e. the composition satisfies (4.1.4).

□

**Proposition 4.1.3** Language systems and commutativity preserving LS homomorphisms form a category.

**Proof** Commutativity preserving LS homomorphisms are closed under composition (proposition 4.1.2) and their compositions are associative. The identity comprises the identity functions on $I$ and $A$ and clearly satisfies condition (4.1.4).

□

**Definition 4.1.2** Let CP-LS denote the category of language systems described above. It is a wide subcategory of LS.

We now address the question as to whether the quotient morphism is in the commutativity preserving category CP-LS. Since it has already been proved that the quotient morphism satisfies (3.2.1) and (3.2.2), it is sufficient to consider whether the quotient morphism satisfies the commutativity condition (4.1.1). Some definitions are needed first.

**Definition 4.1.3**

$$a \cdot b = b \cdot a \iff a = b$$
where

\[
\forall i \in I, a \neq b \implies \begin{cases} a \in \alpha(i) \land b \notin \alpha(i) \\
\alpha(i) \land b \in \alpha(i) \\
\alpha(i) \land b \notin \alpha(i) \end{cases}
\]

**Definition 4.1.4** We say that \( a, b \in A \) are strongly independent if

\[
\forall i \in I, a \in \alpha(i) \implies b \notin \alpha(i)
\]

**Definition 4.1.5** We say that a permutation \( \sigma = (\sigma_I, \sigma_A) \) is commutable if whenever \( a, b \in A \) are strongly independent then \( a \) and \( \sigma_A(b) \) are strongly independent.

**Proposition 4.1.4** A quotient morphism \( \pi \) associated to a group \( G \) of commutable LS automorphisms \( \sigma \) satisfies (4.1.1), i.e.

\[
a \cdot b = b \cdot a \implies \pi_A(a) \cdot \pi_A(b) = \pi_A(b) \cdot \pi_A(a)
\]

**Proof**

For \( a, b \in A \) we have two possible cases:

- **Case 1:** \( \exists \sigma \in G \) such that \( \sigma_A(a) = b \), i.e. \( a \) and \( b \) are in the same orbit.
- **Case 2:** \( \forall \sigma \in G, \sigma_A(a) \neq b \), i.e. \( a \) and \( b \) are not in the same orbit.

Let \( a \cdot b = b \cdot a \). We consider each of the two cases.

**Case 1:**

\[
\sigma_A(a) = b \implies [a] = [b]
\]

\[
\pi_A(a) = [a] = [b] = \pi_A(b)
\]

Therefore \( \pi_A(a) \cdot \pi_A(b) = \pi_A(b) \cdot \pi_A(a) \) when \( \exists \sigma \in G \) s.t. \( \sigma_A(a) = b \)

**Case 2:**

\[
a \cdot b = b \cdot a \iff a \neq b \text{ or } a = b
\]

If \( a = b \) then \( \pi_A(a) = \pi_A(b) \) and the commutativity condition is satisfied.
For $a \circ b$ we show that $\pi_A(a) \circ \pi_A(b)$. Assume, for a contradiction, that $\pi_A(a) \not\subseteq \pi_A(b)$.

$$\pi_A(a) \not\subseteq \pi_A(b) \implies [a] \land [b] \in \alpha_G([i])$$

$$\implies a \land \sigma_A(b) \in \alpha(i) \quad \text{by definition of } \alpha_G([i])$$

but since all $\sigma \in G$ are commutable, $a \in \alpha(i) \implies \sigma_A(b) \not\in \alpha(i)$ so $\pi_A(a) \not\subseteq \pi_A(b)$. Hence $\pi_A(a)$ and $\pi_A(b)$ commute.

4.2 The Category of Vector Languages

In this section a category of vector languages will be constructed. It will then be proved that there exists a functor from the category of LS isomorphism identified in section 3.3 to this vector language category. This functor will then be used to show that isomorphic systems have identical behaviour. In particular, the action of a group of permutations satisfying the conditions of an LS homorphism will result in a language system with an identical behaviour. First it is necessary to prove that LS isomorphisms, definition 3.3.1, satisfy the commutativity condition (4.1.1). This proof is needed in defining the functor.

**Proposition 4.2.1** LS isomorphisms satisfy the commutativity condition 4.1.1, that is:

$$a \circ b = b \circ a \implies \phi_A(a) \circ \phi_A(b) = \phi_A(b) \circ \phi_A(a)$$

for $\phi = (\phi_A, \phi_I)$ satisfying definition 3.3.1 and $a, b \in A$.

**Proof**

Let $a \circ b = b \circ a$ then $a \circ b$ or $a = b$

It has been shown in proposition 3.3.2 that for a LS isomorphism $\phi = (\phi_A, \phi_I)$,

$$\phi_A(\alpha(i)) = \alpha'(i')$$  \hspace{1cm} (4.2.5)

$$\phi_A(L_i) = L'_i$$  \hspace{1cm} (4.2.6)
4.2. The Category of Vector Languages

Assume $a \neq b$, hence $a \neq b$. From (4.2.5),

$$a \in \alpha(i) \implies \phi_A(a) \in \alpha'(\phi_I(i))$$

and further

$$b \notin \alpha(i) \implies \phi_A(b) \notin \alpha'(\phi_I(i))$$

since $\phi_A$ is one to one

Hence

$$a \in \alpha(i) \land b \notin \alpha(i) \implies \phi_A(a) \in \alpha'(\phi_I(i)) \land \phi_A(b) \notin \alpha'(\phi_I(i))$$

Similarly

$$a \notin \alpha(i) \land b \in \alpha(i) \implies \phi_A(a) \notin \alpha'(\phi_I(i)) \land \phi_A(b) \in \alpha'(\phi_I(i))$$

and

$$a \notin \alpha(i) \land b \notin \alpha(i) \implies \phi_A(a) \notin \alpha'(\phi_I(i)) \land \phi_A(b) \notin \alpha'(\phi_I(i))$$

Therefore

$$a \neq b \implies \phi_A(a) \neq \phi_A(b)$$

and

$$\phi_A(a).\phi_A(b) = \phi_A(b).\phi_A(a) \quad (4.2.7)$$

Assume now that $a = b$, then $\phi_A(a) = \phi_A(b)$ and

$$\phi_A(a).\phi_A(b) = \phi_A(b).\phi_A(a) \quad (4.2.8)$$

Combining (4.2.7) and (4.2.8) we have that

$$a.b = b.a \implies \phi_A(a).\phi_A(b) = \phi_A(b).\phi_A(a)$$

The category of vector languages will now be considered. An example of an object in this category is a vector language $\mathcal{L}$ as defined is section 3.2:

$$\mathcal{L} = \{ \underline{u} \in A^*_\alpha | \forall i \in I : \underline{u}(i) \in L_i \}$$

Candidate morphisms between such objects are the monoid morphisms defined

$$\phi_A^*: A^*_\alpha \rightarrow (A^*_\alpha')^*$$

but which preserve behaviour. We show that it is possible to construct a category using the following definition of morphism.
Definition 4.2.1 The monoid morphism \( \phi_A^* : A^*_a \rightarrow (A'_{a'})^* \) is a vector language morphism if it satisfies:

\[
\phi_A^*(\mathcal{L}) \subseteq \mathcal{L}' \tag{4.2.9}
\]

where \( \mathcal{L} \subseteq A^*_a \) and \( \mathcal{L}' \subseteq (A'_{a'})^* \).

It will now be shown that such morphisms are closed under composition.

Proposition 4.2.2 For any two vector languages morphisms \( \phi_A^* : A^*_a \rightarrow (A'_{a'})^* \) and \( \theta_A^* : (A'_{a'})^* \rightarrow (A''_{a''})^* \) we have the composition \( \theta_A^* \circ \phi_A^* : A^*_a \rightarrow (A''_{a''})^* \). The composition \( \theta_A^* \circ \phi_A^* \) is a vector language morphism.

Proof The composition of two functions is a function so it is sufficient to show that condition (4.2.9) holds for \( \theta_A^* \circ \phi_A^* \).

Let \( x \in \mathcal{L} \),

\[
\theta_A^* \circ \phi_A^*(x) = \theta_A^* (\phi_A^*(x)) = \theta_A^* (x') \quad \text{where } x' \in \mathcal{L}' \quad \text{by (4.2.9) for } \phi_A^* \nonumber
\]

\[
= x'' \quad \text{where } x'' \in \mathcal{L}'' \quad \text{by (4.2.9) for } \theta_A^*
\]

Hence \( \theta_A^* \circ \phi_A^*(x) \subseteq \mathcal{L}'' \) and the composition satisfies (4.2.9).

\[\square\]

Proposition 4.2.3 Vector languages and their morphisms form a category.

Proof Vector language morphisms are closed under composition (proposition 4.2.2) and their compositions are associative (by virtue of being functions/monoid morphisms). The identity is simply the morphism satisfying \( \phi_A^* (x) = x \) for \( x \in A^*_a \).

\[\square\]

Definition 4.2.2 Let \( \mathcal{VL} \) be the category of vector languages described above.
4.2. The Category of Vector Languages

In order to determine when behaviour is preserved within a category we would like to define a functor from the category $LS$ or one of its wide subcategories to the category $VL$. Considering the first part of the definition of a functor 3.1.3 the morphisms of the domain category must be extendable to a well defined function $\phi_A^*$ satisfying definition 4.2.1. The next example shows that it is not always possible to extend an $LS$ homomorphism in this way.

**Example 4.2.1** Consider language systems $LS$ and $LS'$ with

<table>
<thead>
<tr>
<th>$I$</th>
<th>$I'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${i_1, i_2}$</td>
<td>${j_1, j_2}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A$</th>
<th>$A'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a, b, c}$</td>
<td>${p, q, r}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\alpha(i_1)$</th>
<th>$\alpha'(j_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a, b}$</td>
<td>${p, q}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\alpha(i_2)$</th>
<th>$\alpha'(j_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${c}$</td>
<td>${p, q, r}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$L_{i_1} = \downarrow (ab)^*$</th>
<th>$L_{i_2} = c^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L'_{j_1} = \downarrow (pq)^*$</td>
<td>$L'_{j_2} = L'(j_1) \cup r^*$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_\alpha = \left{ \begin{pmatrix} a \ \Lambda \ b \ c \end{pmatrix}, \begin{pmatrix} a \ \Lambda \ b \end{pmatrix}, \begin{pmatrix} a \end{pmatrix} \right}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A'_{\alpha'} = \left{ \begin{pmatrix} p \ p \ q \ q \ r \end{pmatrix}, \begin{pmatrix} p \ q \ r \end{pmatrix}, \begin{pmatrix} \Lambda \ r \end{pmatrix} \right}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$L = \downarrow \left( \begin{pmatrix} ab \ c \end{pmatrix}^* \right)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L' = \downarrow \left( \begin{pmatrix} pq \ pq \end{pmatrix}^* \right)$</td>
</tr>
</tbody>
</table>

Given $\phi : LS \to LS'$ comprising a relation $\phi_I$ and the function $\phi_A$ such that

$$\phi_I = \{(i_1, j_1), (i_1, j_2), (i_2, j_2)\}$$

and

$$\phi_A(a) = p \quad \phi_A(b) = q \quad \phi_A(c) = r$$

Clearly, $\phi = (\phi_I, \phi_A)$ is a $LS$ homomorphism.

Returning to the relation $\phi_A^*$,

$$\phi_A^*(L) = \phi_A^* \downarrow \left( \begin{pmatrix} ab \\ c \end{pmatrix}^* \right)$$

$$= \phi_A^* \downarrow (ab,c)^*$$

$$= \downarrow (p,q,r)^* \cup \downarrow (p,r,q)^* \cup \downarrow (q,p,r)^*$$

$$\subseteq (A'_{\alpha'})^* \text{ but } \not\subseteq L'$$
In this example, the vector language is not preserved and the event vectors commute in LS but their images do not commute in LS'. For example,

$$a \cdot c = c \cdot a \quad \text{but} \quad \phi_A(a) \cdot \phi_A(c) = p \cdot r \neq r \cdot p = \phi_A(c) \cdot \phi_A(a)$$

Even if we restrict $\phi$ so that $\phi_I$ and $\phi_A$ are both bijections, it is still not the case that the relation $\phi_A^*$ preserves vector languages, as the next example shows.

**Example 4.2.2** For language systems LS and LS' let

$$I = \{i_1, i_2\} \quad I' = \{j_1, j_2\}$$

$$A = \{a, b\} \quad A' = \{p, q\}$$

$$\alpha(i_1) = \{a\} \quad \alpha(i_2) = \{b\} \quad \alpha'(j_1) = \{p\} \quad \alpha'(j_2) = \{p, q\}$$

$$L_{i_1} = a^* \quad L_{i_2} = b^* \quad L'_{j_1} = p^* \quad L'_{j_2} = q^* \cup q^* \cdot p$$

$$A^*_\alpha = \left\{ \begin{pmatrix} a \\ \Lambda \end{pmatrix}, \begin{pmatrix} \Lambda \\ b \end{pmatrix} \right\} \quad A'^*_\alpha = \left\{ \begin{pmatrix} p \\ \Lambda \end{pmatrix}, \begin{pmatrix} \Lambda \\ q \end{pmatrix} \right\}$$

$$L = A^*_\alpha \quad L' = q^* \cup q^* \cdot p$$

Let $\phi_I$ and $\phi_A$ be bijective functions such that

$$\phi_I(i_1) = j_1 \quad \phi_I(i_2) = j_2 \quad \text{and} \quad \phi_A(a) = p \quad \phi_A(b) = q$$

Clearly, $\phi = (\phi_I, \phi_A)$ is a LS homomorphism

$$\phi_A(L_{i_1}) = p^* = L'_{\phi_I(i_1)} \quad \text{and} \quad \phi_A(L_{i_2}) = q^* \subseteq L'_{\phi_I(i_2)}$$

but

$$\phi_A^*(L) = \phi_A^*(A^*_\alpha) = (A'^*_\alpha)^* \nsubseteq L'$$

Note that the event vectors in LS commute, that is $a \cdot b = b \cdot a$, but their images in LS' do not commute since $p \cdot q \neq q \cdot p$, which is to say that $\phi_A(a) \cdot \phi_A(b) \neq \phi_A(b) \cdot \phi_A(a)$. It is also worth noting that $\phi = (\phi_I, \phi_A)$ is a bijective function but not an isomorphism in the categorical sense, since $\phi_A^{-1}(\alpha'(j_2)) = \{a, b\} \neq \alpha(i_2)$.

In the next example, the morphism satisfies the commutativity condition and $\phi_I$ and $\phi_A$ are both onto functions.
Example 4.2.3 Consider language systems $LS$ and $LS'$ with

$I = \{i_1, i_2\}$ \hspace{1cm} $I' = \{j_1\}$

$A = \{a, b, c\}$ \hspace{1cm} $A' = \{p, q\}$

$\alpha(i_1) = \{a, b\}$ \hspace{0.5cm} $\alpha(i_2) = \{a, c\}$ \hspace{0.5cm} $\alpha'(j_1) = \{p, q\}$

$L_{i_1} = \downarrow (ab)^*$ \hspace{0.5cm} $L_{i_2} = \downarrow (ac)^*$ \hspace{0.5cm} $L'_{j_1} = \downarrow (pq)^*$

$L = \downarrow (a.b.c)^* \cup \downarrow (a.c.b)^*$ \hspace{0.5cm} $L' = \downarrow (p.q)^*$

Given $\phi : LS \to LS'$ comprising onto functions $\phi_I$ and $\phi_A$ such that

$\phi_I(i_1) = \phi_I(i_2) = j_1$ \hspace{0.5cm} and \hspace{0.5cm} $\phi_A(a) = p$ \hspace{0.5cm} $\phi_A(b) = q$ \hspace{0.5cm} $\phi_A(c) = q$

Clearly, $\phi = (\phi_I, \phi_A)$ is a LS homomorphism and the event vectors $b$ and $c$ commute in $LS$ and their images in $LS'$ also commute, hence $\phi_A^*$ is a well defined function, however

$\phi_A^*(a.b.c) = p.q.q = \varnothing L'$

and so behaviour is not preserved.

So far, in attempting to find a morphism that satisfies satisfying definition 4.2.1, the following counter-examples have been given:

1. An LS homomorphism comprising bijective functions $\phi_I$ and $\phi_A$ where $\phi_A^*$ is a relation (hence commutativity is not preserved).

2. An LS homomorphism comprising onto functions $\phi_I$ and $\phi_A$ where $\phi_A^*$ is a well-defined function (hence commutativity is preserved).

From the first counter-example, we conclude that an LS homomorphism comprising bijective functions that do not satisfy the commutativity condition will not preserve behaviour.

From the second counter-example, we conclude that an LS homomorphism comprising onto functions that preserve commutativity will not preserve behaviour. This case is of particular interest since the functions of the projection morphism are both onto. The other possibilities for $\phi_I$ and $\phi_A$ is that they are one to one functions or bijective.
functions. We can exclude \( \phi_I \) a one to one function as this may result in processes in \( LS' \) that are not in the image of \( \phi_I \) and hence the contribution of such processes to the structure of the language system would not be preserved by the morphism. Similarly, we can exclude \( \phi_A \) a one to one function as this may result in \( p \in A'_{cr} \) that are not in the image of \( \phi_A^* \). Consequently we next consider an example in which commutativity is preserved and \( \phi_I \) is onto and \( \phi_A \) a bijection.

**Example 4.2.4** Consider language systems \( LS \) and \( LS' \) with

\[
\begin{align*}
I &= \{i_1, i_2, i_3\} \\
A &= \{a, b, c\} \\
\alpha(i_1) &= \{a, b\} \\
\alpha(i_2) &= \{a, c\} \\
\alpha(i_3) &= \{b, c\} \\
L_{i_1} &= \downarrow (ab)^* \\
L_{i_2} &= \downarrow (ac)^* \\
L_{i_3} &= \downarrow (bc)^* \\
L &= \downarrow (a,b,c)^*
\end{align*}
\]

Given \( \phi : LS \to LS' \) comprising \( \phi_I \) an onto function and and \( \phi_A \) a bijection such that

\[
\phi_I(i_1) = \phi_I(i_2) = \phi_I(i_3) = j_1 \quad \text{and} \quad \phi_A(a) = 1d_A
\]

The function \( \phi_A^* \) is well defined but does not preserve behaviour:

\[
\phi_A^*(a,b,c) = a,b,c \not= \mathbb{Q}, L'
\]

We next consider a morphism such that \( \phi_I \) is a bijection, \( \phi_A \) is an onto function.

**Example 4.2.5** Consider language systems \( LS \) and \( LS' \) with

\[
\begin{align*}
I &= \{i_1, i_2, i_3\} \\
A &= \{a, b, c\} \\
\alpha(i_1) &= \{a, b\} \\
\alpha(i_2) &= \{b, c\} \\
\alpha(i_3) &= \{a, c\} \\
L_{i_1} &= \downarrow (ab)^* \\
L_{i_2} &= \downarrow (bc)^* \\
L_{i_3} &= \downarrow (ac)^* \\
L &= \downarrow (a,b,c)^*
\end{align*}
\]

Given \( \phi : LS \to LS' \) comprising \( \phi_I \) a bijection and \( \phi_A \) an onto function such that

\[
\phi_I(i_1) = j_1 \quad \phi_I(i_2) = j_2 \quad \phi_I(i_3) = j_3 \quad \text{and} \quad \phi_A(a) = p \quad \phi_A(b) = q \quad \phi_A(c) = q
\]
4.2. The Category of Vector Languages

The function \( \phi_A^* \) is well defined but does not preserve behaviour:

\[
\phi_A^*(a.b.c) = p.q.q = \not\subseteq L'
\]

We now restrict the definition of morphism still further so that commutativity is preserved and \( \phi = (\phi_I, \phi_A) \) is an LS homomorphism with \( \phi_I \) and \( \phi_A \) both bijective functions.

Example 4.2.6 Consider language systems \( LS \) and \( LS' \) with

- \( I = \{i_1, i_2\} \)
- \( I' = \{j_1, j_2\} \)
- \( A = \{a, b\} \)
- \( A' = \{p, q\} \)
- \( \alpha(i_1) = \{a, b\} \quad \alpha(i_2) = \{a\} \)
- \( \alpha'(j_1) = \{p, q\} \quad \alpha'(j_2) = \{p, q\} \)
- \( L_{i_1} = \downarrow (ab)^* \quad L_{i_2} = a \)
- \( L'_{j_1} = \downarrow (pq)^* \quad L'_{j_2} = p \cup q \)
- \( \mathcal{L} = (a.b) \quad \mathcal{L}' = p \)

Given \( \phi: LS \rightarrow LS' \) comprising \( \phi_I \) and \( \phi_A \) both bijections such that

\[
\phi_I(i_1) = j_1 \quad \phi_I(i_2) = j_2 \quad \text{and} \quad \phi_A(a) = p \quad \phi_A(b) = q
\]

The function \( \phi_A^* \) is well defined but does not preserve behaviour:

\[
\phi_A^*a.b = p.q = \not\subseteq L'
\]

Note that although the functions \( \phi_I \) and \( \phi_A \) in the last example are both bijections, \( \phi = (\phi_I, \phi_A) \) is not an LS isomorphism, e.g. \( \phi_A^{-1}(\alpha'(j_2)) \not\subseteq \alpha(i_2) \).

The results of the preceding examples will now be summarised. It has been shown that morphisms in the category \( LS \), i.e. LS homomorphisms \( \phi \), may not always be extended to a function \( \phi_A^* \) which preserves behaviour, the required property of morphism in the category \( VL \), even if \( \phi \) comprises \( \phi_I \) and \( \phi_A \) which are bijections. Similarly, it has been shown that morphisms in the category \( CP-LS \) may not always be extended to a function \( \phi_A^* \) which preserves behaviour. This suggests we need to consider restricting the definition of an LS homomorphism still further, for example requiring \( \phi_A \) to be a bijection on the alphabets. However, the next example shows that an LS homomorphism which for which \( \phi_A \) and \( \phi_I \) are both onto and \( \phi_A \) a bijection on the alphabets may also not be extended to \( \phi_A^* \) satisfying 4.2.1.
Example 4.2.7 Consider language systems $LS$ and $LS'$ with

- $I = \{i_1, i_2\}$
- $I' = \{j_1\}$
- $A = \{a, b\}$
- $A' = \{p, q\}$
- $\alpha(i_1) = \{a, b\}$, $\alpha(i_2) = \{b, c\}$
- $\alpha'(j_1) = \{p, q\}$
- $L_{i_1} = \downarrow (ab)^*$
- $L_{i_2} = \downarrow (cb)^*$
- $L'_{j_1} = \downarrow (pq)^*$

Given $\phi : LS \rightarrow LS'$ comprising $\phi_I$ and $\phi_A$ both onto functions such that

- $\phi_I(i_1) = j_1$, $\phi_I(i_2) = j_1$ and $\phi_A(a) = \phi_A(c) = p$, $\phi_A(b) = q$

and $\phi_A : \alpha(i) \rightarrow \alpha'(\phi_I(i))$ a bijection such that

- $\phi_A(\alpha(i_1)) = \phi_A(\{a, b\}) = \{p, q\}$ and $\phi_A(\alpha(i_2)) = \phi_A(\{b, c\}) = \{p, q\}$

the function $\phi_A^*$ is well defined but does not preserve behaviour:

- $\phi_A^* (a.c.b) = p.p.q = \notin L' \cdots (p.q)^*$

We conclude that the domain of a functor to the category of vector languages can not be any of the subcategories of $LS$ considered in the above examples. However, it is possible to define a functor from the wide subcategory of $LS$ in which the morphisms are $LS$ isomorphisms, definition 3.3.1, to the category $VL$, as the next proposition proves.

Proposition 4.2.4 The mapping $LV : LS \rightarrow VL$ comprising $LV_{\text{obj}}$ which maps language systems to vector languages and $LV_{\text{mor}}$ which maps $LS$ isomorphisms to vector language morphisms satisfies the definition 3.1.3 so is a functor. The domain of this functor is the wide subcategory of $LS$ in which the morphisms are $LS$ isomorphisms.

Proof The mapping $LV_{\text{obj}}$ is satisfied by definition 3.2.3.

An $LS$ isomorphism $\phi = (\phi_A, \phi_I)$ preserves commutativity, proposition 4.2.1, so may be extended to the monoid morphism $\phi_A^* : A^*_\alpha \rightarrow (A^*_\alpha)'^*$. An $LS$ isomorphism also satisfies the vector language morphism condition 4.2.9, proposition 3.3.2.

□
4.2. The Category of Vector Languages

For LS isomorphisms \( \phi \) and \( \theta \), \( \phi \circ \theta = Id \). The functor \( LV \) ensures that isomorphisms in LS map to isomorphisms in VL:

\[
LV_{\text{mor}}(\phi) \circ LV_{\text{mor}}(\theta) = LV_{\text{mor}}(\phi \circ \theta) \quad \text{by definition of a functor}
\]

\[
= LV_{\text{mor}}(\text{Id}) = \text{Id}
\]

The behavioural conclusion to be drawn from this is that isomorphic language systems have identical behaviour. Hence if \( \phi : LS \rightarrow LS' \) is an LS isomorphism then \( L = L' \) (where \( L \) is the vector language of the language system LS, etc).

Recall that permutations on finite sets \( I \) and \( A \) satisfying the conditions (3.2.1) and (3.2.2), are LS automorphisms, proposition 3.4.1. Hence choosing permutations satisfying (3.2.1) and (3.2.2) ensures behavioural preservation, so these permutations represent a symmetry of the system. This suggests an automatic method for finding symmetries of a language system, i.e. if \( I \) and \( A \) are finite then, for any permutation on \( I \) and \( A \), it is only necessary to check that 3.2.1 and 3.2.2 are satisfied to conclude that the permutation is a symmetry of the language system.

The examples in section 3.5 will now be reconsidered in the context of preservation of languages between a language system and its quotient. Proposition 3.5.3 tells us that the quotient morphism functions are in the category LS. Therefore in order for \( \pi : LS \rightarrow LS_G \) to be extended to a well defined monoid morphism between a language system and its quotient, \( \pi \) must satisfy the commutativity condition 4.1.1.

In example 3.5.2, we have

\[ b.c = c.b \quad \text{and} \quad \pi_A(b).\pi_A(c) = \pi_A(c).\pi_A(b) \]

thus \( \pi \) satisfies the commutativity condition 4.1.1. However, in example 3.5.3, condition 4.1.1 does not hold. The functions which make up the quotient morphism, \( \pi_I \) and \( \pi_A \), are onto functions. The quotient morphism satisfies the properties of an LS homomorphism, proposition 3.5.3. It has been shown (example 4.2.3) that even with commutativity preservation, LS homomorphisms comprising onto functions on the sets \( I \) and \( A \) do not satisfy the definition of a VL morphism and hence do not preserve
behaviour. Hence the quotient morphism, as currently defined, does not preserve behaviour and so it would not be possible to infer behaviour properties of a language system from its quotient.

In attempting to define a morphism from the category of language systems to the category of vector languages, we have shown in this section that the properties of an LS homomorphism, (3.2.1) and (3.2.2), do not preserve behaviour between two language systems. This motivates a direct proof of the conditions that a LS homomorphism must satisfy to preserve behaviour, which is the subject of the next section.

4.3 A Proof of Behaviour Preservation

This section begins with an informal discussion of behaviour preservation which motivates the conditions required for the proofs that follow.

Let $\phi : LS \to LS'$ be an LS homomorphism which satisfies the commutativity condition. If $x = a_1 \ldots a_n$ is in the vector language of the language system $LS$, then by definition, $x(i) \in L_i$ for any $i \in I$ where $x(i) = a_1(i) \ldots a_n(i)$.

Now

$$\phi_A^*(x)(i') = \phi_A^*(a_1 \ldots a_n)(i')$$

$$= (\phi_A(a_1) \ldots \phi_A(a_n))(i')$$

$$= \phi_A(a_1)(i') \ldots \phi_A(a_n)(i')$$

The $\alpha$-event vector of $\phi_A(a_1)$, is defined as

$$\phi_A(a_1)(i') = \begin{cases} 
\phi_A(a_1) & \text{if } \phi_A(a_1) \in \alpha'(i') \\
\Lambda & \text{if } \phi_A(a_1) \not\in \alpha'(i')
\end{cases}$$

Denote $\phi_A(a_1)(i') \ldots \phi_A(a_n)(i')$ by $\phi_A(a_{1'})\phi_A(a_{2'}) \ldots \phi_A(a_{k'})$ where $k \leq n$. From the definition of the $\alpha$-event vector above, it is clear that each $\phi_A(a_{j'})$ represents a $\phi_A(a_j) \in \alpha'(i')$. 
We know that \( \phi_A^*(x) \in (A'_\alpha)^* \) but for the (well defined) function \( \phi_A^* \) to preserve behaviour, i.e. \( \phi_A^*(L) \subseteq L' \), we require

\[
\phi_A^*(x)(i') = \phi_A(a_{1i'})\phi_A(a_{2i'})\ldots\phi_A(a_{ki'}) \in L'_i \quad \forall i' \in I' \quad (4.3.10)
\]

To illustrate this discussion let us return to example 4.2.5. For \( a,b,c \in L \), we have

\[
\phi_A^*(a\cdot b\cdot c) = p\cdot q\cdot q = \begin{pmatrix} p & q & q \\ \Lambda & q & q \\ p & q & q \end{pmatrix}
\]

so the \( j_1 \)-th component of \( \phi_A^*(a\cdot b\cdot c) \), denoted \( \phi_A(a_{j_1})\phi_A(b_{j_1})\phi_A(c_{j_1}) \) is \( pqq \), similarly the \( j_2 \)-th component denoted \( \phi_A(a_{j_2})\phi_A(b_{j_2})\phi_A(c_{j_2}) \) is \( qqq \) and the \( j_3 \)-th component, \( \phi_A(a_{j_3})\phi_A(b_{j_3})\phi_A(c_{j_3}) \), \( pqq \). In this example, only the \( j_2 \)-th component is an element of its corresponding string language in \( L' \), which is expected since \( \phi_A^*(L) \not\subseteq L' \) in this example.

This discussion of behaviour preservation will now be formalised, after some definitions.

\[ \text{Definition 4.3.1} \]

Let \( \phi = (\phi_I, \phi_A) \) be an LS homomorphism such that the commutativity condition holds for \( \phi_A : A_\alpha \rightarrow A'_\alpha \), yielding \( \phi_A^* : A_\alpha^* \rightarrow (A'_\alpha)^* \) a well defined function and for \( x \in L \), \( \phi_A^*(x) \subseteq (A'_\alpha)^* \).

Define

\[
L_\phi(i') = \begin{cases}
\bigcup_{i \neq i'} \{ \phi_A(x) | x \in L_i \} \\
\{ \Lambda \} \text{ if no } i \in I \text{ exists such that } i \neq i' \end{cases} \quad (4.3.11)
\]

By property (3.2.2) of a LS homomorphism, \( L_\phi(i') \subseteq L'_i \).

The resulting vector language is defined as \( L'_\phi = \{ y \in (A'_\alpha)^* | y(i') \in L_\phi(i') \} \)

These new definitions are included in figure 4.1. The lower part of this figure refers to the possibility of extending \( \phi_A : A_\alpha \rightarrow A'_\alpha \) to a well-defined function \( \phi_A^* : L \rightarrow (A'_\alpha)^* \) if the commutativity condition holds. Our objective is to determine when \( \phi_A^*(L) \subseteq L' \) and in the informal discussion at the start of this section, the condition required to achieve this was identified, \( (4.3.10) \). The vector language \( L_\phi \) has been introduced in order to facilitate our objective. In the next lemma we see that, under certain conditions, \( \phi_A^*(L) \subseteq L'_\phi \).
Lemma 4.3.1 For $\phi$ and $L_\phi^*$ as described in definition 4.3.1,

$$\phi_A^*(L) \subseteq L_\phi^* \iff \\
\forall \bar{x} \in L \text{ of the form } \bar{x} = a_1 \ldots a_n \\
\text{if } \phi_A^*(a_1 \ldots a_n) = \phi_A(a_{1'})\phi_A(a_{2'}) \ldots \phi_A(a_{k'}) \neq \Lambda \\
\text{then } \exists i \in I \text{ and } b_1b_2\ldots b_k \in L_i \\
\text{such that} \\
\phi_i i' \text{ and } \phi_A(b_n) = \phi_A(a_{n'})$$

Proof

In the $\Rightarrow$ direction:

Let $\bar{x} = a_1 \ldots a_n \in L$ then by hypothesis

$$\phi_A^*(\bar{x}) = \phi_A^*(a_1 \ldots a_n) \subseteq L_\phi^*$$
the $i'$-th component of which, some $i' \in I'$, is of the form $\phi_A(a_{1i'})\phi_A(a_{2i'})\ldots\phi_A(a_{ki'})$ and is an element of $L_\phi(i')$, also by hypothesis. Hence, by definition of $L_\phi(i')$, there exists $i \in I$ and $b_1b_2\ldots b_k \in L_i$ such that $i\phi_Ii'$ and $\phi_A(b_n) = \phi_A(a_{ni'})$.

In the $\iff$ direction:

To show $\phi_A^*(L) \subseteq L'_\phi$, it must be proved that $\phi_A^*(x)(i') \in L_\phi(i') \ \forall i' \in I', x \in L$.

By hypothesis, $\forall i' \in I', \exists i \in I$ such that $i\phi_Ii'$.

Now

$$\phi_A^*(x)(i') = \phi_A(a_{1i'})\phi_A(a_{2i'})\ldots\phi_A(a_{ki'})$$
$$= \phi_A(b_1)\phi_A(b_2)\ldots\phi_A(b_k) \ \text{by hypothesis}$$
$$= \phi_A(b_1b_2\ldots b_k) \ \text{where } b_1b_2\ldots b_k \in L_i$$
$$\in L_\phi(i')$$

Hence $\phi_A^*(L) \subseteq L'_\phi$.

To clarify how the conditions in lemma 4.3.1 correspond to the requirements for behaviour preservation set out in the preceding informal discussion, consider the proof in the $\iff$ direction. Notice that the existence of $i \in I$ and $b_1b_2\ldots b_k \in L_i$ such that $i\phi_Ii'$ and $\phi_A(b_n) = \phi_A(a_{ni'})$ ensures a result analogous to condition (4.3.10) in the informal discussion. In order for these criteria to be present, the function $\phi_I : I \to I'$ must be onto. Further we require $\phi_A : \alpha(i) \to \alpha(\phi_I(i))$ to be a bijection. In the next proposition it is proved that these requirements are needed to ensure that $\phi$ satisfies lemma 4.3.1.

**Proposition 4.3.1** If

$$\phi_A : A \to A' \ \text{is a bijection} \ \ (4.3.12)$$
$$\phi_I : I \to I' \ \text{is onto} \ \ (4.3.13)$$

and for all $i \in I$ $\phi_A : \alpha(i) \to \alpha(\phi_I(i))$ is a bijection $\ \ (4.3.14)$

then

$$\phi = (\phi_I, \phi_A) \text{ satisfies lemma 4.3.1}$$
4.3. A Proof of Behaviour Preservation

Proof

Let \( x = a_1 \ldots a_n \in \mathcal{L} \) and let \( i' \) be an element of \( I' \) such that \( \phi_A^*(a_1 \ldots a_n)(i') \neq \Lambda \).

Without loss of generality, let \( \phi_A^*(a_1 \ldots a_n)(i') = \phi_A(a_1'i') \ldots \phi_A(a_k'i') \) where \( k \leq n \).
Then since \( \phi_I \) is onto we have \( i \in I \) such that \( \phi_I(i) = i' \).

Further since \( \phi_A : \alpha(i) \rightarrow \alpha(\phi_I(i)) \) is a bijection we have a unique element \( b_r \in \alpha(i) \) such that \( \phi_A(b_r) = \phi_A(a_r'i') \), where \( r = 1 \ldots k \). As \( \phi_A : A \rightarrow A' \) is a bijection we have \( b_r = a_r'i' \).

It will now be shown that \( b_1 \ldots b_k = (a_1 \ldots a_n)(i) \) to prove that \( b_1 \ldots b_k \in L_i \). We have \( b_1 \ldots b_k \in \alpha(i)^* \) and \( (a_1 \ldots a_n)(i) \in L_i \subseteq \alpha(i)^* \) (by definition of \( \mathcal{L} \)) such that

\[
\phi_A(b_1 \ldots b_k) = \phi_A^*(a_1 \ldots a_n)(i') \\
= \phi_A(a_1) \ldots \phi_A(a_n)(i') \\
= \phi_A(a_1) \ldots \phi_A(a_r) \text{ where each } \phi_A(a_1), \ldots, \phi_A(a_r) \in \alpha'(i') \text{ and } r \leq n \\
= \phi_A(a_1 \ldots a_r) \\
= \phi_A[(a_1 \ldots a_n)(i)] \text{ since } \{a_1, \ldots, a_r\} = \alpha(i) \text{ for } i \text{ such that } \phi_I(i) = i'
\]

It has been shown that \( \phi_A(b_1 \ldots b_k) = \phi_A[(a_1 \ldots a_n)(i)] \). Since \( \phi_A \) is one to one, \( b_1 \ldots b_k = (a_1 \ldots a_n)(i) \in L_i \).

\[\square\]

Example 4.2.7 demonstrates that the conditions stated in proposition 4.3.1 cannot in general be weakened to a bijection on alphabets with \( \phi_A : A \rightarrow A' \) not a bijection. Example 4.2.4 shows that the conditions cannot be weakened to \( \phi_A : A \rightarrow A' \text{ a bijection but } \phi_A \text{ not a bijection on alphabets.} \)

Together, lemma 4.3.1 and proposition 4.3.1 give the conditions under which \( \phi \) an LS homomorphism preserves behaviour. It has been proved that if \( \phi : \mathcal{L}S \rightarrow \mathcal{L}S' \) satisfies

\[
\phi_A : A \rightarrow A' \text{ is a bijection} \\
\phi_I : I \rightarrow I' \text{ is onto} \\
\text{and for all } i \in I \quad \phi_A : \alpha(i) \rightarrow \alpha(\phi_I(i)) \text{ is a bijection}
\]


then $\phi_A^*(L) \subseteq L'_\phi$ where $L'_\phi \subseteq L'$. The projection morphism $\pi$ will not in general meet the conditions above since $\pi_A : A \to A_G$ is not a bijection. It is therefore not possible to use the results of this section in their current form to prove that $\pi_A^*(L) \subseteq L_G$ where $L_G$ is the vector language of the quotient system. In the next section we address this problem by splitting the projection morphism.

4.4 The Split Morphism

We can split the an LS homomorphism $\phi$ into $\eta$ and $\mu$ where $\mu \circ \eta = \phi$, as stated in the next definition.

**Definition 4.4.1** Let $\phi = (\phi_I, \phi_A)$ an LS homomorphism be such that $\phi_I : I \to I'$ and $\phi_A : A \to A'$ are both onto. Then we form a new system $LS = (I'', A'', \alpha'', L'')$ with

\begin{align*}
I'' &= I \\
A'' &= A' \\
\alpha''(i) &= \phi_A(\alpha(i)) \\
L''_i &= \phi_A(L_i)
\end{align*}

Then we have $\eta : (I, A) \to (I'', A'')$ such that

\begin{align*}
\eta_I(i) &= i \quad \text{for } i \in I \\
\eta_A(a) &= \phi_A(a) \quad \text{for } a \in A
\end{align*}

and $\mu : (I'', A'') \to (I', A')$ such that

\begin{align*}
\mu_I(i) &= \phi_I(i) \quad \text{for } i \in I'' \\
\mu_A(a'') &= a'' \quad \text{for } a'' \in A''
\end{align*}

In summary:

\begin{align*}
A &\xrightarrow{\eta_A} A'' \xrightarrow{\mu_A} A' \\
I &\xrightarrow{\eta_I} I'' \xrightarrow{\mu_I} I'
\end{align*}
Proposition 4.4.1 The functions \( \eta \) and \( \mu \) are LS homomorphisms, i.e. they satisfy (3.2.1) and (3.2.2).

Proof

For \( \eta : (I, A) \to (I'', A'') \) it must be shown that
\[
\eta_A(\alpha(i)) \subseteq \alpha''(\eta_I(i)) \quad \text{and} \quad \eta_A(L_i) \subseteq L''_{\eta_I(i)}
\]

Now
\[
\eta_A(\alpha(i)) = \phi_A(\alpha(i)) \quad \text{by definition of } \eta_A
\]
\[
= \alpha''(i) \quad \text{by 4.4.17}
\]
\[
= \alpha''(\eta_I(i))
\]
further
\[
\eta_A(L_i) = \phi_A(L_i) \quad \text{by definition of } \eta_A
\]
\[
= L''_i \quad \text{by 4.4.18}
\]
\[
= L''_{\eta_I(i)}
\]

For \( \mu : (I'', A'') \to (I', A') \) it must be shown that
\[
\mu_A(\alpha''(i)) \subseteq \alpha'(\mu_I(i)) \quad \text{and} \quad \mu_A(L''_i) \subseteq L'_{\mu_I(i)}
\]

Now
\[
\mu_A(\alpha''(i)) = \mu_A(\phi_A(\alpha(i))) \quad \text{by 4.4.17}
\]
\[
= \phi_A(\alpha(i)) \quad \text{as } \mu_A \text{ is the identity on } A''
\]
\[
\subseteq \alpha'(i') \quad \text{by LS homomorphism property (3.2.1)}
\]
\[
= \alpha'(\mu_I(i)) \quad \text{since } i' = \phi_I(i) = \mu_I(i)
\]
further
\[
\mu_A(L''_i) = \mu_A(\phi_A(L_i)) \quad \text{by 4.4.18}
\]
\[
= \phi_A(L_i) \quad \text{as } \mu_A \text{ is the identity on } A''
\]
\[
\subseteq L'_{i'} \quad \text{by LS homomorphism property (3.2.2)}
\]
\[
= L'_{\mu_I(i)} \quad \text{since } i' = \phi_I(i) = \mu_I(i)
\]
\[\square\]
The motivation for splitting the morphism was to make the results of the last section applicable to the projection morphism $\pi$. Definition 4.4.1 of a split LS homomorphism $\phi$ required $\phi_I$ and $\phi_A$ to be onto, which is satisfied by the projection morphism $\pi$.

The split projection morphism takes the form:

$$
\begin{align*}
\eta & : A \xrightarrow{\pi_A} A_G \\
\mu & : I \xrightarrow{\pi_I} I_G
\end{align*}
$$

From proposition 4.4.1 we have that $\eta \text{ and } \mu \text{ satisfy the properties of an LS homomorphism, hence they also satisfy the conditions of } \gamma \text{ the projection morphism as } \pi \text{ is in the category LS.}$

If we split $\pi \text{ into } \mu \text{ and } \eta \text{ into two morphisms in this way then it can be shown that one of the morphisms } (\mu) \text{ preserves behaviour; the second morphism is simpler than } \pi. \text{ So the question of whether } \pi \text{ preserves behaviour is reduced to the question of whether the more simple morphism } \eta \text{ preserves behaviour.}$

**Theorem 4.4.1** If $\phi$ is the projection morphism from a group action then $\mu$ satisfies the conditions of lemma 4.3.1.

**Proof**

If $\phi_1$ is the projection morphism then $\mu : (A'', I'') \to (A', I')$ comprises

$$
\begin{align*}
\mu & : A'' \to A' \text{ a bijection as } \mu(a'') = a' \forall a'' \in A'' \\
\mu & : I'' \to I' \text{ onto as } \mu(i) = \pi_I(i)
\end{align*}
$$

It remains to be proved that $\mu : \alpha(i) \to \alpha([i])$ is a bijection.

Since $\mu$ is the identity on $A''$ then $\mu : \alpha(i) \to \alpha([i])$ is one to one.

If $\pi_I$ is the projection morphism associated to the action of the permutation $\sigma = (\sigma_I, \sigma_A)$ then since $\sigma_A(\alpha(i)) = \alpha(\sigma_I(i))$, we have

$$
\alpha(i) \sim \alpha(\phi_I(i)) \quad (4.4.19)
$$
Let $\sigma_I(i) = i'$ and let $p \in \alpha([i])$, then there exists $a \in A$ such that $\sigma_A(a) = p$, since $\sigma_A$ is a bijection, hence $a \sim p$. By (4.4.19), we must have $i_1 \in I$ such that $a \in \alpha(i_1)$ and $\sigma_I(i_1) = i'$. Whence since $\sigma_I(i) = \sigma_I(i_1)$ we have $a \in \alpha(i)$ such that $\sigma_A(a) = p$.

\[ \square \]

It was stated in definition 4.3.11, that for a $LS$ homomorphism $\phi : LS \to LS'$, $L_\phi(i') \subseteq L'_\phi$. An analogous definition for a language system and its quotient will now be considered. For $\pi$ a projection morphism associated to the action of a group $G$ of $LS$ automorphism, we have

$$L_\pi([i]) = \bigcup \{ \pi_A(ab\ldots) | ab\ldots \in L_j \}$$

In the next proposition it is shown that $L_\pi([i]) = L_G([i])$, where $L_G([i])$ is the string language of the quotient language system.

**Proposition 4.4.2** For $\pi : (I, A) \to (I_G, A_G)$ a projection morphism associated to a group $G$ of $LS$ automorphisms $\sigma : (I, A) \to (I, A)$,

$$L_\pi([i]) = L_G([i])$$

**Proof**

$$L_\pi([i]) = \bigcup \{ \pi_A(x) | x \ldots \in L_j \}$$

$$= \bigcup_{j \sim i} \{ \pi_A(x) | x \in L_j \}$$

$$= \bigcup_{\sigma A \in G} \{ \pi_A(\sigma_A(x)) | x \in L_i \}$$

$$= \{ \pi_A(x) | x \in L_i \} \quad \text{since } \pi_A(\sigma_A(a)) = \pi_A(a)$$

$$= L_G([i]) \quad \text{by definition}$$

\[ \square \]
4.4. The Split Morphism

Two approaches have been taken to address the important question of behaviour preservation. Firstly, a category of vector languages $\mathbf{VL}$ with morphisms $\phi_A^*$ which preserve behaviour was constructed, and an investigation of which, if any, of the subcategories of $\mathbf{LS}$ would allow a functor $LV : \mathbf{LS} \to \mathbf{VL}$ to be defined followed. This investigation took the form of a series of examples of $\mathbf{LS}$ homomorphisms between language systems, with the functions that $\phi$ comprises being progressive more strictly defined. The conclusion was that even with $\phi_I$ and $\phi_A$ bijective functions, the $\mathbf{LS}$ homomorphism $\phi$ could not be extended to a function $\phi_A^*$ that preserved behaviour. However, it was proved that with the wide subcategory of $\mathbf{LS}$ in which the morphisms are $\mathbf{LS}$ isomorphism as the domain, it was possible to define the functor $LV$. The behavioural conclusion to be drawn from this is that isomorphic language systems have identical behaviour.
4.4. The Split Morphism

The second approach identified a condition, 4.3.10, that would ensure behaviour preservation:

\[ \phi_A^*(\bar{x})(t') = \phi_A(a_{1'}b')\phi_A(a_{2'}b') \ldots \phi_A(a_{k'b'}) \in L' \quad \forall t' \in I' \]

and then determined the properties of an LS homomorphism \( \phi \) needed to meet that condition. One of these properties was \( \phi_A \) a bijective function, which would not generally be satisfied by the quotient morphism which comprises \( \pi_I \) and \( \pi_A \) both onto functions. However, when \( \pi \) was split into two morphisms, \( \mu \) and \( \eta \), both of which are simpler than \( \pi \), it was proved that \( \mu \) had the properties required to preserve behaviour. Hence the problem has been reduced to considering if \( \eta \), which is simpler than \( \pi \), preserves behaviour. If it does, then the projection morphism (which is the composition of \( \eta \) and \( \mu \)) preserves behaviour.
Chapter 5

Properties of Language Systems

In the last chapter, we identified the conditions that a LS homomorphism $\phi : LS \to LS'$ must satisfy to ensure $\phi A^*(\mathcal{L}) \subseteq \mathcal{L}'$. Under these conditions, any behaviour of the language system $LS$ is contained in the image of $\phi A^*$. In this chapter we consider whether specific behaviours, or properties, of a language system are preserved under $\phi A^*$.

Given language systems $LS$ and $LS'$ and a morphism $\phi : LS \to LS'$, the questions we want to address are:

- Given $LS$ satisfies a property $P$, does $\phi$ guarantee that $LS'$ satisfies $P$?
- Given $LS'$ satisfies a property $P$, does $\phi$ guarantee that $LS$ satisfies $P$?

These questions will be considered for $\phi$ any LS homomorphism and for $\phi$ an LS homomorphism such that $\phi A^*(\mathcal{L}) \subseteq \mathcal{L}'$.

Reading $LS_G$ for $LS'$ the answers to these questions can provide insights into the behavioural conclusions that can be made about a language system from its quotient, and such observations will be made.

The properties that will be considered are absence of deadlock and extensibility.
5.1 Absence of Deadlock

A deadlock in a language system would be characterised by a vector that could not be extended. A behaviour vector \( \mathbf{x} \in \mathcal{L} \) cannot be extended if there is no \( a \in A_\alpha \) such that \( \mathbf{x} . a \in \mathcal{L} \). Therefore, a language system in which there are no deadlocks is one in which each of the behaviour vectors in \( \mathcal{L} \) can be extended. We say that such a language system is live.

**Definition 5.1.1** A language system is live if for all \( \mathbf{x} \in \mathcal{L} \), there exists \( a \in A_\alpha \) such that \( \mathbf{x} . a \in \mathcal{L} \).

We first consider the case when a language system \( LS \) is live and examine whether \( LS' \) is live given any \( LS \) homomorphism \( \phi : LS \rightarrow LS' \).

**Example 5.1.1** Consider language systems \( LS \) and \( LS' \) with

\[
egin{align*}
I &= \{i_1, i_2\} \quad I' = \{j_1, j_2\} \\
A &= \{a, b\} \quad A' = \{p, q\} \\
\alpha(i_1) &= \{a, b\} \quad \alpha'(j_1) = \{p, q\} \\
\alpha(i_2) &= \{a\} \quad \alpha'(j_2) = \{p, q\} \\
L_{i_1} &= \downarrow (ab)^* \quad L'_{j_1} = \downarrow (pq)^* \\
L_{i_2} &= a^* \quad L'_{j_2} = p^* \cup q \\
A_\alpha &= \left\{ \begin{pmatrix} a \\ b \\ \Lambda \end{pmatrix} \right\} \quad A'_{\alpha} = \left\{ \begin{pmatrix} p \\ q \end{pmatrix} \right\} \\
\mathcal{L} &= \downarrow (ab)^* \quad \mathcal{L}' = p \\
\mathcal{L}'_{\phi_\alpha}^* &\subseteq \mathcal{L}'.
\end{align*}
\]

Define \( \phi : LS \rightarrow LS' \) an \( LS \) homomorphism comprising a bijection \( \phi_I \) and a bijection \( \phi_A \) such that

\[
\phi_I(i_1) = j_1 \quad \phi_I(i_2) = j_2 \quad \phi_A(a) = p \quad \phi_A(b) = q
\]

The function \( \phi_{A}^* \) is well defined. The language system \( LS \) is live but the language system \( LS' \) is not.

This last example shows for \( \phi : LS \rightarrow LS' \) a \( LS \) homomorphism and \( LS \) live, \( LS' \) is not necessarily live. We now consider \( \phi \) an \( LS \) homomorphisms satisfying the conditions of proposition 4.3.1, so that \( \phi_{A}^*(\mathcal{L}) \subseteq \mathcal{L}' \).
Proposition 5.1.1 Let $\phi : LS \rightarrow LS'$ be an LS homomorphism satisfying the conditions of proposition 4.3.1 and further let $\phi_A^* : L \rightarrow L'$ be onto. If LS is live then LS' is live.

Proof

Let $x' \in L'$. Since $\phi_A^* : L \rightarrow L'$ is onto we have $x \in L$ such that $\phi_A^*(x) = x'$.

As LS is live we have $a \in A_a$ such that $x.a \in L$.

Now

$$\phi_A^*(x.a) = \phi_A^*(x)\phi_A(a) = x'\phi_A(a) \subseteq L'$$

by definition of $\phi$

Hence LS' is live.

□

The above proposition requires that $\phi_A^* : L \rightarrow L'$ is onto. The next examples shows that this condition can not in general be relaxed.

Example 5.1.2 Consider language systems LS and LS' with

$$I = \{i_1\} \quad I' = \{j_1\}$$

$$A = \{a, b\} \quad A' = \{p, q\}$$

$$\alpha(i_1) = \{a, b\} \quad \alpha'(j_1) = \{p, q\}$$

$$L_{i_1} = \downarrow (ab)^* \quad L'_{j_1} = \downarrow (pq)^* \cup q$$

$$L = \downarrow (ab)^* \quad L' = \downarrow (pq)^* \cup q$$

If $\phi : LS \rightarrow LS'$ comprises bijections $\phi_I$ and $\phi_A$ such that

$$\phi_I(i_1) = j_1 \quad \phi_A(a) = p \quad \phi_A(b) = q$$

then $\phi$ satisfies the conditions of proposition 4.3.1 and we have $\phi_A^*(L) = \downarrow (pq)^* \subseteq L'$ but there is no behaviour vector $x$ in $L$ such that $\phi_A^*(x) = q$, hence $\phi_A^*$ is not onto.
The language system $LS$ is live and the image of $L$ under $\phi_A^*$ is a behaviour vector that can be extended, but the language system $LS'$ is not live since there exists a behaviour in $L'$, namely $q$, which can not be extended.

We now consider whether the property that a language system is live transfers in the opposite direction, i.e. if the language system $LS'$ is live is $LS$ live given any LS homomorphism $\phi : LS \rightarrow LS'$? In the next example, $\phi$ satisfies the conditions of proposition 4.3.1 so $\phi_A^*(L) \subseteq L'$.

**Example 5.1.3** Consider language systems $LS$ and $LS'$ with

\[
\begin{align*}
I &= \{i_1\} & I' &= \{j_1\} \\
A &= \{a, b\} & A' &= \{p, q\} \\
\alpha(i_1) &= \{a, b\} & \alpha'(j_1) &= \{p, q\} \\
L_{i_1} &= \downarrow (ab) & L'_{j_1} &= \downarrow (pq)^* \\
L &= \downarrow (a \ b) & L' &= \downarrow (pq)^*
\end{align*}
\]

If $\phi : LS \rightarrow LS'$ comprises bijections $\phi_I$ and $\phi_A$ such that

\[
\phi_I(i_1) = j_1 \quad \phi_A(a) = p \quad \phi_A(b) = q
\]

then $\phi$ satisfies the conditions of proposition 4.3.1 and we have $\phi_A^*(L) = \downarrow (pq)$. The language system $LS'$ is live but the language system $LS$ is not live.

Note that in the last example $\phi_A^* : L \rightarrow L'$ is not onto. The next proposition proves that if it is onto and $LS'$ is live then $LS$ is live.

**Proposition 5.1.2** Let $\phi : LS \rightarrow LS'$ be an LS homomorphism satisfying the conditions of proposition 4.3.1 and further let $\phi_A^* : L \rightarrow L'$ be onto. If $LS'$ is live then $LS$ is live.

**Proof**

Under the conditions of proposition 4.3.1, $\phi_A^* : L \rightarrow L'$ is a one to one function.

Let $a \in L$. Then $\phi_A^*(a) = a' \in L'$.
5.1. Absence of Deadlock

As \( LS' \) is live we have \( a' \in A'_\gamma \) such that \( x'a' \in L' \). The function \( \phi_A : A \to A' \) is a bijection under the conditions of proposition 4.3.1 so we have \( a \in A \) such that \( \phi_A(a) = a' \). Since \( \phi_A^* : L \to L' \) is onto, there exists \( y \in L \) such that \( \phi_A^*(y) = x'a' = \phi_A^*(x) \phi_A(a) = \phi_A^*(\pi a) \).

Since \( \phi_A^* : L \to L' \) is a bijection we have \( y = x \cdot a \). Hence \( LS' \) is live.

This results for absence of deadlock will now be considered for a language system \( LS \), its quotient \( LS_Q \) and the projection morphism \( \pi : LS \to LS_Q \). If \( \pi \) satisfies the conditions of propositions 5.1.1 and 5.1.2 we can infer that a language system is live (has no deadlocks) if and only if its quotient is live (has no deadlocks).

In section 4.4 \( \pi \) was split such that \( \pi = \mu \circ \eta \) where \( \mu \) satisfies the conditions of proposition 4.3.1. Whether \( \pi \) satisfies conditions of 4.3.1 can then be determined by considering the simpler morphism \( \eta \).

Propositions 5.1.1 and 5.1.2 require that \( \pi_A^* : L \to L_G \) is an onto function which will not, in general, be satisfied by the projection morphism \( \pi \). In example 3.5.2 we have \( L = \downarrow (a \cdot c)^* \cup \downarrow (a \cdot c)^* \) and \( L_G = \downarrow ([a] [b] [a])^* \). For the vector \( [a] [b] [a] \in L_G \) there is no \( x \in L \) such that \( \pi_A^*(x) = [a] [b] [a] \).

The examples given in section 4.2 can be used to identify when \( \phi_A^* : L \to L' \) is not onto. In example 4.2.3, we have \( \phi_I : I \to I' \) and \( \phi_A : A \to A' \) both onto functions and \( \phi_A \) satisfies the commutivity condition, but we have \( pqp \in L' \) which is not in the image of \( \phi_A^*(L) \). When \( \phi_I : I \to I' \) is onto, \( \phi_A : A \to A' \) a bijection and \( \phi_A \) satisfies the commutivity condition (example 4.2.4), \( \phi_A^* : L \to L' \) is not, in general, onto. Similarly, when \( \phi_I : I \to I' \) is a bijection, \( \phi_A : A \to A' \) onto and \( \phi_A \) satisfies the commutivity condition (example 4.2.5).

In the next section we consider a property which is stronger than absence of deadlock, extensibility.
5.2 Extensibility

A live language system has the property that any of its behaviour vectors can be extended. Extensibility is the stronger property that a behaviour vector can be extended in such a way that each process, \( i \in I \) of the language system, makes progress.

**Definition 5.2.1** A language system \( LS \) is extensible if for all \( i \in I \) and \( \underline{x} \in \mathcal{L} \) there exists \( z \in A^*_\alpha \) and \( \alpha \in \alpha(i) \) such that \( \underline{x}.z.\alpha \in \mathcal{L} \).

We first consider the case when a language system \( LS \) is extensible and examine whether \( LS' \) is extensible given an LS homomorphism \( \phi : LS \to LS' \).

In example 5.1.1, \( LS \) is extensible but \( LS' \) is not extensible. (Note that a live language system is not always extensible). Hence, for \( \phi : LS \to LS' \) any LS homomorphism, if \( LS \) is extensible, \( LS' \) will not be extensible, in general. We now consider \( \phi \) satisfying the conditions of proposition 4.3.1.

**Proposition 5.2.1** Let \( \phi : LS \to LS' \) be an LS homomorphism satisfying the conditions of proposition 4.3.1 and further let \( \phi^*: \mathcal{L} \to \mathcal{L'} \) be onto. If \( LS \) is extensible then \( LS' \) is extensible.

**Proof**

Let \( \underline{x}' \in \mathcal{L}' \) and \( i' \in I' \). Since \( \phi^*: \mathcal{L} \to \mathcal{L}' \) is onto we have \( \underline{x} \in \mathcal{L} \) such that \( \phi^*(\underline{x}) = \underline{x}' \). Under the conditions of proposition 4.3.1, \( \phi_I : I \to I' \) is onto, so we have \( i \in I \) such that \( \phi_I(i) = i' \).

As \( LS \) is extensible we have, for all \( i \in I \), \( \underline{x} \in A^*_\alpha \) and \( \alpha \in \alpha(i) \) such that \( \underline{x}.z.\alpha \in \mathcal{L} \).

Now

\[
\phi^*_A(\underline{x}.z.\alpha) = \phi^*_A(\underline{x}) \cdot \phi^*_A(z) \cdot \phi_A(\alpha) = \underline{x}' \cdot \phi^*_A(z) \cdot \phi_A(\alpha) \subseteq \mathcal{L}' \quad \text{by definition of} \ \phi
\]

We have \( \phi^*_A(\underline{x}) \subseteq (A^*_{\alpha'})^* \) and \( \phi_A(\alpha) \in \alpha'(i') \), hence \( LS' \) is extensible. \( \square \)
We now examine whether the language system $LS$ is extensible given the language system $LS'$ is extensible and $\phi : LS \rightarrow LS'$ is an LS homomorphism. Example 5.1.3 in which $LS'$ is extensible but $LS$ is not shows that this is not generally the case. The LS homomorphism $\phi$ in example 5.1.3 satisfies the conditions of proposition 4.3.1 so $\phi_A^*(L) \subseteq L'$, but $\phi_A^*: (L) \rightarrow L'$ is not onto. When $\phi_A^*: (L) \rightarrow L'$ is onto, if $LS'$ is extensible then $LS$ is extensible, as the next proposition proves.

**Proposition 5.2.2** Let $\phi : LS \rightarrow LS'$ be an LS homomorphism satisfying the conditions of proposition 4.3.1 and further let $\phi_A^*: L \rightarrow L'$ be onto. If $LS'$ is extensible then $LS$ is extensible.

**Proof**

Under the conditions of proposition 4.3.1, $\phi_A^*: L \rightarrow L'$ is a one to one function.

Let $x \in L$ and $i \in I$. Then $\phi_A^*(x) = x' \in L'$ and $\phi_I(i) = i' \in I'$.

The language system $LS'$ is extensible, so there exists $z' \in (A_{i'}')^*$ and $a' \in a'(i')$ such that $x'z'a' \in L'$.

The function $\phi_A : A \rightarrow A'$ is a bijection so we have $a \in A$ such that $\phi_A(a) = a'$ and $z \in A_a^*$ such that $\phi_A^*(z) = z'$. As $\phi_A^*: L \rightarrow L'$ is onto we have $y \in L$ such that $\phi_A^*(y) = x'z'a' = \phi_A^*(x)\phi_A^*(z)\phi_A(a) = \phi_A^*(xza)$. As $\phi_A^*: L \rightarrow L'$ is a bijection we have $y = xza$.

Under the conditions of proposition 4.3.1, $\phi_A : \alpha(i) \rightarrow \alpha(\phi_I(i))$ is a bijection. We have $a' = \phi_A(a) \in \alpha'(i')$ hence $a \in \alpha(i)$. Hence $LS$ is extensible.

$\square$

Given a language system $LS$, its quotient $LS_G$ and the projection morphism $\pi : LS \rightarrow LS_G$, we can infer from propositions 5.2.1 and 5.2.2 that if $\pi$ satisfies the conditions of those propositions then $LS$ is extensible if and only if $LS_G$ is extensible.
Chapter 6

Categorical Relationship Between Nets and Language Systems

In this chapter we extend our reasoning about concurrent systems to another model of concurrency, namely elementary net systems. This is achieved by establishing the categorical relationship between elementary net systems and language systems. It starts by presenting a construction that describes the behaviour of an elementary net system in terms of a vector language. It is then shown that this construction extends to a functor from the category of elementary net systems, denoted $\mathbf{EN}$, to the category $\mathbf{LS}$. The category of elementary net systems used in this chapter is as presented by Nielsen and Sassone in [20].

A functor from a subcategory of $\mathbf{LS}$ to $\mathbf{EN}$ is defined, which together with the previously defined functor will comprise an adjunction.

6.1 Elementary Nets and the category $\mathbf{EN}$

Elementary net systems were introduced by Thiagarajan [31]. Some basic concepts concerning this fundamental class of nets will be given.

**Definition 6.1.1** An elementary net is a triple $(P,T,F)$ where $P$ and $T$ are disjoint...
sets of places and transitions, respectively, \( F \subseteq (P \times T) \cup (T \times P) \) is the flow relation. and \( P \cap T = \emptyset \) and \( P \cup T \neq \emptyset \), where \( \emptyset \) is the empty set.

The pre-elements and post-elements of \( x \in P \cup T \) are denoted:

\[
. x = \{y | (y, x) \in F\} \quad \text{the set of pre-elements of } x \\
. x^* = \{y | (x, y) \in F\} \quad \text{the set of post-elements of } x
\]

**Definition 6.1.2** An elementary net system is a quadruple \( EN = (P, T, F, M_{in}) \) where \( (P, T, F) \) is the underlying net and \( M_{in} \subseteq P \) is the initial marking of \( EN \).

A net may be represented graphical using circles as places, rectangles as transitions and directed arcs denoting the elements of the flow relation. A marked place contains a token, denoted \( \bullet \). Each place in an elementary net system may only hold one token. Below is an elementary net system with initial marking \( \{1, 2\} \).

![Figure 6.1: An elementary net](image)

The state of a net is the set of marked places holding concurrently, called the marking. A net can have an infinite state space and an example of such a net is that in figure 6.1. A transition \( t \) is enabled, or can fire, in a given marking \( \tilde{\rho} \subseteq P \) (denoted \( \tilde{\rho}[t] \)) if all of its pre-elements are marked and none of its post-elements are marked. The firing (or occurrence) rule of an elementary net states that the transition \( t \) can occur at the marking \( \tilde{\rho} \) and lead to the marking \( \tilde{\rho}' \), denoted \( \tilde{\rho}[t] \tilde{\rho}' \), if \( \tilde{\rho}[t] \) and \( \tilde{\rho}' = (\tilde{\rho} - . t) \cup . t \). The dynamic behaviour of an elementary net is determined by its firing rule.

When a transition occurs, each of its pre-elements ceases to be marked and all of its post-elements become marked. So in our example, figure 6.1, transition \( a \) is enabled in
the initial marking. After \( a \) occurs, the marking of the net is \( \{3, 4\} \) and both \( b \) and \( c \) are enabled.

The state space of a net, denoted \( P_N \), is the set of reachable markings, which may be defined as the least subset of \( P(P) \) containing \( M_{in} \) such that \( (\bar{p} \in P(P) \text{ and } \bar{p}[t] \bar{p}') \Rightarrow \bar{p}' \in P_N \).

To determine whether a transition can occur it is necessary to check both the pre-elements (to see that they are marked) and the post-elements (to see if they are not marked). An elementary net is said to be contact-free if, for all \( t \in T \) and for all \( \bar{p} \in P_N \), then \( \text{span}(\bar{p}) \subseteq \text{span}(\bar{p}') \Rightarrow \exists p \in \text{span}(\bar{p}) \) such that \( \bar{p} \cap \bar{p}' = \emptyset \). Hence in a contact-free net it is only necessary to check the pre-elements of a transition to determine whether it can fire. The example net, figure 6.1, is contact-free. An example of a net which is not contact-free is that following example 6.3.2, later in this chapter. Here we give an example of a simple elementary net system which is not contact free:

![Diagram](image)

**Figure 6.2: An elementary net which is not contact-free**

In the net in figure 6.2, transition \( a \) can not occur since its post-element is marked, illustrating contact. Transition \( a \) can only occur after transition \( b \) has occurred.

For any elementary net it is possible to construct another elementary net with the same firing rule but which is contact free. This construction is called the place complementation of a net. In the place complemented version of a net it is only necessary to consider the pre-elements of a transition to determine whether it can fire. The construction of a place complemented net will be illustrated by an example before the formal definition is given.

To construct the place complementation of an elementary net, for each \( p \in P \) add its place complementation, denoted \( \bar{p}^* \), such that

\[
\bar{p}^* = \text{span}(p) \quad \bar{p}^* = p^* \quad \text{and} \quad \bar{p} \in M_{in} \iff p \notin M_{in}
\]
The set of complementary places is denoted $\bar{P}$ and has the property that $P \cap \bar{P} = \emptyset$.

Figure 6.3 depicts the place complemented version of the net in figure 6.2. The original net has places \{1, 2\} in the initial marking and place 3 not in the initial marking. The place complementation of places 1 and 2 are places $\bar{1}$ and $\bar{2}$ respectively, both of which will not be in the initial marking. The place complementation of place 3 is place $\bar{3}$ will is in the initial marking. Hence the places in the place complemented net are \{1, 2, 3, $\bar{1}, \bar{2}, \bar{3}$\}. The flow relation of the complemented net comprises the flow relation of the original net together with additional elements determined by $\bar{p}^* = ^*p \quad ^*\bar{p} = p^*$. For example, in the original net place 2 has the pre-element $a$ so $a$ will be the post-element of 2. The post-element of place 2 in the original net is transition $b$ so $b$ will be the pre-element of 2.

![Figure 6.3: The place complementation of the net in figure 6.2](image)

Notice that the contact previously described for the net in figure 6.2 is no longer present in its place complemented form since $a$ is no longer enabled in the initial marking. Transition $b$ is enabled by the initial marking and the occurrence of $b$ results in marking in which $a$ is enabled, which is the firing sequence of the not contact-free net, figure 6.2.

A place complementation of an elementary net will now be defined.

**Definition 6.1.3** Given an elementary net $EN = (P_1, T_1, F_1, M_{in1})$, its place complementation is another elementary net $EN = (P_2, T_2, F_2, M_{in2})$ where

\[
\begin{align*}
P_2 &= P_1 \cup \bar{P}_1 \\
T_2 &= T_1
\end{align*}
\]
6.2. The Functor from EN to LS

\[ F_2 = F_1 \cup \bar{F}_1 \quad \text{where} \quad (\bar{p}, a) \in \bar{F}_1 \iff (a, p) \in F_1 \quad \text{and} \quad (a, \bar{p}) \in \bar{F}_1 \iff (p, a) \in F_1 \]

\[ M_{\text{in}_2} = M_{\text{in}_1} \cup \{ p \mid p \notin M_{\text{in}_1} \} \]

A further example of a place complemented net is now given.

![Diagram of the place complementation of the net in figure 6.1]

Figure 6.4: The place complementation of the net in figure 6.1

In [20], Nielsen and Sassone present a category EN of elementary net systems with their morphisms defined as follows:

**Definition 6.1.4** For two elementary net systems EN = (P, T, F, M_{\text{in}}) and EN' = (P', T', F', M'_{\text{in}}), a morphism \((\beta, \eta) : EN \to EN'\) consists of a relation \(\beta \subseteq P \times P'\) such that \(\beta^{-1}\) is a partial function \(P' \to P\), and a function \(\eta : T \to T'\) such that

\[ \forall (p, p') \in \beta \quad p \in M_{\text{in}} \iff p' \in M'_{\text{in}} \]  

(6.1.1)

\[ \forall t \in T \quad \beta(\ast t) = \ast \eta(t) \]  

(6.1.2)

\[ \beta(t^*) = \eta(t)^* \]  

(6.1.3)

In [20] a proof is given that elementary net morphisms defined in this way preserve behaviour.

6.2 The Functor from EN to LS

In this section, a functor between the categories of elementary nets and language systems will be constructed. As the first step, a vector language semantics for elementary
nets is defined.

**Definition 6.2.1** Given a net \( EN = (P,T,F,Min) \) define \( EL_{obj}(EN) = (I,A,\hat{\alpha},\hat{L}) \) where

\[
I = P \\
A = T \\
\hat{\alpha}(i) = \{ a \in A \mid (a,i) \in F \text{ or } (i,a) \in F \} = \text{ }^i \cup \text{ }^i \\
\hat{L}_i \subseteq \hat{\alpha}(i)^* 
\]

\( \hat{L}_i \) is defined according to whether the place is marked or not, i.e. if \( i \in Min \).

If \( i \in Min \),

\[
\hat{L}_i = \begin{cases} 
\downarrow \{ a \mid a \in i^* \} & \text{if } \text{ }^i = \emptyset \text{ (source)} \\
\Lambda & \text{if } \text{ }^i = \emptyset \text{ (sink)} \\
\downarrow \{ a.\bar{a} \mid \bar{a} \in i^* \text{ and } a \in i^* \}^* & \text{otherwise} 
\end{cases}
\]

If \( i \notin Min \),

\[
\hat{L}_i = \begin{cases} 
\downarrow \{ a \mid a \in i^* \} & \text{if } \text{ }^i = \emptyset \text{ (sink)} \\
\Lambda & \text{if } \text{ }^i = \emptyset \text{ (source)} \\
\downarrow \{ a.\bar{a} \mid \bar{a} \in i^* \text{ and } a \in i^* \}^* & \text{otherwise} 
\end{cases}
\]

**Example 6.2.1** To illustrate this construction, consider the elementary net in figure 6.1, with initial marking given by \( \bullet \).

Here \( I = \{1,2,3,4\} \), \( A = \{a,b,c\} \) with

alphanets \( \hat{\alpha}(1) = \{a,b\} \) \( \hat{\alpha}(2) = \{a,c\} \) \( \hat{\alpha}(3) = \{a,b\} \) \( \hat{\alpha}(4) = \{a,c\} \)

string languages \( \hat{L}_1 = \downarrow (ab)^* \) \( \hat{L}_2 = \downarrow (ac)^* \) \( \hat{L}_3 = \downarrow (ab)^* \) \( \hat{L}_4 = \downarrow (ac)^* \)

and event vectors \( a(i) = \begin{pmatrix} a \\ a \\ a \end{pmatrix} \) \( b(i) = \begin{pmatrix} b \\ \Lambda \\ b \end{pmatrix} \) \( c(i) = \begin{pmatrix} \Lambda \\ c \\ \Lambda \end{pmatrix} \)
6.2. The Functor from \( EN \) to \( LS \)

giving the vector language

\[
\downarrow (a \cdot b \cdot c)^* = \begin{pmatrix} ab \\ ac \\ ab \\ ac \end{pmatrix}
\]

Definition 6.2.1 establishes how our functor from \( EN \) to \( LS \) would map objects to objects. The next proposition proves that an elementary net morphism satisfies the requirements of a language system morphism, hence establishing how the functor maps morphisms to morphisms.

Proposition 6.2.1 Given an elementary net morphism \((\beta, \eta) : (P, T, F, M_{in}) \rightarrow (P', T', F', M'_{in})\)

then \((\beta, \eta) : (I, A, \hat{\alpha}, \hat{L}) \rightarrow (I', A', \hat{\alpha}', \hat{L}')\) is an LS homomorphism.

Proof

We show that an elementary net morphism satisfies the conditions (3.2.1) and (3.2.2) of an LS homomorphism.

Writing \( \beta \) and \( \eta \) in terms of the sets \( I \) and \( A \) gives a partial function \( \beta^{-1} : I' \rightarrow I \) and a function \( \eta : A \rightarrow A' \) such that

\[
\forall (i, i') \in \beta \quad i \in M_{in} \iff i' \in M'_{in} \quad (6.2.4)
\]

\[
\forall a \in A \quad \beta(a) = \eta(a) \quad (6.2.5)
\]

\[
\beta(a^*) = \eta(a)^* \quad (6.2.6)
\]

(6.2.5) may be written as

\[
\{ \beta(i) \mid (i, a) \in F \} = \{ i' \mid (i', \eta(a)) \in F' \} \quad \text{whenever} \quad (i, i') \in \beta \text{ or } i = \beta^{-1}(i')
\]

from which we have

\[
(\beta^{-1}(i'), a) \in F \iff (i', \eta(a)) \in F'
\]

similarly, (6.2.6) gives

\[
(a, \beta^{-1}(i')) \in F \iff (\eta(a), i') \in F'
\]
hence

\[(i', \eta(a)) \in F' \quad \text{or} \quad (\eta(a), i') \in F' \iff (\beta^{-1}(i'), a) \in F \quad \text{or} \quad (a, \beta^{-1}(i')) \in F\]

giving

\[\hat{\alpha}'(i') = \eta(\hat{\alpha}(\beta^{-1}(i'))) \quad \text{using (6.2.4)}\]

So for any \(i \in I\) whenever

\[i' \in \beta(i) \quad \text{(i.e.} \ i = \beta^{-1}(i')) \text{ then} \quad \hat{\alpha}'(i') = \eta(\hat{\alpha}(i))\]

so

\[\hat{\alpha}'(\beta(i)) \supseteq \eta(\hat{\alpha}(i))\]

which is (3.2.1) expressed in terms of \(\beta\) and \(\eta\), so an elementary net morphism has the first property of a LS homomorphism.

For property (3.2.2), only the case where \(i \in M_{in}\) will be considered (case for \(i \notin M_{in}\) can be proved similarly). Recall that for \(i \in M_{in}\):

\[\hat{L}_i = \begin{cases} \{a | (i, a) \in F\}^* & \text{if } *i = \emptyset \text{ (source)} \\ \Lambda & \text{if } *i = \emptyset \text{ (sink)} \\ \downarrow \{a.\bar{a} | (\bar{a}, i) \in F \text{ and } (i, a) \in F\}^* & \text{otherwise} \end{cases}\]

If \(i \in M_{in}\) then from (6.2.4) \(i' \in M_{in}\) whenever \((i, i') \in \beta\). So we have

\[\hat{L}'_i = \begin{cases} \{a' | (i', a') \in F'\}^* & \text{if } *i' = \emptyset \text{ (source)} \\ \Lambda & \text{if } *i' = \emptyset \text{ (sink)} \\ \downarrow \{a'.\bar{a}' | (\bar{a}', i') \in F' \text{ and } (i', a') \in F'\}^* & \text{otherwise} \end{cases}\]

It will be shown that for any \(i = \beta^{-1}(i')\), \(\eta\hat{L}_i \subseteq \hat{L}'_i\) in each of the three cases that define \(\hat{L}_i\). First note that (6.2.5) and (6.2.6) may be written as, respectively,

\[\beta^{-1}(j) \in \text{ } *a \quad \text{for each} \quad j \in \text{ } *\eta(a)\]  \hfill (6.2.7)

\[\beta^{-1}(j) \in \text{ } a^* \quad \text{for each} \quad j \in \eta(a)^*\]  \hfill (6.2.8)

Case 1: \(*i = \emptyset*

In this case we have, from the definition of \(*i\) that

\[\forall a \in A, \ i \notin a^* \implies \beta^{-1}(i') \notin a^* \quad \text{since} \quad i = \beta^{-1}(i')\]  \hfill (6.2.9)
so, by (6.2.7),

\[ i' \notin \eta(a)^* \quad \forall a \in A \quad (6.2.10) \]

Assume, for a contradiction, that \( *i' \neq \emptyset \) so \( \exists a' \in A' \) such that \( i' \in a'^* \). But \( a' = \eta(a) \), some \( a \in A \) contradicting (6.2.10), hence \( *i' = \emptyset \).

Since \( *i = \emptyset \) and \( *i' = \emptyset \), \( \hat{L}_i = \{ a \mid (i,a) \in F \}^* \) and \( \hat{L}_{i'} = \{ a' \mid (i',a') \in F' \}^* \) so it is sufficient to show that \( \eta\{ a \mid (i,a) \in F \} \subseteq \{ a' \mid (i',a') \in F' \} \).

\[
(i,a) \in F \implies i \in a^* \\
\implies \beta^{-1}(i') \in a^* \\
\implies i' \in \eta(a) \text{ by (6.2.7)} \\
\implies (i',\eta(a)) \in F'
\]

Hence \( \{ \eta(a) \mid (i,a) \in F \} = \eta\{ a \mid (i,a) \in F \} \subseteq \{ a' \mid (i',a') \in F' \} \)

Case 2: \( *i^* = \emptyset \)

In the case \( *i^* = \emptyset \), it can be proved using a similar argument to that in case 1 that \( i'^* = \emptyset \). Hence \( \hat{L}_i = \Lambda \) and \( \hat{L}_{i'} = \Lambda \) and clearly \( \eta\hat{L}_i \subseteq \hat{L}_{i'} \).

Case 3: \( *i \neq \emptyset \) and \( *i^* \neq \emptyset \).

For this case it must be shown that

\[
\eta\{ a.\bar{a} \mid (\bar{a},i) \in F \text{ and } (i,a) \in F \} \subseteq \{ \eta(a).\eta(\bar{a}) \mid (\bar{a},i) \in F \text{ and } (i,a) \in F \}
\]

Now

\[
(\bar{a},i) \in F \implies i \in \bar{a}^* \\
\implies \beta^{-1}(i') \in \bar{a}^* \\
\implies i' \in \eta(\bar{a})^* \text{ by (6.2.8)} \\
\implies (\eta(\bar{a}),i') \in F'
\]

Similarly,

\[
(i,a) \in F \implies (i',\eta(a)) \in F'
\]
Hence
\[
\{ \eta(a) \cdot \eta(\bar{a}) | (\bar{a}, i) \in F \text{ and } (i, a) \in F \} = \eta(\bar{a} \cdot a) \cdot (\bar{a}, i) \in F \text{ and } (i, a) \in F \}
\subseteq \{ \eta(a) \cdot \eta(\bar{a}) | (\bar{a}, i) \in F \text{ and } (i, a) \in F \}
\]
Together these three cases give the result that
\[
\text{for } (i, i') \in \beta \text{ i.e. } i = \beta^{-1}(i'), \quad \eta\hat{L}_i \subseteq \hat{L}_{i'}
\]
which is (3.2.2) expressed in terms of $\beta$ and $\eta$.

**Definition 6.2.2** The functor $EL : EN \to LS$ consists of two functions, $EL_{\text{obj}}$ (given in definition 6.2.1) which maps nets to vector languages and $EL_{\text{mor}}$ which maps net morphisms to language morphisms where

\[
EL_{\text{mor}} (\beta, \eta) = (\beta, \eta)
\]

From the definition of $EL$, it is clear that it satisfies the requirements of a functor (see definition 3.1.3).

### 6.3 The Functor from LS to EN

In this section a functor from language systems to elementary nets will constructed. First the way in which such a functor translates objects will be considered. Initially the obvious approach is used, which is to take any $LS = (I, A, L, \alpha)$ and create a net $EN = (P, T, F, M_{in})$, where $P = I$, $T = A$ and $F$ is determined by the alphabets and string languages of $LS$ using definition 6.2.1. The following example demonstrates this initial approach.

**Example 6.3.1** The following language system
\[
I = \{1, 2, 3, 4, 5\} \quad A = \{x, y, a, b, c\}
\]
\[
\alpha(1) = \{x, y\} \quad \alpha(2) = \{x, y, a\} \quad \alpha(3) = \{y, b\} \quad \alpha(4) = \{a, b, c\} \quad \alpha(5) = \{c\}
\]
\[
L_1 = \{\Lambda, x, y\} \quad L_2 = \downarrow \{xa, ya\} \quad L_3 = \downarrow \{yb\} \quad L_4 = \downarrow \{ac, bc\} \quad L_5 = \{\Lambda, c\}
\]
An elementary net for this language system (determined using definition 6.2.1) is:

![Diagram of an elementary net]

Notice that when \( y \) fires it enables \( a \) and \( b \). Assume that \( a \) then fires first, followed by \( c \). Transition \( b \) is still enabled and can fire. However the structure of the vector languages \( L \) of the language system do not allow actions \( a \) and \( b \) to occur in this way, since \( ab \notin L_4 \).

The last example shows that definition 6.2.1 can not in general be used to construct an elementary net from any language system. Hence an alternative approach to constructing objects is needed.

A state machine net is a net in which every transition has exactly one pre-place and exactly one post-place. The net in the example 6.3.1 is not a state machine net as transition \( y \) has two post-places. Recall that the firing of transition \( y \) introduced the concurrency that was not permitted by the vector language of the language system from which the net was derived. A language system which ensures that definition 6.2.1 will result in a state machine net is now presented.

**Definition 6.3.1** Let \( L \) be a string language of \( LS = (I, A, \alpha, L) \) and define \( \equiv \) by

\[
x \equiv y \iff \forall u \in A^* : (xu \in L \iff yu \in L)
\]

Note that \( \equiv \) is an equivalence relation.
This equivalence relation will now be used to define a reduced string language.

**Definition 6.3.2** The string language $L$ is reduced if

$$x.a, y.a \in L \implies x \equiv y \quad \text{for } a \in A \text{ and } x, y \in L$$

An example of a string language which is not reduced is $L = \{xa, ya, yb\}$. We see that $xa, ya \in L$ but $x \neq y$ because $yb \in L$ but $xb \notin L$. Hence $L$ is not reduced.

**Proposition 6.3.1** The string languages derived from state machine nets using definition 6.2.1 will be reduced.

**Proof**

Suppose $x.a$, $y.a$ are in the string language derived from a state machine net using definition 6.2.1, then $x$ and $y$ must both lead to a marking from which $a$ can fire. This must be a marking which marks only the single input place of $a$, $p$. So $M_0(x)p$, $M_0(y)p$ and

$$xu \in L \iff p(u) \iff yu \in L \quad x \equiv y$$

\[\square\]

**Corollary 6.3.1** The string languages derived from an elementary nets using definition 6.2.1 will be reduced.

**Proof** As for proposition 6.3.1 except that there may be more than one input place of $a$.

\[\square\]

As the first step in constructing the net from a language system we define a state machine net for each reduced string language $L_i$ for the language system.
Definition 6.3.3 For \( L_i \) reduced, define the tuple \( N_{L_i} = (P_{L_i}, T_{L_i}, F_{L_i}, M_{L_i}) \) as follows:

\[
\begin{align*}
P_{L_i} &= \{\{x\} | x \in L_i\} \quad \text{uses equivalence relation definition 6.3.1} \\
T_{L_i} &= \{a \in A | \exists x \in L_i : \#_a x > 0\} \quad \text{where} \#_a x \text{ is the number of } a \text{'s in } x \\
[x] F_{L_i} a &\iff xa \in L_i \\
a F_{L_i} [y] &\iff \exists x \in L_i : x.a \in [y] \\
M_{L_i} &= \{[\Lambda]\}
\end{align*}
\]

Proposition 6.3.2 \( N_{L_i} \) is a net, providing \( L_i \neq \emptyset, L_i \neq \{\Lambda\} \)

Proof Since \( \Lambda \in L_i \), \( P_{L_i} \cup T_{L_i} \neq \emptyset \). We make an assumption that no \( a \in A \) is a set, so \( P_{L_i} \cap T_{L_i} = \emptyset \).

If \( [x] \in P_{L_i} \), then if \( x \neq \Lambda \), \( x = y.a \) so \( a F_{L_i} [x] \). If \( x = \Lambda \) then as \( L_i \neq \{\Lambda\} \), \( a \in L_i \), some \( a \in A \) and \([\Lambda] F_{L_i} a \). Hence \( P_{L_i} \subseteq \text{domain}(F_{L_i}) \cup \text{range}(F_{L_i}) \).

If \( a \in T_{L_i} \), then \( x.a \in L_i \), some \( x \), so \( [x] F_{L_i} a \) and \( a \in \text{range}(F_{L_i}) \). It has been shown that \( P_{L_i} \cup T_{L_i} \subseteq \text{domain}(F_{L_i}) \cup \text{range}(F_{L_i}) \).

\[ \square \]

Definition 6.3.4 A language system \( LS = (I, A, \alpha, L) \) is reduced if each of the \( L_i \) is reduced.

An example of the nets representing each string language of a reduced language system will now be given.

Example 6.3.2 For the language \( L_1 = \{ (ab, ac)^* \} \), the equivalence classes are \([a]\), \([ab]\) = \([ac]\) and the state machine net, \( N_{L_1} \):
For the language $L_2 = \downarrow (bc)^*$, the equivalence classes are $[b]$ and $[bc]$ and the state machine net, $N_{L_2}$:

The nets for each reduced string language are composed to form the net for the language system, according to the following definition.

**Definition 6.3.5** For each $i \in I$ of a language system $LS$, let $N_{L_i} = (P_{L_i}, T_{L_i}, F_{L_i}, M_{L_i})$ be the state machine net of definition 6.3.3. Define the tuple $N_{LS} = (P_{LS}, T_{LS}, F_{LS}, M_{LS})$ as follows:

\[
\begin{align*}
P_{LS} &= \cup (P_{L_i} \times \{i\}) \\
T_{LS} &= \cup T_{L_i} \\
(p, i) F_{LS} t &\iff p F_{L_i} t \\
t F_{LS} (p, i) &\iff t F_{L_i} p \\
M_{LS} &= \{([\Lambda], i) | i \in I\}
\end{align*}
\]

**Proposition 6.3.3** $N_{LS}$ is a net, providing $L_i \neq \emptyset$, $L_i \neq \{\Lambda\}$, at least one $i$. 
6.3. The Functor from LS to EN

Proof

By proposition 6.3.2, \( P_L \cap T_L = \emptyset \), each \( i \in I \), so \( P_{LS} \cap T_{LS} = \emptyset \).

If \((p, i) \in P_{LS}\), then if \( p \neq \Lambda \) there exists \( t \in T_{LS} \) such that \( t F_{Li} p \), proposition 6.3.2, and \( t F_{Li} p \implies F_{LS} (p, i) \), definition 6.3.5. If \( p = \Lambda \) then as \( L_i \neq \{\Lambda\} \), some \( i \), there exists \( a \in L \) such that \([\Lambda] F_{Li} a \), proposition 6.3.2, and \([\Lambda] F_{Li} a \implies (p, i) F_{LS} t \), definition 6.3.5. Hence \( P_{LS} \subseteq \text{domain} (F_L) \cup \text{range} (F_L) \).

If \( a \in T_{LS} \), then \([x] F_L a \), proposition 6.3.2, and \([x] F_L a \implies (p, i) F_{LS} a \), definition 6.3.5, and \( a \in \text{range} (F_{LS}) \). It has been shown that \( P_{LS} \cup T_{LS} \subseteq \text{domain} (F_{LS}) \cup \text{range} (F_{LS}) \).

\[ \square \]

The composition of the state machine nets in example 6.3.2 gives \( N_{LS} \):

Notice that after the firing sequence \( aba \), places \([a]\) and \([b]\) are marked. Hence there is a contact at \( b \) and it cannot fire but \( c \) can fire, which is entirely consistent with the vector language of the system the net represents.

We now define how a functor from the category of reduced language systems, denoted \( \text{RLS} \), to the category of elementary nets maps objects to objects. The objects of \( \text{RLS} \)
are reduced language systems, definition 6.3.1, and morphisms are LS homomorphisms, definition 3.2.4. Hence $\text{RLS}$ is a full subcategory of $\text{LS}$ and as such there is an inclusion functor, denoted $i$, from $\text{RLS}$ to $\text{LS}$ such that for $\text{RLS}$ an object of $\text{RLS}$ and $\phi$ a morphism in $\text{RLS}$, $i(\text{RLS}) = \text{RLS}$ and $i(\phi) = \phi$.

**Definition 6.3.6** Given a reduced language system $\text{LS} = (I, A, \alpha, L)$ we define $\text{LE}_{\text{obj}}(\text{LS}) = (P_{\text{LS}}, T_{\text{LS}}, F_{\text{LS}}, M_{\text{LS}})$, to be the elementary net given in definition 6.3.5.

Turning to how the functor $\text{LE}$ translates morphisms, it is required that the morphisms in $\text{LS}$ satisfy the definition of morphism in $\text{EN}$, definition 6.1.4. The fact that LS homomorphisms ($\phi$) preserve structure whereas $\text{EN}$ morphisms preserve behaviour suggest that some restrictions on $\phi$ will be needed for this to be achieved. To test this, we return to the two language systems introduced in example 4.2.2. This example showed that it was possible to define a LS homomorphism between the language systems $\text{LS}$ and $\text{LS}'$, but it did not preserve behaviour. In the following example, the nets for the language systems given in example 4.2.2 are constructed using $\text{LE}_{\text{obj}}$, definition 6.3.6. We then attempt to define a net morphism between these two nets.

**Example 6.3.3** The string languages for $\text{LS}$ are $L_{i_1} = a^*$ and $L_{i_2} = b^*$. This is a reduced language system. The net $\text{LE}_{\text{obj}}(\text{LS}) = N_{\text{LS}}$ is:

\[
\begin{align*}
([A], i_1) & \quad \bullet \quad a \\
([A], i_2) & \quad \bullet \quad b
\end{align*}
\]

The string languages for $\text{LS}'$ are $L'_{j_1} = p^*$ and $L'_{j_2} = q^* \cup q^*p$. Again, this is a reduced language system. The net $\text{LE}_{\text{obj}}(\text{LS}') = N_{\text{LS}'}$ is:

\[
\begin{align*}
([A], j_1) & \quad \bullet \quad p \\
q & \quad \bullet \quad ([A], j_2) \\
([p], j_2) & \quad \bullet
\end{align*}
\]
We now attempt to define a net morphism \((\beta, \eta)\), definition 6.1.4, between \(N_{\text{LS}}\) and \(N_{\text{LS'}}\).

If \(\eta(a) = p\), then by condition 6.1.2 we require that \(\beta(a^*) = p^*\), which is satisfied if \(\text{([A],i_1) \beta ([A],j_1)}\) and \(\text{([A],i_1) \beta ([p],j_2)}\). However, \(\text{([A],i_1) \beta ([p],j_2)}\) violates condition 6.1.1 as place \(\text{([p],j_2)}\) is not in the initial marking. Similarly, condition 6.1.1 is also violated if \(\eta(b) = p\).

We therefore consider \(\eta(a) = \eta(b) = q\). The conditions \(\beta(a^*) = q^*\) and \(\beta(b^*) = q^*\) require, respectively, \(\text{([A],i_1) \beta ([A],j_2)}\) and \(\text{([A],i_2) \beta ([A],j_2)}\). These requirements are not met by a net morphism since \(\beta^{-1}\) is defined as a partial function \(\mathcal{P}_{\text{LS}} \rightarrow \mathcal{P}_{\text{LS'}}\).

The last example shows that given an LS homomorphism \(\phi\) from \(\text{LS}\) to \(\text{LS'}\) it is not always possible to define \((\beta, \eta)\) a net morphism from \(\mathcal{L}_{\text{obj}}(\text{LS})\) to \(\mathcal{L}_{\text{obj}}(\text{LS'})\). So in order to define \(\mathcal{L}_{\text{mor}}\), which maps morphisms in \(\text{RLS}\) to morphisms in \(\text{EN}\), we need some restrictions on LS homomorphisms \(\phi\). In order to identify these conditions the case of the state machine nets, \(\text{NL_i}\), definition 6.3.3, will be considered.

**Lemma 6.3.1** For \(\text{LS} = (I, A, L, \alpha)\) and \(\text{LS'} = (I', A', L', \alpha')\) reduced language systems let \(\phi : (I, A, L, \alpha) \rightarrow (I', A', L', \alpha')\) be a LS homomorphism, such that for \(x, y \in A^*\) and \(a, b \in A\):

1. \(\phi_A : A \rightarrow A'\) preserves the equivalence relation in definition 6.3.1, i.e. \(x = y \iff \phi_A(x) = \phi_A(y)\).

2. \(\phi_A : L \rightarrow L'\) onto and hence by (3.2.2), \(\phi_A(L_i) = \phi_A(L'_i)\) for all \(i \in I\).

3. For each \(a \in A\), \(a \in L_i\), some \(i\). Similarly for each \(a' \in A'\).

4. Whenever \(\phi_A(a) = \phi_A(b)\) and \(a'b \in L_i\) then \(xa \in L_i\), for all \(i \in I\).

For \(\text{NL_i} = (P_{L_i}, T_{L_i}, F_{L_i}, M_{L_i})\) and \(\text{NL'_i} = (P_{L'_i}, T_{L'_i}, F_{L'_i}, M_{L'_i})\) the state machine nets (definition 6.3.3) for \(L_i\) and \(L'_i\), where \(i \in I\), let \((\beta_{i'}, \phi_A) : (P_{L_i}, T_{L_i}, F_{L_i}, M_{L_i}) \rightarrow (P_{L'_i}, T_{L'_i}, F_{L'_i}, M_{L'_i})\) be defined by

\[
\phi_A : T_{L_i} \rightarrow T_{L'_i}
\]

\([x] \beta_{i'} [\phi_A(x)]\)
then \((\beta_{i^*}, \phi_A)\) satisfies definition 6.1.4 of an elementary net morphism.

Proof

By hypothesis 1, \(\beta : P_{L_i} \rightarrow P_{L_i}\) is a partial function.

It will now be shown that \((\beta_{i^*}, \phi_A)\) meets the conditions that an EN morphism must satisfy.

\[
[x] \in M_{L_i} \iff x \equiv \Lambda \\
\iff \phi_A(x) = \phi_A(\Lambda) = \Lambda \\
\iff [\phi_A(x)] \in M_{L_{i^*}}
\]

It has been shown that for \([x] \beta_{i^*}(\phi_A(x))\), \([x] \in M_{L_i}\) if and only if \([\phi_A(x)] \in M_{L_{i^*}}\), hence (6.1.1) is satisfied.

For any \(a \in A\), \([y] \in \beta_{i^*}(a^*)\) if and only if there exists \([x] \in a^*\) such that \([y] = [\phi_A(x)]\). Now

\[
[x] \in a^* \implies xa \in L_i \text{ and } [y] = [\phi_A(x)] \implies y \equiv \phi_A(x)
\]

from which it follows that \(\phi_A(x)\phi_A(a) \in L_{i^*}\), hence \(y \phi_A(a) \in L_{i^*}\) by definition 6.3.2.

So \([y] \in \phi_A(a)\).

Let \([y] \in \phi_A(a)\) then \(y \phi_A(a) \in L_{i^*}\). By hypothesis 3, we have \(ua \in L_i \implies \phi_A(u)\phi_A(a) \in L_{i^*}\), so \(\phi_A(u) \equiv y\), by definition 6.3.2 and \([\phi_A(u)] = [y]\). Hence there exists \(u \in a^*\) such that \([y] = [\phi_A(u)]\), so \([y] \in \beta_{i^*}(a^*)\).

It has been shown that \([y] \in \beta_{i^*}(a^*) \implies [y] \in \phi_A(a)\) and \([y] \in \phi_A(a) \implies [y] \in \beta_{i^*}(a^*)\), hence (6.1.2) is satisfied.

For any \(a \in A\), \([y] \in \beta_{i^*}(a^*)\) if and only if there exists \([x] \in a^*\) such that \([y] = [\phi_A(x)]\). Without loss of generality, we have \((x =)ua \in L_i\) such that \(y \equiv \phi_A(u)\phi_A(a) \in L_{i^*} \implies [y] \in \phi_A(a)^*\).

If \([y] \in \phi(a)^*\), then \(y \equiv w \phi(a) \in L_{i^*}\). We have \(vb \in L_i\) such that \(\phi(vb) = w \phi(a)\) by hypothesis 2. Hence \(y \equiv \phi(vb)\) with \(vb \in L_i\) and \(\phi(b) = \phi(a)\). But if \(vb \in L_i\) and \(\phi(b) = \phi(a)\), then \(va \in L_i\) by hypothesis 4. Hence \(y \equiv \phi(va)\) from which we have \([y] \in \beta_{i^*}(a^*)\).
It has been shown that \([y] \in \beta_{i'}(a^*) \implies [y] \in \phi_A(a)^* \) and \([y] \in \phi_A(a)^* \implies [y] \in \beta_{i'}(a^*)\), hence (6.1.3) is satisfied.

□

In the next proposition, it is proved that the elementary net morphism \((\beta_{i'}, \phi_A) : (P_{L_i}, T_{L_i}, F_{L_i}, M_{L_i}) \to (P_{L'_i}, T_{L'_i}, F_{L'_i}, M_{L'_i})\) may be extended to an elementary net morphism from \(LE_{obj}(LS) = N_{LS}\) to \(LE_{obj}(LS') = N_{LS'}\).

**Proposition 6.3.4** For \(LS = (I, A, L, \alpha)\) and \(LS' = (I', A', L', \alpha')\) reduced language systems let \(\phi : (I, A, L, \alpha) \to (I', A', L', \alpha')\) be a \(LS\) homomorphism, satisfying the conditions given in lemma 6.3.1, then for \(i \in \nu', (\beta, \phi_A) : (P_{L_i}, T_{L_i}, F_{L_i}, M_{L_i}) \to (P_{L'_i}, T_{L'_i}, F_{L'_i}, M_{L'_i})\) defined by

\[
\phi_A : T_{L_i} \to T_{L'_i}, \\
[x] \beta_{i'} [\phi_A(x)]
\]

is an elementary net morphism, lemma 6.3.1

For the elementary nets \(N_{LS} = (P_{LS}, T_{LS}, F_{LS}, M_{LS})\) and \(N_{LS'} = (P_{LS'}, T_{LS'}, F_{LS'}, M_{LS'})\) the morphism \((\beta, \phi_A) : (P_{LS}, T_{LS}, F_{LS}, M_{LS}) \to (P_{LS'}, T_{LS'}, F_{LS'}, M_{LS'})\) consisting of a relation \(\beta \subseteq P_{LS} \times P_{LS'}\) such that \(\beta : (UP_{i} \times \{i'\}) \leftarrow (UP_{i} \times \{i\})\) defined by

\[
(p, i) \beta (p', i') \iff i \phi_{i'} i' \text{ and } p \beta_{i'} p'
\]

and the function

\[
\phi_A : T_{LS} \to T_{LS'}
\]

is an elementary net morphism.

**Proof**

Follows from lemma 6.3.1 and the definition of \(N_{LS}\).
Definition 6.3.7 The functor $LE: RLS \rightarrow EN$ consists of two functions, $LE_{obj}$ (given in definition 6.3.6) which maps reduced language systems to elementary nets and $LE_{mor}$ which maps language morphisms to net morphisms where

$$LE_{mor}(\phi_I, \phi_A) = (\beta, \phi_A)$$

where $(\beta, \phi_A)$ are as defined in proposition 6.3.4.

The domain of this functor is the subcategory of $LS$ in which the morphisms are $LS$ homomorphisms satisfying the conditions 1, 2 and 4 of lemma 6.3.1 and the objects are reduced language systems satisfying condition 3 of lemma 6.3.1.

6.4 An Adjunction

We have proved the existence of functors in both directions between the categories which are the subject of this chapter. If it can be shown that these functors satisfy certain properties then we will have an adjunction between the two categories. An adjunction is an important categorical tool for comparing models of concurrency and we begin this section with its definition.

Definition 6.4.1 An adjunction between two categories $A$ and $B$ is the triple $(L, R, \kappa)$ where $L$ is a functor from $A$ to $B$, $R$ is a functor from $B$ to $A$ and for every object $a$ in $A$, there is a morphism $\kappa: a \rightarrow R \circ L(a)$ such that for each object $b$ of $B$ and each morphism $f: a \rightarrow R(b)$ there is a unique morphism $g: L(a) \rightarrow b$ such that the following diagram commutes:

\[
\begin{array}{ccc}
\text{a} & \xrightarrow{\kappa} & R \circ L(a) \\
\downarrow{f} & & \downarrow{R(g)} \\
R(b) & \xrightarrow{f} & b \\
\end{array}
\]

The functor $L$ is called the left adjoint and the functor $R$ the right adjoint. The morphisms $\kappa$ are the units of the adjunction.
If all units of an adjunction are isomorphisms then the adjunction is called a coreflection.

In an alternative definition, an adjunction is \((L, R, \gamma)\) where for every object \(b\) of \(B\), \(\gamma\) is a morphism from \(L \circ R(b)\) to \(b\) such that for each object \(a\) of \(A\) and each morphism \(g : L(a) \to b\) there is a unique morphism \(f : a \to R(b)\) such that the following diagram commutes:

\[
\begin{array}{ccc}
L(a) & \xrightarrow{f} & \gamma \xrightarrow{L} L \circ R(b) \\
\downarrow{g} & & \downarrow{L(\gamma)} \\
 b & \xleftarrow{\gamma} & L \circ R(b)
\end{array}
\]

The morphisms \(\gamma\) are the co-units of the adjunction. If all co-units of an adjunction are isomorphisms then the adjunction is called a reflection.

These definitions of adjunction are equivalent [2, proposition 9.2.2]. Here we shall use the definition of an adjunction which consists of the functors \(EL, LE\) and a co-unit \(\gamma\). The commutivity diagram for this definition of adjunction is repeated here, annotated for the categories \(LS\) (on the left side of the diagram) and \(EN\).

\[
\begin{array}{ccc}
is & \xrightarrow{\phi} & LE(is) \\
\downarrow{!} & & \downarrow{(\beta, \eta)} \\
EL(en) & \xleftarrow{\gamma} & LE \circ EL(en)
\end{array}
\]

In this diagram \(is\) is any object in the category \(LS\) and \(en\) is any object in the category \(EN\). To prove that \(LE\) and \(EL\) form an adjunction it must be proved the co-unit \(\gamma\) exists and is a net morphism. It must further be shown that for \((\beta, \eta)\) a net morphism from \(LE(is)\) to \(en\), there exists a unique language system morphism \(\phi\) from \(ls\) to \(EL(en)\) such that the above diagram commutes, i.e. \(\gamma \circ LE(\phi) = (\beta, \eta)\).
The first step in examining whether the functors described in this chapter form part of an adjunction is to consider $LE \circ EL(en_1)$ where $en_1$ is some elementary net. We take $en_1$ to be the net drawn in figure 6.5.

![Figure 6.5: The elementary net $en_1$](image)

Then $EL(en_1)$ is a reduced language system with string languages:

$L_1 = \{\Lambda, x, y\}$  
$L_2 = \downarrow \{xa, ya\}^*$  
$L_3 = \downarrow (yb)^*$  
$L_4 = \{\Lambda, a, b\}$

The equivalences classes of these string languages are:

$L_1$  
$[\Lambda] \text{ and } [x] = [y]$

$L_2$  
$[\Lambda] = [xa] = [ya] \text{ and } [x] = [y]$

$L_3$  
$[\Lambda] = [yb] \text{ and } [y]$

$L_4$  
$[\Lambda] \text{ and } [a] = [b]$

To construct the elementary net of this language system, i.e. $LE(EL(en_1))$, we first define a state machine net for each string language, which are as follows (the first net corresponds to $L_1$, the second to $L_2$, etc):

![State machine nets for string languages](image)
6.4. An Adjunction

The elementary net representing $LE \circ EL(en_1)$ is the composition of these state machine nets (definition 6.3.3):

![Diagrams](image)

The place complementation of the net in example 6.3.1 is given in figure 6.7.

If the net in figure 6.6 is compared to the net in figure 6.7 we see that they have the same initial marking and the same firing rule which indicates that they have the same behaviour. Furthermore, the structure of the two nets is identical. This example suggests that for any net $en$, $LE \circ EL(en)$ is its place complemented version.

**Proposition 6.4.1** The place complementation of any elementary net $EN$ has the same behaviour as and is structurally identical to the elementary net $LE \circ EL(EN)$.

**Proof** Given an elementary net $EN = (P, T, F, M_m)$, we define the language system $\hat{LS} = (I, A, \hat{\alpha}, \hat{L})$, where $\hat{LS} = EL_{obj}(EN)$, definition 6.2.1.
6.4. An Adjunction

Each \( \hat{L}_i \) will be reduced, Proposition 6.3.1, so \( \hat{L}S \) is a reduced language system.

We define an elementary net \( N_{L_S} = (P_{L_S}, T_{L_S} F_{L_S} M_{L_S}) \), where \( N_{L_S} = L E_{obj}(\hat{L}S) = L E_{obj} \circ E L_{obj}(EN) \), Definition 6.3.5. The net \( N_{L_S} \) is composed of state machine nets \( N_{L_i} \), Definition 6.3.3, one for each \( \hat{L}_i \) in the language system \( L_S \). The places \( P_i^s \) of \( N_{L_i} \) are defined as \( \{[x] \mid x \in \hat{L}_i\} \). Recall that \( \hat{L}_i \) is defined according to whether \( i \) is in the initial marking of \( EN \). We now define the state machine net \( N_{L_i} \) first for \( i \) in the initial marking and then for \( i \) not in the initial marking.

**Case 1:** \( i \in M_{in} \), where \( M_{in} \) is the initial marking of \( EN \).

If \( i^* = \emptyset \) then \( \hat{L}_i = \{A\} \) and hence \( N_{L_i} \) is not a net.

\[
\begin{align*}
P_{L_i} &= \begin{cases} 
\{[A], [a] \mid a \in i^*\} & \text{if } i^* = \emptyset \\
\{[a], [a \bar{a}] = [A] \mid \bar{a} \in i \text{ and } a \in i^*\} & \text{if } i^* \neq \emptyset \text{ and } i^* 
\end{cases} \\
T_{L_i} &= \{a, \bar{a} \mid a \in i^* \text{ or } a \in i^*\} \\
F_{L_i} &= [A] F_{L_i} a \cup a F_{L_i} [a] \cup [a] F_{L_i} \bar{a} \cup \bar{a} F_{L_i} [A] \\
M_{L_i} &= \{[A]\}
\end{align*}
\]

If we replace \([A]\) with \( i \) (to preserve the initial marking) and \([a]\) as the complement of \( i \) then the net \( N_{L_i} \) is the place complementation of \( i \).

**Case 2:** \( i \notin M_{in} \), where \( M_{in} \) is the initial marking of \( EN \).
If \( *i = \emptyset \) then \( \dot{i} = \{A\} \) and hence \( N_{L_i} \) is not a net.

\[
P_{L_i} = \begin{cases} 
\{[A], [a] \mid a \in \mathit{\dot{i}}\} & \text{if } \mathit{\dot{i}} = \emptyset \\
\{[a], [\bar{a}] = [A] \mid a \in \mathit{\dot{i}} \text{ and } \bar{a} \in \mathit{\dot{i}^*}\} & \text{if } \mathit{\dot{i}} \neq \emptyset \text{ and } \mathit{\dot{i}^*} \neq \emptyset
\end{cases}
\]

\[
T_{L_i} = \{a, \bar{a} \mid a \in \mathit{\dot{i}} \text{ or } \bar{a} \in \mathit{\dot{i}^*}\}
\]

\( F_{L_i} \) and \( M_{L_i} \) are as defined in case 1.

If we replace \([a] \) with \( i \) (to preserve the initial marking) and \([A] \) as the complement of \( i \) then the net \( N_{L_i} \) is the place complementation of \( i \).

The nets \( N_{L_i} \) are composed into the \( N_{L_S} \) in such a way that \( N_{L_S} = L\mathit{E}_{obj} \circ EL_{obj}(EN) \), with the replacements stated above, is the place complemented version of the net \( EN \).

\[\Box\]

We now define the co-unit \( \gamma \).

**Definition 6.4.2** For \( en \) an elementary net with places \( I \) and transitions \( T \) and \( L\mathit{E} \circ EL(en) \) an elementary net with places \( \cup(P_{L_i} \times \{i\}) \) and transitions \( T \), the co-unit \( \gamma : L\mathit{E} \circ EL(en) \to en \) is comprises as function \( \gamma_T : T \to T \) and a relation \( \gamma_I \) on \( \cup(P_{L_i} \times \{i\}) \times I \) such that

\[
\gamma_T(t) = t
\]

\[
([A], i) \gamma_I i \quad \text{for } i \text{ in the initial marking of } en
\]

\[
([a], i) \gamma_I i \quad \text{for } i \text{ not in the initial marking of } en \text{ and where } x \in \mathit{\dot{i}}
\]

If we consider \( \gamma \) mapping a place complemented net to the net from which it was derived, then to put it simply, \( \gamma \) ignores the places that have been added as a result of place complementation. Using the notation in the definition 6.1.3 of a place complemented net, \( \gamma_I \) is the identity function on the set of places \( P_1 \), the domain of \( \gamma_I \) being \( P_2 \) restricted to \( P_1 \).
Proposition 6.4.2 The co-unit $\gamma$ satisfies the definition of an elementary net morphism, definition 6.1.4.

Proof

We show that $\gamma$ satisfies the conditions of a net morphism, (6.1.1), (6.1.2) and (6.1.3).

By definition, $\gamma_t^{-1}$ is a partial function.

The initial marking of the net $LE \circ EL(en)$ is $\{([A], i) \mid i \in I\}$ hence $\gamma_t$ satisfies condition (6.1.1).

Let $i \in \ast \gamma_T(t)$. Then $\gamma_T(t) = t \in \ast$. If $i$ is in the initial marking of $en$, then $([A], i) \gamma_t i$, otherwise $([x], i) \gamma_t i$ where $x \in \ast i$. We show that $([A], i) \in \ast t$ and $([x], i) \in \ast t$.

Let $F$ be the flow relation of the net $LE \circ EL(en)$, derived from the flow relation given in the proof of proposition 6.4.1. For $i$ in the initial marking of $en$ we have $([A], i) F a$ for some $a \in \ast$. Hence $([A], i) F t$ since $t \in \ast$. For $i$ not in the initial marking of $en$ we have $([x], i) F a$ for some $a \in \ast$. Hence $([x], i) F t$ since $t \in \ast$. It has been shown that $([A], i) \in \ast t$ and $([x], i) \in \ast t$, hence $\gamma_t(\ast t) = \ast \gamma(t)$ and condition 6.1.2 is satisfied.

Let $i \in \gamma_T(t)^\ast$. Then $\gamma_T(t) = t \in \ast i$. For $i$ in the initial marking of $en$ we have $a F ([A], i)$ for some $a \in \ast i$. Hence $t F ([A], i)$ since $t \in \ast i$. For $i$ not in the initial marking of $en$ we have a $F ([x], i)$ for some $a \in \ast i$. Hence $t F ([x], i)$ since $t \in \ast i$. It has been shown that $([A], i) \in t^\ast$ and $([x], i) \in t^\ast$, hence $\gamma_t(t^\ast) = \gamma(t)^\ast$ and condition 6.1.3 is satisfied.

Having defined the co-unit $\gamma$ and proved that it is a net morphism, we proceed with the other requirements needed to prove the existence of an adjunction, which are: for $(\beta, \eta)$ a net morphism from $LE(ls)$ to $en$, there exists a unique language system morphism $\phi$ from $ls$ to $EL(en)$ such that $\gamma \circ LE(\phi) = (\beta, \eta)$.

These requirements will be illustrated using examples. Let the language system $ls$ have:

$I = \{1, 2, 3\}, \ A = \{a, b\}$ and $L_1 = \{A, a\} L_2 = \{A, a, b\} L_3 = \{A, b\}$
6.4. An Adjunction

Each of the string languages of \(ls\) is reduced. Then \(LE(ls)\) is given in figure 6.8.

We further take the following as the net \(en\).

\[ \begin{array}{c}
([\Lambda], 1) \bullet \\
\downarrow a \\
\hline ([\Lambda], 2) \bullet \\
\downarrow b \\
\hline ([\Lambda], 3) \bullet
\end{array} \rightarrow
\begin{array}{c}
\circ ([a], 1) \\
\circ ([a], 2) \\
\circ ([b], 3)
\end{array} \]

Figure 6.8: The net \(LE(ls)\)

We define a net morphism \((\beta, \eta)\) from \(LS(ls)\) to \(en\) as follows:

\[ \eta(a) = \eta(b) = c \quad \text{and} \quad ([\Lambda], 2) \beta I \quad ([a], 2) \beta II \]

The language system \(EL(en)\) has string languages \(L_I = \{\Lambda, c\}\) and \(L_{II} = \{\Lambda, c\}\).

We can define a LS homomorphism \(\phi : ls \rightarrow EL(en)\) as follows:

\[ \phi_A(a) = \phi_A(b) = c \quad \text{with} \quad \phi_I \text{ any relation on } \{1, 2, 3\} \times \{I, II\} \]

Note that \(\phi\) and \(ls\) satisfy the conditions on lemma 6.3.1.

There are many different possibilities for \(\phi\) but we require a unique \(\phi\) such that

\[ \gamma \circ LE(\phi) = (\beta, \eta) \quad (6.4.11) \]

We first consider equality for transitions and secondly for places.

The co-unit \(\gamma_T\) is the identity on transitions, so (6.4.11) requires \(\phi_A : \{a, b\} \rightarrow \{c\}\) to be the same function as \(\eta : \{a, b\} \rightarrow \{c\}\). This is the case in the above example.
Consider how each side of this equality (6.4.11) acts on the places of $LE(ls)$ (see figure 6.8). Recall that $([A], 2) \beta (I)$ and $([a], 2) \beta (II)$, no other places of $LE(ls)$ are in the relation $\beta$, where $\beta$ is the morphism from $LE(ls)$ to $en$. Now given the definition of $\gamma$, we require the $\beta$ component of the morphism $LE(\phi)$ to be defined as follows:

$$([A], 2) \beta ([A], I) \quad \text{and} \quad ([a], 2) \beta ([c], II) \quad (6.4.12)$$

The following table is given in summary.

<table>
<thead>
<tr>
<th>$LE(\phi)$</th>
<th>$\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$([A], 1)$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$([A], 2)$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$([A], 3)$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$([a], 1)$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$([a], 2)$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$([b], 3)$</td>
<td>$\rightarrow$</td>
</tr>
</tbody>
</table>

We now return to the definition of $\phi$. The $\beta$ component of $LE(\phi)$ is defined as

$$(p, i) \beta (p', i') \iff i \phi_I i' \text{ and } p \beta_{i'p'} p'$$

Hence the only LS homomorphism $\phi$ that results in (6.4.12) is one in which $2 \phi_I I$ and $2 \phi_I II$.

A possible generalisation of the conclusions from this example is that for $\phi : ls \rightarrow EL(en)$ to be unique it must be defined as follows:

$$\phi_A(a) = b \iff \eta(a) = b \quad \text{and} \quad i \phi_I i' \iff (p, i) \beta i'$$

The next theorem proves that if $\phi$ is defined in this way then $(LE, EL, \gamma)$ is an adjunction.

**Theorem 6.4.1** The functors $LE$ (definition 6.3.7) and $EL$ (definition 6.2.2), with the co-unit $\gamma$ (definition 6.4.2) are an adjunction.
Proof

Let $ls = (I, A, L, \alpha)$ be a reduced language system such that for each $a \in A$, $a \in L_i$, some $i$ (condition 3, lemma 6.3.1).

Then $LE(ls)$ is an elementary net with places $LE(I)$ and transitions $LE(A)$ where

$$LE(I) = \{([x], i) \mid x \in L_i\}$$

$$LE(A) = A$$

Let $en = (P, T, F, M_{in})$ be an elementary net. Further let $(\beta, \eta) : LE(ls) \rightarrow en$ be an elementary net morphism, i.e. a relation

$$\beta \subseteq LE(I) \times P$$

and a function

$$\eta : A \rightarrow T$$

satisfying conditions (6.1.1), (6.1.2) and (6.1.3).

By definition 6.2.1, $EL(en)$ is the reduced language system $(P, T, \hat{\alpha}, \hat{L})$. Then $LE \circ EL(en) = LE(P, T, \hat{\alpha}, \hat{L})$ is an elementary net with places $LE(P)$ and transitions $LE(T)$ where

$$LE(P) = \{([y], p) \mid y \in \hat{L}_p\} \text{ where } p \in P$$

$$LE(T) = \{t \in T \mid t \in \hat{L}_p\} = T$$

Let $\phi : ls \rightarrow EL(en)$ be any LS homomorphism, i.e. any relation

$$\phi_I : I \leftrightarrow P$$

and any function

$$\phi_A : A \rightarrow T$$

satisfying (3.2.1), (3.2.2) and the conditions of lemma 6.3.1.

Then $LE(\phi) : LE(ls) \rightarrow LE \circ EL(en)$ is a elementary net morphism consisting of a relation $\beta \subseteq LE(I) \times LE(P)$ and the function $\phi_A : A \rightarrow T$ such that

$$([x], i) \beta ([y], p) \iff i \phi_I p \text{ and } \phi_A(x) = y$$
For \((LE, EL, \gamma)\) to be an adjunction we require a unique \(\phi\) such that \(\gamma \circ LE(\phi) = (\beta, \eta)\). Since \(\gamma_T\) is the identity on \(T\), (6.4.16) must be the same function as \(\eta\) to satisfy \(\gamma \circ LE(\phi) = (\beta, \eta)\).

The co-unit \(\gamma_T \subseteq LE(P) \times P\) is defined as \([y], p, \gamma_T\) for any \([y], p \in \gamma_T\). Let \(([x], i) \beta p\), where \(\beta\) is the relation (6.4.13). Then \(\gamma \circ LE(\phi) = (\beta, \eta)\) is satisfied if \(i \phi_p\). Hence (6.4.15) must be defined as \(i \phi_p\) for any \([x], i \beta p\), some \(x \in L_i\), where \(\beta\) is the relation in (6.4.13).

It has been shown that (6.4.16) and (6.4.15) must be uniquely defined to satisfy \(\gamma \circ LE(\phi) = (\beta, \eta)\), hence \((LE, EL, \gamma)\) is an adjunction.

\[\square\]

The co-units \(\gamma\) of this adjunction are not isomorphisms as it is not always possible to define a net morphism \((\beta, \eta) : en \rightarrow LE \circ EL(en)\) where \(en\) is any net in \(EN\), as the following counterexample shows. Consider our example nets in figures 6.5 and 6.6. A net morphism from the net in figure 6.5 to the net in figure 6.6 must satisfy definition 6.1.4, in particular for every transition \(t\) we require \(\beta(\ast t) = \ast \eta(t)\). As the transitions of a net are invariant under \(LE \circ EL\) we take \(\eta\) to be the identity function. Hence

\[\ast \eta(a) = \{([x], 20), ([\Lambda], 4)\}\] \quad and \quad \ast a = 2

\[\ast \eta(b) = \{([y], 3), ([\Lambda], 4)\}\] \quad and \quad \ast b = 3

We require \(\beta(\ast a) = \ast \eta(a)\) which is satisfied if \(2\beta([x], 2) = 2\beta([\Lambda], 4)\). Considering transition \(b\) we require \(3\beta([y], 3) = 3\beta([\Lambda], 4)\). However, for \(\beta\) to be a net morphism we require \(\beta^{-1}\) to be a partial function which is contradicted by \(\beta^{-1}([\Lambda], 4) = 2\) and \(\beta^{-1}([\Lambda], 4) = 3\).

The co-units \(\gamma\) of this adjunction are not isomorphisms so the adjunction is not a reflection. The co-units are net morphisms from a place complemented net to the original net. However, it has been demonstrated that it is not in general possible to define a morphism from a net to the place complementation of that net due to the extra places that are introduced. In [21] a coreflection between asynchronous transition
systems and nets is presented. This coreflection is used to transfer a general concept of bisimulation to Petri nets. The unit and co-unit of the adjunction initially presented in [21] were not isomorphisms (due to the extra places introduced) and extra structure had to be added to the category of asynchronous transition systems in order to obtain the coreflection.

Adjunctions have been used to classify concurrency models, [24]. One of the parameters used is behaviour or system model. A language system is an example of the first category and an elementary net an example of the second, neither of these models is featured in [24].

An important result in category theory is that left adjoints preserve colimits and right adjoints preserve limits. Investigating the significance of this for the adjunction presented here is a possible avenue for future work, and is discussed in the next chapter.
Chapter 7

Conclusions and Future Work

This thesis presents a concurrency model called a language system and explores its relationships with its symmetry-reduced structure and elementary nets. Both are investigated in a categorical framework. Category theory has previously been used to determine relationships between different concurrency models [20] [24] but to the author's knowledge quotient structures have not previously been presented in the context of category theory. This is supported by the review of the literature on symmetries of concurrent systems which is contained in this thesis.

A new category of language systems LS has been presented. The morphisms that define this category are structure preserving. Permutations of language systems which define the notion of symmetry for the language system model were studied. A symmetry of a language system is determined by its string languages which differs from the state space symmetries discussed in chapter 2. A symmetry-reduction, or quotient structure, of a language system was then defined within a categorical framework. A proof that a language system and its quotient are both in the same category indicates that the quotient has comparable structure.

The use of symmetries to relieve the state explosion problem relies on the system and its quotient having similar behaviour. This important question of behaviour preservation was generalised to language system homomorphisms. A category of vector languages in which behaviour is preserved was defined. To address the question of behaviour
preservation we then attempted to define a functor from the category LS, or one of its wide subcategories, to the category of vector languages. Counter-examples of LS homomorphisms \( \phi \) between language systems, with the functions that \( \phi \) comprises being progressively more strictly defined, were presented which suggested that in general behaviour was not preserved. This is not unexpected since the language system morphisms were not defined to be behaviour preserving. The net morphisms considered later in this thesis were more strictly defined so as to ensure behaviour preservation. The automorphisms on Kripke structures used exploit symmetry in model checking to preserve the structure of the state space and hence preserve the behaviour of the underlying model.

The second approach to addressing the question of behaviour preservation identified a condition that would ensure behaviour preservation. The properties of an LS homomorphism \( \phi \) needed to meet that condition were given. To extend these results to the projection morphisms \( \pi \) which define the quotient system, a split morphism was constructed. When the projection morphism was split into two morphisms, \( \mu \) and \( \eta \), both of which are simpler than \( \pi \), it was proved that \( \mu \) had the properties required to preserve behaviour. Hence the problem has been reduced to considering if \( \eta \), which is simpler than \( \pi \), preserves behaviour. If it does, then the projection morphism (which is the composition of \( \eta \) and \( \mu \)) preserves behaviour.

Two specific behaviours, or properties, namely absence of deadlock and extensibility, were considered for the language system model. The conditions that \( \phi \) must satisfy to ensure that these properties were preserved have been identified and observations regarding the behavioural conclusions that can be made about a language system and its quotient were given. Model checking under symmetries requires that the group of automorphisms preserves the structure of the temporal logic formula that specifies the property of the system in addition to preserving the structure of the model. No such restriction on the group of symmetries was applied using the approach set out in this thesis.

The second main contribution of this thesis is to establish a categorical relationship between a category of elementary net systems \( \mathcal{EN} \) and the category of language systems.
defined in this thesis. The morphisms that define the category of elementary nets used preserve behaviour. A vector language semantics for elementary nets was given and this was then extended to a functor from \( \text{EN} \) to \( \text{LS} \). It is found that a functor may be defined from a subcategory of \( \text{LS} \) to \( \text{EN} \). The morphisms in this subcategory are \( \text{LS} \) homomorphisms with conditions added so that they satisfy the definition of morphisms in \( \text{EN} \). Given that morphisms in \( \text{EN} \) preserve behaviour it would be interesting to consider if the morphisms in this subcategory of \( \text{LS} \) preserve behaviour.

The existence of an adjunction comprising the functors between the categories \( \text{LS} \) and \( \text{EN} \) was proved. As the co-units of this adjunction are not isomorphisms the adjunction is not a reflection. It may be possible to add extra structure to the category of language systems to obtain a reflection. This kind of adjunction is useful because it implies that one category is embedded in (is more abstract) than the other.

Before giving ideas for future work, the main results of this thesis are given in summary:

- A new category of language systems and several wide subcategories
- A study of permutations on language systems and the definition of a language system symmetry
- Identification of the conditions under which a language system morphism preserves behaviour
- The application of behaviour preservation results to the quotient morphism using a split morphism
- A vector language semantics for elementary nets
- Definition of an adjunction between the categories of nets and language systems

Adjoints preserve limits and colimits. These preservation properties can be used to show how a semantics in one model translates to a semantics in the other. As mentioned in the introduction, an example of a limit is the categorical product between two objects in a category and this often corresponds to the parallel composition of the systems that the objects represent. Exploiting these preservation properties to reason about the semantics of the models is a potential area of future work.
This thesis has focused on using the group of symmetries to form the quotient structure of a system. This is just one property of a group of symmetries. A study of the structure of the group of symmetries is an example of another potential area of research interest. An investigation into the role of the isotropy subgroups and invariant subsets determined by the group should be included in a study of the structure of the group of symmetries.
Bibliography


