Categories of Asynchronous Systems

Marek A. Bednarczyk

Doctor of Philosophy
University of Sussex
October 1987

Rodzicom

Contents

Acknowledgements v

Declaration vi

Abstract vii

1 Introduction 1

1.1 Basic Ideas 1

1.2 Outline 4

2 Preliminaries 8

2.1 Conventions and Notation 8

2.1.1 Basic Mathematics 9

2.2 Basic Category Theory and Basic Categories 13

2.2.1 Basic Category Theory 13

2.2.2 Category of Small Sets: **Set** 19

2.2.3 Category of Pointed Sets: $\mathbf{Set}_\star$ 20

2.2.4 Category of Partial Functions: **Pfn** 21

2.2.5 Category of Finitary Multisets: $\mu - \mathbf{Set}$ 22

2.3 Special Categories 23

2.3.1 Category of Precategories 23

2.3.2 Categories of (Directed) Graphs with 'Partial' Morphisms: **pGrph** and **ppGrph** 27

I Generalised Transition Systems 35

3 Petri Nets, Petri Systems 36

3.1 Petri Nets: the Concrete Model of Asynchronous Computations 37

3.1.1 Transition Systems 37

3.1.2 Petri Nets — a Definition 39

3.1.3 A Category of Petri Nets 41

3.2 Petri Systems 43

3.3 A Negative Result 45

3.3.1 PNet is not (Co)reflective in PSys . . . 46
3.3.2 PSys is not (Co)reflective in PNet . . . 46

4 Asynchronous Systems 49
4.1 Category of Asynchronous Systems . . . 50
4.1.1 The Definition . . . 50
4.1.2 Relationship with Petri Systems . . . 52
4.1.3 Sequential Systems . . . 53
4.1.4 Labelled Asynchronous Systems . . . 54
4.2 Mazurkiewicz Traces — Computations of Asynchronous Systems . . . 55
4.2.1 Motivation . . . 55
4.2.2 Elementary Properties of Traces . . . 57
4.3 Representation Theorems . . . 63
4.3.1 Asynchronous Systems . . . 64
4.3.2 Forward Stable Asynchronous Systems . . . 67

5 Asynchronous Systems vs Other Models 70
5.1 Prime Event Structures . . . 71
5.1.1 Category of Prime Event Structures . . . 71
5.1.2 From Prime Event Structures to Trace Languages 73
5.1.3 ... and Back . . . 75
5.1.4 Coreflection . . . 79
5.2 General Picture . . . 81
5.2.1 Safe Petri Nets . . . 81
5.2.2 Fundamental Problem of Net Theory . . . 83
5.2.3 General Picture . . . 84

II Non-interleaving Semantics of CCS–like Languages 85

6 Recursive Languages: Their Syntax and Semantics 86
6.1 Syntax . . . 86
6.2 A Crash Course in Denotational Semantics . . . 90
6.2.1 Σ-algebras . . . 90
6.2.2 ω-complete Posets, ω-continuous Functions . . . 93
6.2.3 Σ-domains . . . 97
6.3 Categorical Interpretation of Recursive Languages . . . 100
6.3.1 ω-categories and ω-functors. . . . 100
6.3.2 Categorical Semantics of Recursive Languages. . . . 105

7 $\mathcal{L}$ — a Universal Language for Synchronisation 108
7.1 $\mathcal{L}$ – the Definition . . . 109
7.2 $\mathcal{L}$ – Axiomatic Operational Semantics . . . 110

7.3 *CCS* as a Sublanguage of $\mathcal{L}$ . . . 114
7.3.1 *CCS* – Syntax and Axiomatic Operational Semantics . . . 114
7.3.2 *CCS* – a Sublanguage of $\mathcal{L}$ . . . 116
7.3.3 *CCS* versus *CCS* . . . 121
7.3.4 *CCS* without τ . . . 124
7.4 Modelling in $\mathcal{L}$ Operators from Other Languages . . . 126
7.4.1 *SCCS* – a Synchronous Communication Principle . . . 126
7.4.2 *TCSP* . . . 127
7.4.3 Synchronisation Algebras . . . 129
7.4.4 Loose Ends . . . 131

8 $\mathcal{L}$ — a Categorical Semantics **134**
8.1 An Appropriate Category: $\mathcal{A}_M$ . . . 135
8.1.1 Category of Labelled Skeletal Systems: **LSS** . . . 135
8.1.2 Properties of **LSS** . . . 141
8.1.3 c^*-generated lss-s. . . . 150
8.1.4 Labelled Skeletal Systems over a Fixed Alphabet . . . 154
8.1.5 $\mathcal{A}_M$ . . . 160
8.2 Interpretation of $\mathcal{L}$ in $\mathcal{A}_M$ . . . 161
8.2.1 Interpretation of Ω . . . 161
8.2.2 Interpretation of $\mu \cdot _$ . . . 161
8.2.3 Interpretation of $_ \setminus L$ . . . 163
8.2.4 Interpretation of $_ [R]$ . . . 164
8.2.5 Interpretation of $_ \# _$ . . . 165
8.2.6 Interpretation of $_ \& _$ . . . 165
8.2.7 Interpretation of $\mathcal{L}$ . . . 166

9 From Operational to Denotational Semantics **168**
9.1 *CCS* – the Problem . . . 170
9.2 $\mathcal{L}$ – the Construction . . . 174
9.2.1 The Independency . . . 174
9.2.2 The Sameness . . . 176
9.2.3 $\mathcal{S}$ – a LSS for $\mathcal{L}$ from Syntax and Operational Semantics . . . 181
9.2.4 Counter-example Revisited . . . 182

10 Equivalence **183**
10.1 $\mathcal{L}$ without Recursion . . . 184
10.1.1 Preliminaries . . . 184
10.1.2 Equivalence . . . 191
10.2 $\mathcal{L}$ with Guarded Recursion . . . 192
10.2.1 The Sublanguage . . . 192
10.2.2 Preliminaries . . . 194

10.2.3 Syntactic Unfolding 196
10.2.4 Equivalence 200
10.3 The Problem of Full Abstraction 209

11 Conclusions **213**

Bibliography **216**

Acknowledgements

It is a pleasure to announce many stimulating discussions with my supervisor, Matthew Hennessy. Numerous conversations with Ilaria Castellani, Rocco De Nicola, Eugenio Moggi, Mogens Nielsen, Gordon Plotkin, Edmund Robinson, Mike Shields, Colin Stirling, Allen Stoughton and Glynn Winskel were also helpful.

I owe much to Glynn Winskel, whose ideas and approach guided my own work.

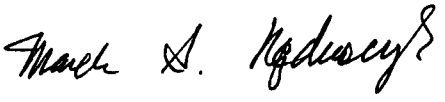
Andy Pitts and Gavin Wraith were always ready and willing to discuss and explain even the most elementary categorical problems. Andy deserves special thanks for reading parts of the manuscript, and helping to improve its presentation.

Most of all I would like to thank Magda and Szymek for being with me all the time — no matter how far away.

My studies in Great Britain were made possible by an ORS Award studentship. Moreover, I was financially supported, first by a University of Edinburgh Studentship, and subsequently by a University of Sussex Studentship. University of Gdańsk granted leave from the didactic duties for the period of my studies.

Declaration

This thesis was composed by myself, and the work reported is my own. It has not, whether in this or a different form, been submitted to this or any other University for a degree.



Abstract

University of Sussex

Marek Antoni Bednarczyk — Doctor of Philosophy

Categories of Asynchronous Systems

Summary

Asynchronous systems — a new non-interleaving model for concurrency is introduced. The study presented goes in two directions, reflected in the partition of the material into two Parts.

In Part I, asynchronous systems are shown to underly a variety of non-interleaving models for concurrency. In particular, they subsume Mazurkiewicz's *trace languages*, and *prime event structures*. The latter result is enough to make a connection with other models, thanks to the work of Winskel in the area. Asynchronous systems arise naturally as an abstraction from the concrete details of presentation of *safe Petri nets*. The existence of an inverse construction, i.e., the problem of *realisation* of an asynchronous system by means of a safe Petri net, is posed as the major open problem remaining to obtain a fairly unified theory.

Part II concentrates on the application of asynchronous systems to the study of the semantics of languages for communication. $\mathcal{L}$ — a 'universal' language, is introduced, and given an axiomatic operational semantics à la Plotkin. It is shown that a variety of languages and synchronisation primitives, can be interpreted as derived sublanguages, and operators of $\mathcal{L}$, resp., with their operational meaning also derived from that of $\mathcal{L}$. The language is given a (categorical) denotational semantics in, essentially, a subcategory of *labelled* asynchronous systems. Most of the denotational interpretation presented, is naturally determined by the categorical structure of the domain category. The two semantic descriptions are compared, and an equivalence result is obtained for the sublanguage of $\mathcal{L}$ with *guarded recursion*.

Chapter 1

Introduction

1.1 Basic Ideas

The advent of formalisms to describe *concurrent systems* had two, but interrelated sources.

The first was practical, and of synthetic nature. As the new technology begun to produce in abundance cheap processors, so grew the temptation to utilise them to accomplish complex tasks quicker. The hope was that one can divide more complex tasks into smaller chunks which could be executed simultaneously by separate processors. Therefore, a methodology was needed that would lead to the development of tools for reasoning about behaviour of complex systems in terms of the behaviours of their subsystem.

The second motivation was philosophical, and analytical in nature. There was a need to develop a general theory that would provide models for concurrency, and thus help recognise the problems and specific phenomena. Ultimately, it should provide a sound theoretical basis for the synthetic methodologies mentioned above.

To answer the practical problems a variety of solutions was proposed. At first, in the late sixties/early seventies, they amounted to syntactic extensions of then existing imperative languages. Normally, they involved introduction of an operator for putting several small programs, called *processes*, to work concurrently together, and adding on some *synchronisation primitives* to facilitate exchange of data between them. The early synchronisation primitives included Dijkstra's *semaphores*, Hoare's *monitors*, *critical sections* devised by Brinch-Hansen, and many others. The common feature of those early proposals was inherited from the programming languages they were meant to extend. Namely, the synchronisation was accomplished via *shared memory* as a communication medium.

Towards the end of last decade a more elegant solution begun to emerge. The idea was to separate the medium of storage from the medium of communication. Two early formalisms based on this principle include Milner's CCS, [Milner 75] and [Milner 80], and Hoare's CSP, [Hoare 78] and [Hoare 85]. They were the first attempts to answer the practical need by creating an entirely new formalism, dedicated to the problem. The first of them does away even with substitution — the main feature of imperative programming. Central to both is the idea that processes communicate along *channels* in a *hand-shake* fashion.

Many of today's programming languages are direct descendants of those two prototypes.

CCS, or *Calculus of Communicating Systems* in full, is not just another programming language, but a complete methodology for constructing and reasoning about complex *communicating systems*. It comes in the form of a calculus that enables one to prove systems *behaviourally equivalent*.

The *Calculus* is based on an extensionally defined notion of equivalence, so called *bisimulation equivalence*, [Park 81], [Milner 83]. One of the basic principles of the definition of bisimulation: *simultaneity is not observable*, results in an identification being made between a system in which two actions can occur concurrently, and its *sequentialisation* in which the actions can happen in any order, but not concurrently.

The other major behavioral theory based on *testing equivalences* also identifies the two systems, see [De Nicola&Hennessy 84]. It is built around the idea that two systems should be identified whenever no *experiment* can differentiate between them. An experiment consists of *coupling* or *synchronising* a system with another one, called a *test*. Then the compound systems is *started*. Tests are capable of reporting a *success* during their *evolution*. Hence, one obtains three equivalences corresponding to whether or not a system *may* and/or *must* pass each test. It is the property of the original testing scenario that a test cannot distinguish between a system, and its sequentialisation.

In both cases concurrency is effectively reduced to non-determinism. This type of semantics is referred to as *interleaving*.

In 1962 Petri introduced his *nets* as a first model in which concurrency of actions can be expressed naturally, [Petri 62]. Petri nets are very concrete model for concurrency — one in which the concurrency between actions is a property derivable from the structure of the net.

One of the properties of nets is that whenever two action can happen concurrently, they also can happen in either order. Therefore the interleaving principle is sound if nets are considered a basic model.

Those models which do contain information about concurrency are usually called *non-interleaving*. They are more natural when it comes to discuss such phenomena like *(partial) deadlock, starvation, finite delay* property, etc. Since some of those properties refer explicitly to the information about concurrency, it is highly unlikely that they can be successfully addressed in an interliving model.

Admittedly, there are only a few examples of properties of practical interest that are lost when choosing interleaving semantics. How many, if there are at all, depends on what one considers to be of practical interest. Anyhow, it seems that the choice of an interleaving semantics is satisfactory for most purposes. It makes the resulting methodology simpler since the concurrent composition operator is no longer primitive. Also, the two major extensional behavioural theories mentioned above result in interleaving semantics.

On the other hand the non-interleaving models are closer to operational intuitions. They provide ideas about *how* a system really works, rather than *what* it looks like. Thus, they should be considered a pattern against which all the methodological theories should be

judged.

Here, we are interested in further development of the theory of non-interleaving models for concurrency. In the light of what was said above it seems the right place to start one's studies into the area of concurrency. Continuation of the programme, and an application of asynchronous systems in the study of behavioural theories is the subject of author's current research.

Petri nets are considered the fundamental, but, we argue, too general and concrete model for concurrency. Here, more abstract *asynchronous systems* are put forward. Asynchronous systems are a simple generalisation of Keller's *transition systems*. They are also an abstraction from *safe* nets — a more regularly behaving category of Petri nets.

We show that the asynchronous systems underly many models known from the literature. Mazurkiewicz's *traces* are natural candidates to take as finite computations of asynchronous systems, and an appropriate category of trace languages is shown to be a *coreflective* subcategory of the category of asynchronous systems.

Trace languages, in turn, are shown to contain *prime event structures* as a coreflective subcategory. Thus, a link is established between asynchronous systems and other models, due to Winskel's work on this subject, see [Winskel 84a] and [Winskel 87].

As an important open problem the question remains whether one can prove (a subcategory of) asynchronous systems themselves being a coreflective subcategory of safe nets. In more elementary terms what we face is a problem of *realisation* of asynchronous systems by means of safe nets. If the former are to be thought of as abstract behaviours of the latter then the problem is if given an abstract description of a concurrent system, i.e., an asynchronous system, one can construct a safe net that exhibits it. In the perspective of other results such a construction would amount to a solution of the problem of building a safe net that accepts given trace language, realises a prime event structure, and so on.

So far the only result to date in this direction was given by Winskel who showed that this kind of construction can be accomplished for prime event structures, and *occurrence nets* a subcategory of safe nets, see e.g., Winskel's article in [APN'86/II]. Thus, one would expect the hypothetical solution to subsume Winskel's construction.

Another subject consists of an application of a *labelled* version of (a category equivalent to) the category of asynchronous systems to the study of the semantics of *CCS*-like languages. Asynchronous systems are *intensional* models. From Petri nets they inherit the property that all the actions have predetermined effect. To model non-deterministic behaviours one needs to accept a two level system of presentation of actions. At the basic, intensional level the actions behave deterministically. The nondeterminism in the extensional behaviour is achieved by introducing the concept of the *labelling* — a function that prescribes observable attributes to each action. A little novelty here comes in a form of an idea to use partial functions as labellings of actions. Intuitively, the labelling being undefined on an action means that the action is not extensionally observable.

We study $\mathcal{L}$ — a universal language for synchronisation. $\mathcal{L}$ is given an axiomatic opera-

tional semantics à la Plotkin. We call the language *universal* for it is possible to consider a variety of synchronisation primitives from languages like CCS and CSP, as derived operators of $\mathcal{L}$ with *derived* operational semantics. In fact, any synchronisation operator definable by means of *synchronisation algebras*, [Winskel 84b], is definable in $\mathcal{L}$ in this strong sense.

$\mathcal{L}$ is also given a (categorical) denotational semantics. It seems justified to use here categorical methods since category theory is used first to study asynchronous systems themselves. Even more so since the operators of $\mathcal{L}$ are interpreted with functors that arise naturally from categorical properties of labelled asynchronous systems. Categorical importance of the *sum* operator in CCS, and various synchronisation operators was first realised by Winskel, [Winskel 84]. It is shown that by working with labelled structures one can understand categorically also renaming, and restriction operators.

Finally, it is shown that the operational, and denotational semantics are equivalent. This is deemed as the main result of the work.

Category Theory is used throughout the thesis as an underlying mathematical theory and philosophy. Admittedly, our use of this powerful tool is somewhat elementary, if not naive. Nevertheless, it was the author's frequent experience that categorical approach gives the right hints on what one should look for. It enables one to look more abstractly at the phenomena, and better understand their relative importance. During the work on concurrency this attitude helped find some order in the place of a chaos.

1.2 Outline

The thesis is partitioned into a preliminary Chapter 2, Part I and Part II which constitute the core material, Chapter 11 that presents conclusions, and ends with Bibliography which contains only the basic references.

Chapter 2 contains notational conventions, and a brief revision of elementary mathematics and category theory. Next, we consider basic categorical constructions in such well-known categories as: **Set** — the category of small sets with total functions, **Pfn** — the category of small sets and partial functions, etc. The Chapter ends a study of two special categories: **preCat** — the category of *precategories*, and **pGrph** — the category of (directed) graphs with 'partial' morphisms. They are proved equivalent, and the primary reason to have the first of them around is to established an important categorical property of the second. Namely, it is shown that finite limits commute with filtered colimits in **preCat**. In terms of denotational semantics considered later in Chapter 8 it will guarantee continuity of one of the functors.

In Part I asynchronous systems, i.e., the main underlying model for concurrency, are introduced and compared with its rivals. This Part is divided into three Chapters.

The first among them, Chapter 3, starts from general Petri nets considered as the fundamental model for concurrency. They are introduced as a generalisation of the notion of a transition system, [Keller 76]. On the other hand Petri nets are presentations of certain

algebras in the category of μ-sets. Following Winskel, [Winskel 84], we take presentations of homomorphisms between such algebras as morphisms of Petri nets.

Next, *Petri systems* are introduced as a way to abstract away from concrete elements of the description of Petri nets. We consider the problem of a uniform realisation of Petri systems by means of Petri nets, and show it is insoluble. This result is taken as an argument against the excessive generality of both models.

Asynchronous systems are defined in Chapter 4 as an attempt to limit this generality. They were conceived independently by M. Shields and the author. They are obtained by adding an information about concurrency, called here *independency relation*, to a (deterministic) transition systems. Asynchronous systems form a category, denoted **AS**. It is compared with the category of Petri systems, and some of subcategories of **AS** are introduced. These include in particular a (full) subcategory with *forward stable asynchronous systems*, and a (full) subcategory with *sequential systems*. The latter coincides with Keller's transition systems.

Mazurkiewicz's traces find their natural place as *finite computations* of asynchronous systems, see [Mazurkiewicz 77]. A category **TLan** of trace languages is defined. The procedure of taking a set of all finite computations of an asynchronous system is shown functorial and, moreover, a coreflector. This coreflection cuts down to a coreflection between a reflective subcategory of **TLan** and forward stable asynchronous systems.

A link between asynchronous systems and other models, beside Petri systems and trace languages, is established in Chapter 5. To this end it is enough to relate asynchronous systems with event structures. The connections of the latter with other models for concurrency, and also with domain theory, is well established due to Winskel's work, e.g., see [Winskel 84a], and also his article in [APN'86/II].

We show that prime event structures form a coreflective subcategory of **TLan**. Again, this coreflection cuts down to a coreflection between already mention reflective subcategory of **TLan**, and what is called here *ideal* prime event structures. In fact, these were the first event structures considered, see [Nielsen et al 81].

Forward stable asynchronous systems arise naturally as an abstraction from the concrete details of *safe* Petri nets. As in Chapter 3 the problem of a uniform realisation of a forward stable asynchronous system as a safe net is posed. In author's opinion it remains the most important open problem in the are unification of the theory of models for concurrency.

The Chapter ends with a diagram illustrating the links established.

Part II has five Chapters. In it labelled asynchronous systems, or more exactly a category equivalent to it, is used in the study of the semantics of *CCS*-like languages.

Since category theory was used in Part I, it seems only fair to use categorical tools also for semantic purposes. Chapter 6 reviews basic notions and methods used to give a denotational semantics to recursive languages. An appropriate generalisation of standard algebraic method to category theory is developed. It is a simple exercise in application of

the general theory developed by Lehmann and Smyth, [Lehmann&Smyth 81], to the area of our particular interest.

Chapter 7 introduces a recursive language $\mathcal{L}$. It is given an axiomatic operational semantics in a manner suggested by Plotkin, [Plotkin 81]. It comes in the form of an inference system in which statements of the form $E \xrightarrow{\mu} F$ can be derived, for closed terms E, F of $\mathcal{L}$, and μ an *action symbol.*

This language is referred to as a *universal language for synchronisation*, for it is possible to interpret in it a variety of synchronisation mechanisms as derived operators.

Milner's *CCS* is considered as a specific example. A sublanguage *CCS* of $\mathcal{L}$ is defined, and is shown to have axiomatic operational semantics derivable from that of $\mathcal{L}$. We show that there are natural translations between *CCS* and *CCS* terms. One of them, from *CCS* to *CCS* extends to translation of *derivation trees* derivable in the corresponding inference systems. A modification of *CCS* in which action symbol τ from *CCS* corresponds to a special action symbol $\imath$, read 'unobservable', in $\mathcal{L}$ is also considered.

Similar technique can be applied to obtain in $\mathcal{L}$ as derived synchronisation operators from *SCCS*, [Milner 83], *TCSP*, [Hoare et. al. 84], and indeed any synchronisation operator definable by means of synchronisation algebras, [Winskel 84b].

In Chapter 8 the theory developed in Chapter 6 is applied to give $\mathcal{L}$ a categorical denotational semantics.

First, a category **LSS** of *labelled skeletal systems* is defined, and shown equivalent to the category of labelled asynchronous systems defined in Chapter 4. The categorical properties of **LSS** and its subcategories are studied. A particular subcategory $\mathcal{A}_{\mathcal{M}}$ of forward stable *c*-generated* labelled skeletal systems over an alphabet $\mathcal{M}$ is chosen as the domain of interpretation. That is we show that $\mathcal{A}_{\mathcal{M}}$ is an *ω-category*, i.e., has colimits of ω-diagrams, and the functors that interpret operators of $\mathcal{L}$ are *continuous*, i.e., preserve the colimiting cocones of ω-diagrams.

All except one of the functors involved arise from the categorical properties of **LSS**. Two of the functors are related to the categorical product and coproduct in **LSS**, respectively. One, interpreting the unique nullary operator of $\mathcal{L}$, is determined by the initial object of $\mathcal{A}_{\mathcal{M}}$. Finally, $\mathcal{A}_{\mathcal{M}}$ can be seen as a *fibre* category determined by certain functors. The interpretation of *renaming*, and *restriction* operators of $\mathcal{L}$ results from the fact that the two functors are an *op-fibration*, and a *fibration*, respectively. Arguably, the argument in the second case is not as natural as one might wish.

The problem of relating the two descriptions of $\mathcal{L}$ given in Chapters 7 and Chapter 8 is addressed in the last two Chapters of Part II.

In Chapter 9 a construction from syntax and axiomatic operational semantics of a labelled skeletal system for each closed term of $\mathcal{L}$ is presented.

First, three simple and natural principles of such a construction are suggested. Surprisingly, *CCS* with its original axiomatic operational semantics does not succumb to such procedure if one aims at achieving an agreement with denotational semantics.

Secondly, it is shown that a construction based on these principles can be accomplished for $\mathcal{L}$. This, incidentally, should help explaining the asymmetric way in which the sublanguage CCS of $\mathcal{L}$ is related to CCS in Chapter 7.

In Chapter 10 a relationship between the denotational semantics, and the syntactic construction is established.

First, we prove that the two constructions coincide for the sublanguage of $\mathcal{L}$ containing all terms without recursive subterms.

For a larger sublanguage $g\mathcal{L}$ of $\mathcal{L}$ which contains terms with *guarded recursion* a similar result is proven. This time, however, a *syntactic unfolding*, i.e., a syntactic transformation of terms, must precede the syntactic construction. Without it the syntactic construction can return a finite systems, with loops, while the denotational would produce infinite. Luckily, the syntactic unfolding is an invariant of the denotational semantics.

In the final Chapter 11 the accomplishments and deficiencies of the work are reconsidered together with open problems, and directions for future research.

Chapter 2

Preliminaries

This Chapter is devoted to the review of basic notions. Standard mathematical notation is used throughout the thesis. Nevertheless, most of the notation and terminology is collected here for the convenience of the reader. The first section is the right place to look for this kind of information.

In the next section we recall basics of category theory. The reader is expected, however, to possess some knowledge in both: elementary set theory and elementary category theory. Appropriate literature that can help fill the gaps is suggested. Subsequently, we review categories like: **Set** — the category of small sets, **Pfn** — the category of partial functions, and other categories basic from the mathematical point of view.

The last section explores properties of categories that later are of special interest. We study **pGrph**, the category of *directed graphs* with *'partial' morphisms*. To this end an equivalent category **preCat** — of *precategories* is introduced. A typical precategory is obtained when one 'forgets' about composition of arrows in a category.

The main reason behind the introduction of **preCat** is that it is easier to prove that finite limits commute with filtered colimits in **preCat**. This property being categorical holds automatically in **pGrph** as well. It will be crucial later to establish that one of the functors used to give a denotational semantics of the language studied in Part II is continuous.

In due time we shall see that there is a strong relationship between *asynchronous systems*, the object of our ultimate interest, and precategories. The former being, up to equivalence of categories, a version of the latter enriched with some extra structure. This serves, in our opinion, as yet another justification for the use of categorical language and methods in the thesis.

2.1 Conventions and Notation

We use the following general conventions related to abbreviation of many times repeated terms. Thus,

iff should be read as 'if and only if';

w.r.t., stands for 'with respect to';

w.l.o.g., abbreviates 'without loss of generality';

s.t., means 'such that'.

The end of a proof is indicated by a box, '□'. Examples are ended with a box turned 45^o (to the right), '◇'.

2.1.1 Basic Mathematics

We assume that the reader is familiar with elementary Set Theory. An introduction to any monograph on the subject, e.g., [Kunen 80] or [Yech 78] will do. There are two predefined, i.e., assumed to be intuitively clear, notions that underly Set Theory. Thus, we are interested in *collections* of certain objects. We will call the collections *sets*. Crucial, and predefined as well, is the notion of *membership*, a correspondence between objects and collections, denoted '$\in$'. This relation serves to define all the others, like inclusion of sets, in notation '$\subseteq$', and equality of sets, denoted '$=$'.

One usually stipulates that certain operations on sets are defined, e.g., in Zermelo-Frankel axiomatisation of Set Theory. We assume the reader is familiar with them, and use the standard notation to denote the *empty set* '$\emptyset$', *union* '$\cup$', *intersection* '$\cap$', *subtraction* '$-$', *powerset* '$\mathcal{P}X$', *ordered pair* '$\langle x, y\rangle$', etc. The set of non-empty subsets of a set X is denoted by $\mathcal{P}_+X$. The *cartesian product* $X \times Y$ of sets X and Y is the set of all pairs $X \times Y = \{\langle x, y\rangle \mid x \in X, y \in Y\}$. The *set of natural numbers* is $\omega = \{0, 1, 2, \ldots\}$, where numerals are given by $0 = \emptyset$, $n + 1 = \{n\} \cup n$.

We write $Z = \{x \in X \mid P(x)\}$ to denote a *subset* Z of a set X which consists of those elements of X which satisfy property $P(x)$. *Relation* is often used as a synonym of the term 'subset'. The so called Comprehension Schema of ZFC allows us to consider the definition of Z above as an acceptable method of sets construction. Problems arise when the qualifying set X is omitted in the above. This is illustrated by the Russell paradox. For $X = \{x \mid x \notin x\}$ we have:

$$X \in X \quad \text{iff} \quad X \notin X.$$

On the other hand it appears useful to consider collections of 'all sets'. The Russell paradox tells us to be cautious when dealing with them.

One solution is to introduce a stratification of collections into 'small' = proper sets, and 'large'. The latter are usually termed *classes*. Then, objects like

$$Y = \{x \mid x = x\}$$

are typical examples of *proper classes*, i.e., classes which are not sets. We avoid the paradox at the price of introducing a new kind of objects into the arena. This complicates the theory, since the right place to consider them would be at the level of metatheory. Other solutions in this direction have their drawbacks as well, see [Kunen 80] for a short discussion.

Another attempt is to introduce the concept of a *universal set*. A universal set V plays the role of collection Y discussed above. Its members are called *small* sets. It is assumed

to be closed with respect to the usual constructions on sets performed on its elements, e.g., $\mathcal{P}x \in V$, for $x \in V$. Then, implicitly, one works in some fixed universe V_0, i.e., all the sets considered are members of V_0. Then, for instance $\{x \mid x = x\}$ corresponds simply V_0. This method allows us later to introduce entities like *category of all (small) sets*, **Set**, and still remain within the framework of Set Theory.

It seems only natural to go a step further, and accept Grothendieck's axiom that every set is contained in a universal set. Then, obviously, there is a universal set V_1, s.t., $V_0 \in V_1$, and we can develop the theory of categories like **Set** while working within V_1.

To a large extent it is irrelevant which of the solutions is adopted, but to fix attention we opt for ZFC axiomatisation of Set Theory, with the axiom asserting existence of universal sets. In the future we will omit any reference to the universal set we shall be currently working in.

Whenever a specific set is introduced, like $\mathrm{Proof}_{\mathcal{L}}$ in Chapter 7, it comes with a bunch of variables to denote its typical elements. In such situations we write p, q: $\mathrm{Proof}_{\mathcal{L}}$ introducing the set and the names of its generic elements at the same time.

Sets are often given by *inductive definitions*, i.e., we define a set X to be the least set, w.r.t., the subset relation, satisfying certain closure conditions.

A proof by *induction over* X of a proposition $\forall x \in X \,.\, P(x)$ consists of showing that the set $Y = \{x \in X \mid P(x)\}$ satisfies the closure conditions. Then, since X is minimal with this property, we can conclude that $Y = X$.

Induction over the natural numbers and structural induction over term algebras are special cases of this general principle.

In Set Theory functions are defined as a kind of binary relations. A *binary relation* on X and Y is a synonym of a subset of $X \times Y$.

Given a binary relation $Q \subseteq X \times Y$, and $A \subseteq X$, $B \subseteq Y$, we define:

$\mathrm{dom}Q = \{x \in X \mid \exists y \in Y \,.\, xQy\}$ the *domain of* Q;

$\mathrm{rg}Q = \{y \in Y \mid \exists x \in X \,.\, xQy\}$ its *range*;

$Q^{-1} = \{\langle y, x\rangle \in Y \times X \mid xQy\}$ the *inverse* of Q.

$QA = \{y \in Y \mid \exists x \in A \,.\, \langle x, y\rangle \in Q\}$ the *image* of A via Q.

A *(total) function* f from a set X to a set Y, denoted $f: X \to Y$, is defined as a binary relation on X and Y such that

$\langle x, y\rangle \in f$ for every $x \in X$, some $y \in Y$, and

$\langle x, y\rangle \in f \;\wedge\; \langle x, y'\rangle \in f \;\Rightarrow\; y = y'$, for $x \in X$, $y \in Y$.

A *partial function* f from a set X to a set Y, denoted $f: X \rightharpoonup Y$, is a subset of $X \times Y$ that needs to satisfy only the second of the above conditions. In the future we shall keep calling the second kind of relations 'partial functions', and often abbreviate the term 'total function' to simple 'function', or 'map'.

The sets of total and partial functions from a set X to Y are denoted by $X \to Y$, and $X \rightharpoonup Y$, respectively.

It is often customary to denote $\langle x, y\rangle \in P$, by xPy, for $P \subseteq X \times Y$ a binary relation, and $x \in X$, $y \in Y$.

We write $f(x) = y$ or even $f\,x = y$ rather than $\langle x, y\rangle \in f$ or xfy, for a partial function $f: X \rightharpoonup Y$.

We shall frequently encounter partially defined objects. It is convenient to have a meta-language capable of dealing with this phenomenon. We introduce the following notation:

$t\downarrow$ means the term t is *defined*;

$t\uparrow$ means the term t is *undefined*.

This convention is extended to other predicates. For instance $f\,x \in X$ means that the $f\,x$ is defined, and it belongs to X. Note that $\neg f\,x \in X$, and $f\,x \notin X$ have different meanings, according to this convention.

Also, by convention, a definition of, say, a partial function in which not all possible cases of the value of argument are considered should be interpreted as defining a partial function which is undefined for all such values.

Example 2.1.1 Let $f: \omega \rightharpoonup \omega$ be defined by

$$f(n) = 4, \qquad \text{for } n \geq 3.$$

Then f is a partial map undefined for the argument values 0, 1 and 2, and defined and equal 3, for all other values of argument. ◇

An *injective* partial function $f: X \rightharpoonup Y$ is one that satisfies: $f(x)\downarrow \wedge f(x) = f(x') \Rightarrow x = x'$, for $x, x' \in X$. It is *surjective* iff $f\,X = Y$. A *bijective* function is one that is both surjective and injective.

The special functions are used to define various notions, e.g., *cardinality*. In particular, a set X is *finite* iff there exists a surjective function $f: n \to X$.

The set of all finite subsets of a set X is denoted by $\mathcal{F}in\,X$.

Sometimes λ-*notation* is used to describe anonymous (partial) functions. Thus, $\lambda x{:}X.\ t(x)$ denotes a function with domain X and some implicitly understood range Y in which all the values of the expression $t(x)$ are contained, for all $x \in X$.

Composition of partial and total functions is derived from the composition of binary relations. Given $P \subseteq X \times Y, Q \subseteq Y \times Z$ their *composition* $Q \circ P \subseteq X \times Z$ is defined as ;

$$Q \circ P = \{\langle x, z\rangle \in X \times Z \mid \exists y \in Y\,.\,\langle x, y\rangle \in X \times Y \wedge \langle y, z\rangle \in Y \times Z\}.$$

Let Q be a binary relation on X. Its n-th *power* Q^n is defined by:

$$Q^0 = id_X, \quad \text{and}$$

$$Q^{n+1} = Q \circ Q^n, \quad \text{for } n \in \omega.$$

The *transitive closure* Q^+ of Q is defined by: $Q^+ = \bigcup_{n=1}^{\infty} Q^n$.

The *reflexive and transitive closure* Q^* of Q is defined by: $Q^* = \bigcup_{n \in \omega} Q^n$.

The set of all sequences over a set A is denoted A^*. It has a structure of a *monoid* when equipped with concatenation operation and ε — the *empty sequence*, as the unit.

Every partial function $f: A \rightharpoonup B$ gives rise to $f^*: A^* \to B^*$, a *homomorphism* of monoids, i.e., a function that preserves concatenation of sequences, and the empty string. It is uniquely determined by:

$$f^*(a) = \begin{cases} f(a) & f(a)\downarrow \\ \varepsilon & \textit{otherwise} \end{cases}.$$

The last two conventions are in conflict, but it should always be obvious from the context whether f^* denotes a transitive closure of a partial function $f: A \rightharpoonup A$, or an automorphism of the monoid A^*.

For a partial function $f: X \rightharpoonup Y$, $x \in X$ and $y \in Y$, the result of *updating* of f at x by y is denoted by $f[x \mapsto y]: X \rightharpoonup Y$, and defined by:

$$f[x \mapsto y](z) = \begin{cases} y & z = x \\ f(z) & \textit{otherwise} \end{cases}.$$

For a (partial) function with a cartesian product of some sets as domain *infix* and *postfix* notation is also used. For instance in usual arithmetic: $a + b$, for $a, b \in \omega$. The function 'plus' will be denoted by $_ + _ : \omega \times \omega \to \omega$ to indicate that it has a pair of arguments, and is used in the infix notation. The same convention applies also to relations, i.e., subsets of a cartesian product. Here, $_ \in _$ could serve as a typical example.

Binary relations are often subject of our interest. Specially those defined on cartesian product $X \times X$, for some set X. Given a binary relation $\approx$ on X, i.e., $\approx \subseteq X \times X$, we call it

identity on X, and denote by id_X, iff $x \approx x' \iff x = x'$, for $x, x' \in X$;

reflexive iff $id_X \subseteq \approx$;

irreflexive iff $id_X \cap \approx = \emptyset$;

symmetric iff $\approx = \approx^{-1}$;

antisymmetric iff $\approx \cap \approx^{-1} = id_X$;

transitive iff $\approx \circ \approx \subseteq \approx$.

It is evident that $id_X \subseteq X^2$ is a function. Every subset $X' \subseteq X$ determines *inclusion function* $\lambda x{:}X'.\, x$. To stress that $f: X \to Y$ is inclusion we write $f: X \hookrightarrow Y$.

A *partial equivalence relation on* X is a symmetric and transitive binary relation on X. An *equivalence relation* is a partial equivalence which is additionally reflexive.

If Q is a binary relation then Q^+ is transitive, while Q^* is transitive and reflexive. Relation $(Q \cup Q^{-1})^*$ is the equivalence relation *generated by* Q, i.e., the least equivalence relation that contains Q.

Let $\approx$ be a partial equivalence relation on X. The set $\{y \in X \mid x \approx y\}$ of all objects related to x via $\approx$, for $x \in \mathrm{dom}(\approx)$, is called *equivalence class of* x *w.r.t.,* $\approx$, and denoted $[x]_\approx$. By $X/\approx$ we denote the set of equivalence classes of $\approx$, for $x \in \mathrm{dom} \approx$.

As a consequence of $\approx$ being symmetric and transitive we have

$$[x]_\approx = [y]_\approx \iff x \approx y, \quad \text{for } x, y \in \mathrm{dom}(\approx).$$

We can define a map $q_\approx = \lambda x{:}X.\,[x]_\approx$. This gives us a partial function $q_\approx{:}\, X \rightharpoonup X/\approx$, called *natural quotient map*. Evidently, this map is total iff $\approx$ is reflexive, i.e., an equivalence.

By a *preorder on* X we mean a reflexive and transitive relation on X. An antisymmetric preorder is called *partial order*. A pair $\mathbf{P} = \langle P, \sqsubseteq \rangle$, where $\sqsubseteq$ is a partial order on P, is called *poset*. The set ω of natural numbers ordered by inclusion is a poset.

A *dependency* on X is a reflexive and symmetric relation on X. An *independency* is an irreflexive and symmetric one.

2.2 Basic Category Theory and Basic Categories

The language of category theory is used throughout the thesis. Category theory itself is treated here as a branch of mathematics founded on set theory. It would be perhaps more along the spirit of the presentation to use it as a fundamental theory, as suggested by Lawvere. This temptation should be rejected, however, on the ground of too weak proliferation of the idea among mathematicians, yet alone in the Computer Science community.

This section is not self contained. What we intend is to give rather a quick overview of some basic notions and, what is perhaps more important, notation used later. The reader can perhaps benefit most from the elementary and illuminating review of central categorical concepts presented in Goldblatt's book on topos theory, [Goldblatt 79]. The most concise and mathematically pleasing text constantly referred to in the thesis is, undoubtedly, Mac Lane's *Categories for the Working Mathematician*, [Mac Lane 71].

The reader may find useful also other basic texts, like [Herrlich&Strecker 73] and [Arbib&Manes 75].

2.2.1 Basic Category Theory

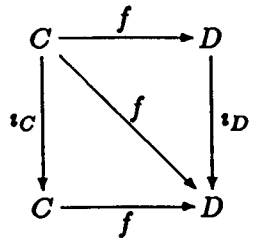
A *category* $\mathbf{C}$ is a structure $\mathbf{C} = \langle \mathbf{C}, \mathbf{Hom_C}, \partial^0, \partial^1, \imath, \circ \rangle$, where $\mathbf{C}$ is ambiguously used to denote a collection of *objects* of $\mathbf{C}$, and $\mathbf{Hom_C}$ is a collection of *arrows* or *morphisms* of $\mathbf{C}$. Next, $\imath{:}\,\mathbf{C} \to \mathbf{Hom_C}$, gives *identity arrows* corresponding to each object. We write $\imath_C$, or even id_C, rather than $\imath(C)$, for $C \in \mathbf{C}$. Functions $\partial^0, \partial^1{:}\,\mathbf{Hom_C} \to \mathbf{C}$, are called *domain* and *codomain* maps.

The relationship between identity maps and (co)domains is subject to the condition: $\partial^0 \circ \imath = id_\mathbf{C} = \partial^1 \circ \imath$, i.e., the identity morphism on C has C as its domain and codomain: $\partial^0(\imath_C) = C = \partial^1(\imath_C)$.

Finally, *composition of arrows* is a partial function: $_ \circ _ : A \times A \rightharpoonup A$. It is defined for *composable pairs* of arrows: $g \circ f\!\downarrow$ iff $\partial^1 f = \partial^0 g$. Composition is associative: $h \circ (g \circ f) = (h \circ g) \circ f$. According to our convention it means that whenever on side of the equation is defined, so is the other, and the two are equal. The last requirement is that identity arrows are identities of the composition:

$\mathfrak{i}_D \circ f = f = f \circ \mathfrak{i}_C$, for $C = \partial^0 f$, $D = \partial^1 f$.

We shall frequently write $f: C \to D$, or $C \xrightarrow{f} D$, as an abbreviation for $C = \partial^0 f \wedge D = \partial^1 f$. Also, we draw diagrams like:



In a diagram as above nodes are labelled with objects, and edges with morphisms in a category. Then paths from one node to another correspond to morphisms obtained by composition of morphisms along a path. A diagram is *commuting* if all paths between any two nodes represent the same morphisms. Thus, commutativity of the diagram above says precisely that the identity morphisms are identities of composition.

If $\mathbf{A}, \mathbf{B}$ are categories, with $\mathbf{A} \subseteq \mathbf{B}$ and $\mathbf{Hom}_\mathbf{A} \subseteq \mathbf{Hom}_\mathbf{B}$, and s.t., the identity morphisms, and morphism composition in $\mathbf{A}$ is inherited from $\mathbf{B}$ then $\mathbf{A}$ is called a *subcategory* of $\mathbf{B}$.

As a first example we consider category of small sets. We refer to it as **Set**. It has small sets as objects and (total) functions as morphisms. Composition of morphisms in **Set** is the composition of functions, and identity functions are taken as identity morphisms. It is an easy exercise to verify that this gives a category.

Another examples follow.

Every subset $\mathbf{A}$ of objects of a category $\mathbf{B}$ determines its *full* subcategory $\mathbf{A}$ when it is equipped with all morphisms between elements of $\mathbf{A}$ that exist in $\mathbf{B}$, with inherited composition, and identities.

Every poset $\mathbf{P} = \langle P, \sqsubseteq \rangle$ can be considered as a category. Take P as objects, and $\{\langle p, q \rangle \in P \times P \mid p \sqsubseteq q\}$ as arrows, with $\partial^0\langle p, q \rangle = p$ and $\partial^1\langle p, q \rangle = q$.Finally, define composition by: $\langle q, r \rangle \circ \langle p, q \rangle = \langle p, r \rangle$. In particular the poset of natural numbers is a category.

An *empty* category has no objects, and so no morphisms.

$\mathbf{1}$ is a category with one object and one, hence identity, morphism.

Every set X can be made into a *discrete* category, by taking its elements as objects, and adding required identity morphisms.

A non-empty category $\mathbf{J}$ is called *filtered* iff for every two arrows $f: I \to I'$ and $g: J \to J'$ in $\mathbf{J}$ there exists a pair of arrows $h: I' \to K$ and $h': J' \to K$ in $\mathbf{J}$. Moreover, if $I = J$ and $I' = J'$ we can assume that $h = h'$ and $h \circ f = h \circ g$.

Some classes of morphism are selected for their special properties. Let $\mathbf{A}$ be a category, and $f: A \to A'$ a morphism in $\mathbf{A}$.

f is called *mono* iff it is *left cancellable*, i.e., for $g, h: B \to A$ in $\mathbf{A}$ if $f \circ g = f \circ h$ then we have already $g = h$. Similarly, right cancellable morphisms are called *epi*.

Morphism f is *split mono* iff it has *right inverse* iff there is a morphism $g: A' \to A$ s.t., $g \circ f = id_A$. Similarly, f is *split epi* iff it has *left inverse* iff there is a morphism $g: A' \to A$ s.t., $f \circ g = id_{A'}$.

An *isomorphism*, or simply *iso*, is a morphism that has at the same time both left and right inverses.

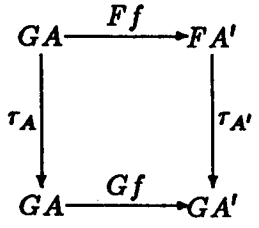
Notions of (split) monos and (split) epis are examples of *dual* notions. Every category $\mathbf{A}$ has its *dual*, denoted $\mathbf{A}^{op}$, obtained by formally reversing the domains and codomains of all arrows. For a notion defined in terms of commuting diagrams this corresponds to reversing of all the arrows in the diagrams. As the result a morphism is mono iff it is epi in the dual category. Isomorphism is an example of a *self-dual* notion.

In our presentation categories are just a kind of partial algebras. It is not surprising therefore that *functors*, their morphisms, are homomorphism of partial algebras, as studied by Burmeister in [Burmeister 82]. Actually, Burmeister considers two different notions of homomorphisms between partial algebras. They coincide in the case of categories.

Formally, a functor $F: \mathbf{A} \to \mathbf{B}$ is a pair of maps, both denoted by F, that preserve the identities, and composition of morphisms, i.e., $F(id_A) = id_{FA}$, and $F(g \circ f) = Fg \circ Ff$.

It is natural to consider the class of all functors between categories $\mathbf{A}$ and $\mathbf{B}$. It becomes a category, denoted $\mathbf{A} \to \mathbf{B}$, when a notion of a morphism between functors is defined.

Given two functors $F, G: \mathbf{A} \to \mathbf{B}$ their morphism τ, called *natural transformation* and denoted $\tau: F \to G$, is defined as a family $\tau = \{\tau_A \mid A \in \mathbf{A}\}$, s.t., for every $f: A \to A'$ in $\mathbf{A}$ the following diagram commutes:



The composition $\tau' \bullet \tau: F \to H$ of natural transformations $\tau': G \to H$ and $\tau: F \to G$, called *vertical*, is defined by $(\tau' \bullet \tau)_A = \tau'_A \circ \tau_A$, for $A \in \mathbf{A}$. The identity natural transformation id_F on $F: \mathbf{A} \to \mathbf{B}$ is defined by $id_F = \{id_{FA} \mid A \in \mathbf{A}\}$.

For $F, G: \mathbf{A} \to \mathbf{B}$, $F', G': \mathbf{B} \to \mathbf{C}$, $\tau: F \to G$ and $\tau': F' \to G'$ the *horizontal* composition $\tau' \circ \tau: F' \circ F \to G' \circ G$ is defined by: $(\tau' \circ \tau)_A = \tau'_{GA} \circ F'\tau_A = G'\tau_A \circ \tau'_{FA}$, for $A \in \mathbf{A}$. This definition is extended to functors where, with above notation, $G \circ \tau$ stands for $id_G \circ \tau$, etc.

Isomorphisms in $\mathbf{A} \to \mathbf{B}$ are called *natural isomorphisms*, or simply *natural isos*, and have isomorphisms in $\mathbf{B}$ as constituents. To indicate that $F, G: \mathbf{A} \to \mathbf{B}$ are naturally isomorphic we write $F \cong G$.

From the point of view of category theory isomorphic objects are indistinguishable. For categories themselves even this property seems too narrow. Following [Mac Lane 71] we say that a functor $F: \mathbf{A} \to \mathbf{B}$ is an *equivalence of categories*, and then categories $\mathbf{A}$ and $\mathbf{B}$ are called *equivalent*, if there exists a functor $G: \mathbf{B} \to \mathbf{A}$, s.t., $G \circ F \cong id_{\mathbf{A}}$ and $F \circ G \cong id_{\mathbf{B}}$. Those properties that are shared by equivalent categories are called *categorical*.

A small category has a small set of objects and a small set of morphisms. We are entitled to consider category of all small categories, denoted **Cat**. It has small categories as objects and functors between them as morphisms.

For a small category **J**, a *diagram of type* **J** in a category **C**, or simply **J**-*diagram* in **C**, is a functor $\Gamma: \mathbf{J} \to \mathbf{C}$. A *cone* ϕ of a diagram Γ, denoted $\phi: C \to \Gamma$, consists of a *vertex* $C \in \mathbf{C}$, and a family of morphisms $\phi = \{\phi_j: C \to \Gamma j \mid j \in \mathbf{J}\}$ in **C**, s.t.,

$$\Gamma f \circ \phi_j = \phi_k, \qquad \text{for } f: j \to k \in \mathbf{Hom_J}.$$

Cone $\phi: C \to \Gamma$ is *limiting* for a **J**-diagram Γ iff for any other cone $\psi: D \to \Gamma$ there exists a unique *mediating* morphism $h: D \to C$, i.e., one that satisfies:

$$\phi_j \circ h = \psi_j, \qquad \text{for } j \in \mathbf{J}.$$

Dually, we arrive at notions of *cocone* and *colimit*.

Among many examples of limits we mention here *terminal objects* = limits of the empty diagram, and *products* = limits of diagrams over discrete categories. *Equalizers* are limits of diagrams over a category with two objects, say 0 and 1, and with two non-identity morphisms $f, g: 0 \to 1$. Dually, we obtain *initial objects*, *coproducts* and *coequalizers*. Also ω-*colimits*, i.e., colimits of ω-diagrams, will be of special interest.

In case of posets, limits correspond to *greatest lower bounds*, abbr.: *glb*-s, while colimits to *least upper bounds*, abbr.: *lub*-s.

We use $\lim_{\mathbf{J}}\Gamma$ to denote the vertex C of a limiting cone $\phi: C \to \Gamma$ of the **J**-diagram Γ, and $\mathrm{colim}_{\mathbf{J}}\Gamma$ for the vertex of a colimiting cocone.

A category **C** *has* **J**-(co)limits iff every **J**-diagram has a (co)limiting (co)cone. It is said to be *finitely, resp., small, (co)complete* iff it has **J**-(co)limits, for every finite, resp., small, category **J**.

Next, we consider the interplay between (co)limits and functors. Three situations are often encountered.

Let $F: \mathbf{C} \to \mathbf{D}$ be a functor. We say F *preserves* limits of type **J** iff for any **J**-diagram Γ in **C**, and any limiting cone $\phi: C \to \Gamma$ the cone $F\phi = \{F\phi_j \mid j \in \mathbf{J}\}$ is limiting in **D**, where $F\phi: FC \to F \circ \Gamma$.

We say F *reflects* limits of type **J** iff for any **J**-diagram Γ in **C**, and every cone $\phi: C \to \Gamma$, if the cone $F\phi: FC \to F \circ \Gamma$ is limiting in **D** then ϕ is already a limiting cone in **C**.

Finally, F *creates* **J**-limits in **C** iff for any **J**-diagram Γ in **C**, and any limiting cone $\psi: D \to F \circ \Gamma$ there exists a unique limiting cone $\phi: C \to \Gamma$, such that '$\psi = F\phi$', i.e., $\psi_j = F\phi_j$, for $j \in \mathbf{J}$.

Dually, we arrive at notions of a functor preserving, reflecting and creating **J**-colimits, for a small category **J**.

The above properties are interrelated.

Fact 2.2.1 *Let* $F: \mathbf{C} \to \mathbf{D}$ *be a functor, and* **I**, **J** *small categories. Then*

(i) If F *creates* **J**-*(co)limits and* **D** *has them then* **C** *has* **J**-*(co)limits.*

(ii) If F creates **J**-*(co)limits then F reflects them.*

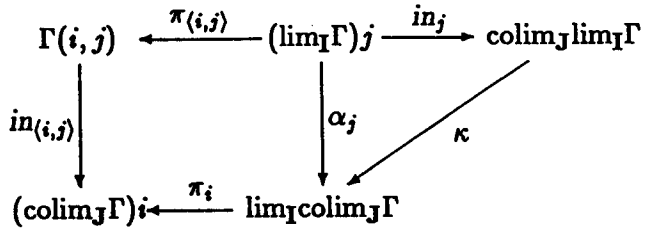
(iii) If F creates **J**-*(co)limits and* **D** *has* **J**-*(co)limits then F preserves them;*

(iv) Assume **D** *has* **I**-*limits and* **J**-*colimits, and that those commute in* **D**. *If F creates both then* **C** *has them, and* **I**-*limits commute with* **J**-*colimits in* **C**.

Proof. The first three properties follow quickly from the definitions. We give a sketch of the proof of the last one.

(iv) Let $\Gamma: \mathbf{I} \times \mathbf{J} \to \mathbf{C}$ be a diagram in **C**. Since **C** has **I**-limits and **J**-colimits, by (i), we obtain a **J**-diagram $\lim_{\mathbf{I}}\Gamma$, and a **I**-diagram $\mathrm{colim}_{\mathbf{J}}\Gamma$, from the categorical properties of limits and colimits.

Next, we obtain the following commutative diagram



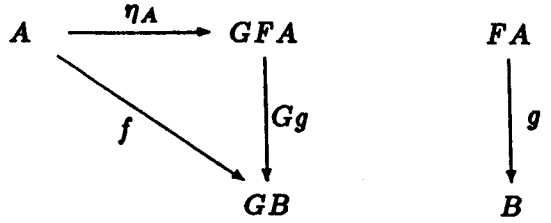
Above α_j-s are defined from the categorical properties of limits. It follows that they constitute a cocone $\alpha: \lim_{\mathbf{I}}\Gamma \to \lim_{\mathbf{I}}\mathrm{colim}_{\mathbf{J}}\Gamma$. Morphism κ is defined as the mediating morphism.

Since F preserves **I**-limits and **J**-colimits, by (iii), we obtain an identical commuting diagram in **D** with $F \circ \Gamma$ in place of Γ. By the assumption **I**-limits commute with **J**-colimits in **D**, i.e., $F\kappa$ is an isomorphism. It follows that $F\alpha: F\lim_{\mathbf{I}}\Gamma \to F\lim_{\mathbf{I}}\mathrm{colim}_{\mathbf{J}}\Gamma$ is a colimiting cocone in **D**. Thus, α is colimiting in **C**, since F reflects **J**-colimits by (ii). Therefore, κ is an isomorphism. □

We should stress that the term 'commute' means not only that with the notation as above $\lim_{\mathbf{I}}\mathrm{colim}_{\mathbf{J}}\Gamma$ is isomorphic to $\mathrm{colim}_{\mathbf{J}}\lim_{\mathbf{I}}\Gamma$, but that the *canonical* morphism κ constructed above is the required isomorphism.

Remark. An easier proof can be given if we assume moreover that F reflects isomorphisms, i.e., f is an iso whenever Ff is iso. It is not satisfactory for us as we shall apply the Fact in cases where this condition is not satisfied.

Very often we shall discuss *adjunctions* between functors. Given two functors $F: \mathbf{A} \to \mathbf{B}$ and $G: \mathbf{B} \to \mathbf{A}$ we say that F is *left adjoint* to G, and G *right adjoint* to F, iff for any $A \in \mathbf{A}$ there exits η_A s.t., for any $B \in \mathbf{B}$ and $f: A \to GB$, there is precisely one $g \in \mathbf{Hom_B}$ that makes the following diagram commute:



From the general theory it follows that $\eta = \{\eta_A \mid A \in \mathbf{A}\}$ is a natural transformation, called the *unit* of the adjunction. A number of results equivalent to the above definition is known, see [Mac Lane 71], but we shall use only this one.

The property of being a left (right) adjoint of a functor is categorical, since any two functors left (right) adjoint to a functor must be naturally isomorphic. The proof of this fact, and also of the following fact used later very often can be found in [Mac Lane 71].

Proposition 2.2.2 *(i) Any functor $G: \mathbf{A} \to \mathbf{B}$ which is right adjoint preserves all limits that exist in* $\mathbf{A}$.

(ii) Any functor $G: \mathbf{A} \to \mathbf{B}$ which is left adjoint preserves all colimits that exist in $\mathbf{A}$.

Proof. The first property is found in [Mac Lane 71] as Theorem 1, p.114. The second is a dual version of the first. □

A special kind of adjoint situation arises when one of the functors is a full inclusion. If, with notation as above, F is a full inclusion with a right adjoint G then $\mathbf{A}$ is called *full coreflective subcategory*, and G is called *coreflector*.

The meaning of the terms 'coreflection', and 'coreflector' is slightly extended. With the same notation $\mathbf{A}$ is allowed to be equivalent to a full coreflective subcategory $\mathbf{B}'$ of $\mathbf{B}$, while G is a composition of a right adjoint to the inclusion $\mathbf{B}' \hookrightarrow \mathbf{B}$ with a natural equivalence $G': \mathbf{B}' \to \mathbf{A}$.

One of the important properties of the coreflection is that the coreflector is a split epi in **Cat**, since we have the following result.

Theorem 2.2.3 *Let $F: \mathbf{A} \to \mathbf{B}$, $G: \mathbf{B} \to \mathbf{A}$ with F left adjoint to G, and F full. Then $G \circ F \cong id_{\mathbf{A}}$ iff G is a coreflector.*

Proof. $G \circ F \cong id_{\mathbf{A}}$ implies G is a coreflector

Let $F': \mathbf{A} \to F\mathbf{A}$, and $G': F\mathbf{A} \to \mathbf{A}$ denote restrictions of F and G, respectively. Then $G' \circ F' \cong id_{\mathbf{A}}$ follows from the assumption. Therefore, $F' \circ G' \circ F' \cong F'$. If $\tau: F' \to F' \circ G' \circ F'$ is a natural isomorphism then let $\theta_{FA} = \tau_A$, for $FA \in F\mathbf{A}$. Now, F full implies *theta* is a well-defined natural transformation $\theta: id_{F\mathbf{A}} \to F' \circ G'$. Clearly, it is a natural isomorphism as well, i.e., $id_{F\mathbf{A}} \cong F' \circ G'$.

Thus, F' is an equivalence of categories, and by Theorem 1 p.91 [Mac Lane 71], it is right adjoint to G'.

Hence, $F \circ G' \dashv F' \circ G$, since adjunctions compose, see Theorem 1 p.101 ibid.

But $F \circ G'$ is naturally isomorphic to inclusion $F\mathbf{A} \hookrightarrow \mathbf{B}$. The result now follows, since $F\mathbf{A}$ is a full subcategory of $\mathbf{B}$ as an image of a full functor.

G is a coreflector implies $G \circ F \cong id_{\mathbf{A}}$

Let $\mathbf{B}'$ be a full coreflective subcategory of $\mathbf{B}$ equivalent to $\mathbf{A}$. Let us consider functor $F': \mathbf{A} \cong \mathbf{B}' \hookrightarrow \mathbf{B}$. As a composition of an equivalence of categories with a full

inclusion functor it is both: full and faithful. Also, F' is a left adjoint to G. Hence, by a result dual to Theorem 1, p.88 in [Mac Lane 71] it follows that $G \circ F' \cong id_{\mathbf{A}}$. But $F \cong F'$ since both are left adjoint to G. The result follows. □

As an immediate consequence of the above Theorem we see that coreflections compose.

Reflective subcategories are characterised as having left adjoints to the inclusion functors.

Next, we quickly review some of the categorical properties of, and constructions in the basic categories. By basic we understand here those which play a central role in mathematics, such as: **Set** - category of (small) sets; $\mathbf{Set}_*$ - category of (small) pointed sets, **Pfn** - category of sets with partial functions, etc.

Their properties are well known, and we do not waste time to review all of them, all the more their proofs. More attention is devoted to those which are usually not covered in the standard literature.

2.2.2 Category of Small Sets: Set

Category of small sets which was mentioned already is perhaps the most intensively studied category. Below we review some of its categorical properties. Attention is paid to the categorical relevance of various constructions on sets.

$\emptyset$ the empty set, is (the) initial object in Set.

$1_{\mathbf{Set}} = \{\star\}$ a 1-element set, is a terminal object in Set.

$X \times Y$ the cartesian product of X and Y, is a categorical product in Set.
$\pi_X: X \times Y \to X, \pi_Y: X \times Y \to Y$ the usual projections, are projections in the categorical sense. Recall that, e.g., $\pi_X(\langle x, y\rangle) = x$, for $x \in X$.

$\prod \mathcal{X} = \prod_{k \in K} X_k$ the cartesian product of the family $\mathcal{X} = \{X_k \mid k \in K\}$, is a categorical product of $\mathcal{X}$.
$\pi_k: \prod \mathcal{X} \to X_k$ the usual projections, are projections in the categorical sense.

Given a pair $f, g: X \to Y$ of functions their equalizer is an inclusion map $e: E \hookrightarrow X$, where $E = \{x \in X \mid f(x) = g(x)\}$.

$X + Y$ the disjoint union: $X + Y = \{0\} \times X \cup \{1\} \times Y$, is a categorical coproduct in Set.
$in_X: X \to X + Y, in_Y: Y \to X + Y$ categorical injections, where for instance in_X is given by: $in_X(x) = \langle 0, x\rangle$.

$\coprod \mathcal{X} = \coprod_{k \in K} X_k$ the disjoint union of a family $\mathcal{X} = \{X_k \mid k \in K\}$, is a categorical coproduct: $\coprod \mathcal{X} = \bigcup \{k\} \times X_k$.
$in_k: X_k \to \coprod \mathcal{X}$ are categorical injections, given by: $in_k(x) = \langle k, x\rangle$.

Given a pair of functions $f, g: X \to Y$ their coequalizer $q: Y \to Y/\approx$ is the natural quotient map $q(y) = [y]_\approx$, where $\approx \subseteq Y \times Y$ is an equivalence relation generated by the set $\{\langle f(x), g(x)\rangle \mid x \in X\}$.

It is well known that **Set** is both: small complete and small cocomplete, i.e., that every small diagram in **Set** has a limit and a colimit.

It is worthwhile to dedicate a bit more attention to a construction of ω-colimits in **Set**. Assume $\Gamma = X_0 \xrightarrow{f^0} X_1 \xrightarrow{f^1} \dots$, is a ω-diagram in **Set**. Put

$$X_\rightarrow = (\coprod_{n\in\omega} X_n)/\approx, \quad \text{and define} \quad \phi_n: X_n \to X_\rightarrow \text{ by } \phi_n(x) = [x]_\approx$$

where $\approx \subseteq (\coprod_{n\in\omega} X_n)^2$ is an equivalence relation generated by the set $\{\langle x, f^n(x)\rangle \mid n \in \omega, x \in X_n\}$. This relation has an alternative characterisation.

Let us introduce the following notation, for $m \in \omega$

$$f^{m,m} = id_{X_m}, \text{ and}$$
$$f^{m+n+1,m} = f^{n+1} \circ f^{m+n,m}, \qquad \text{for } n \in \omega.$$

Then one can easily show that for $x \in X_m, y \in X_n$ we have

$$x \approx y \iff \exists k \in \omega \,.\, k \geq m,n \wedge f^{k,m}(x) = f^{k,n}(y)$$

With the above definitions one can easily show that the following proposition holds.

Proposition 2.2.4 *$\phi = \{\phi_n: X_n \to X_\rightarrow \mid n \in \omega\}$ is a colimiting cocone $\phi: \Gamma \to X_\rightarrow$ in* **Set**. *For any other cocone $\psi: \Gamma \to Y$ the mediating map $f: \phi \to \psi$, i.e., the unique function $f: X_\rightarrow \to Y$ such that*

$$f \circ \phi_n = \psi_n \qquad \text{for } n \in \omega,$$

is given by: $f([x]_\approx) = \psi_n(x)$, for $x \in X_n$.

Proof. Elementary. □

Remark. One can recognise the construction presented as a specialisation of the general construction of (small) colimits in (small) cocomplete category by means of coproducts and coequalisers. Compare dual construction in the proof of Theorem 1, p. 109 in [Mac Lane 71].

We finish the discussion of the properties of the category of sets by invoking an important property it enjoys.

Proposition 2.2.5 *Finite limits commute with filtered colimits in* **Set**.

Proof. See e.g., Theorem 1 p211 in [Mac Lane 71]. □

2.2.3 Category of Pointed Sets: $\mathbf{Set}_\star$

Category of (small) *pointed sets*, denoted $\mathbf{Set}_\star$, has as objects nonempty sets with *points*, i.e., pairs consisting of a set together with its designated element. Morphisms in $\mathbf{Set}_\star$ are those functions that preserve the designated points.

First, note that the evident functor $\mathbf{U}: \mathbf{Set}_\star \to \mathbf{Set}$ that 'forgets about the points', i.e., sends pairs $\langle X, x\rangle$ to X, creates limits, but not all colimits. In particular we have:

$\langle \mathbf{1}_{\mathbf{Set}}, \star \rangle$ is a terminal, but also an initial object of $\mathbf{Set}_\star$. Hence, a *zero-object.*

$\langle X \times Y, \langle x, y \rangle \rangle$ is the categorical product of $\langle X, x \rangle$ and $\langle Y, y \rangle$ with usual projections, since they preserve points. Hence, U creates binary products. It is left to the reader to check that it creates arbitrary (small) products and equalisers, and so all (small) limits.

$\langle X, x \rangle \amalg \langle Y, y \rangle$ a coproduct in $\mathbf{Set}_\star$, is defined as an *amalgamated* sum: we take disjoint union of X and Y, and identify x with y. The injections are as previously, modulo the identification. In a similar way one can define coproduct of any small family in $\mathbf{Set}_\star$.

Contrary to the last case, the coequalisers, and more importantly ω-colimits, are created via U. To see that the last claim is true let us consider a ω-diagram $\Gamma = \langle X_0, x_0 \rangle \xrightarrow{f^0} \langle X_1, x_1 \rangle \xrightarrow{f^1} \dots$ in $\mathbf{Set}_\star$. It is enough to note that we can take the colimiting cocone of the underlying sets in $\mathbf{Set}$, and pick up the element: $[x_0]_\approx$, as the point. Clearly, what we get is a colimiting cocone in $\mathbf{Set}_\star$.

2.2.4 Category of Partial Functions: Pfn

The category **Pfn** of *partial functions* is defined as **Set**, but with partial functions taking place of morphisms. It has sets, as before, as objects. Identity functions are again identity morphisms, and composition of partial functions is taken as composition of morphisms in **Pfn**. Therefore, **Set** is a subcategory of **Pfn**.

Next, **Pfn** has the same categorical properties as $\mathbf{Set}_\star$, since the two are equivalent. For we can define two functors:

$$\textsc{Pfn}\colon \mathbf{Set}_\star \to \mathbf{Pfn} \quad \text{where}$$

$$\textsc{Pfn}(\langle X, x \rangle) = X - \{x\} \quad \text{on objects, and}$$
$$\textsc{Pfn}(f\colon \langle X, x_0 \rangle \to \langle Y, y_0 \rangle)(x) = f(x), \qquad \text{if } f(x) \neq y_0;$$

and one in the opposite direction

$$\textsc{Set}_\star\colon \mathbf{Pfn} \to \mathbf{Set}_\star \quad \text{where}$$

$$\textsc{Set}_\star(X) = X + \{\star\} = X + \mathbf{1}_{\mathbf{Set}} \quad \text{on objects, and}$$
$$\textsc{Set}_\star(f\colon X \rightharpoonup Y)(x) = \begin{cases} \star & f(x)\uparrow \\ f(x) & \text{otherwise} \end{cases}.$$

One readily sees that

$$\textsc{Set}_\star \circ \textsc{Pfn} \cong id_{\mathbf{Set}_\star} \qquad \text{and} \qquad \textsc{Pfn} \circ \textsc{Set}_\star \cong id_{\mathbf{Pfn}}$$

As a consequence **Pfn** is (small) complete and cocomplete. Moreover, finite products commute with ω-colimits.

For the record let us notice that the coproducts in **Pfn** coincide with those in **Set**: they are disjoint unions with usual (total) injections. The construction of categorical products is more complicated. For instance the binary product of sets X and Y has the form: $X \amalg_{\mathrm{p}} Y = X + X \times Y + Y$, and the projections are really partial maps, e.g., $\pi_X\colon X \amalg_{\mathrm{p}} Y \rightharpoonup X$ is given by:

$$\pi_X(z) = \begin{cases} x & z = in_X(x) \\ x & z = \langle x, y \rangle \in X \times Y \end{cases} .$$

Note that according to our convention $\pi_X(z)$ is undefined for $z = in_Y(y)$, $y \in Y$.

Finally, let us see what the construction of ω-colimits looks like after translation via PFN. Let $\Gamma = X_0 \xrightarrow{f^0} X_1 \xrightarrow{f^1} \ldots$, be a ω-diagram in **Pfn**. Put

$$X_\rightarrow = (\coprod_{n \in \omega} X_n)/\approx, \quad \text{where}$$

$\approx \subseteq X_\rightarrow \times X_\rightarrow$ is a partial equivalence relation given by

$$x \in \mathrm{dom}(\approx) \iff f^{n,m}(x)\downarrow, \qquad \text{for } m \leq n, x \in X_m$$

$$x \approx y \iff x, y \in \mathrm{dom}(\approx) \wedge \exists k \geq m, n \,.\, f^{k,m}(x) = f^{k,n}(y), \quad \text{for } x \in X_m, y \in X_n.$$

The first condition restricts attention to those elements of $\coprod_{n \in \omega} X_n$ for which every appropriate sequence of subsequent applications of f^n-s gives defined values. As once already we used above $f^{n,m}$ to denote the composition $f^n \circ \ldots \circ f^m$.

Finally, for $n \in \omega$, define $\phi_n : X_n \rightharpoonup X_\rightarrow$ by

$$\phi_n(x) = [x]_\approx, \qquad \text{for } x \in X_n \cap \mathrm{dom}(\approx).$$

Again, ϕ_n are partial in general, since they are undefined for $x \in X_n - \mathrm{dom}(\approx)$.

And, as before, we obtain the following fact.

Proposition 2.2.6 *$\phi = \{\phi_n : X_n \rightharpoonup X \mid n \in \omega\}$ is a colimiting cocone $\phi : \Gamma \to X$ in* **Pfn**. *For any other cocone $\psi : \Gamma \to Y$ the mediating (partial) map $f : \phi \rightharpoonup \psi$, i.e., the unique (partial) function $f : X \rightharpoonup Y$ such that*

$$f \circ \phi_n = \psi_n \qquad \text{for } n \in \omega,$$

is given by: $f([x]_\approx) = \psi_n(x)$, for $x \in X_n$.

2.2.5 Category of Finitary Multisets: $\mu - \mathbf{Set}$

A *multiset χ on a set X*, abbr., μ-set, is a mapping $\chi : X \to \omega$. We call it *finitary* iff $\{x \in X \mid \chi x > 0\}$ is finite. The set of all finitary μ-sets on X is denoted by μX.

Let us assume that the set μX of finitary μ-sets was fixed. The ordering, and arithmetical operations on ω extend on finitary multisets, e.g.:

$$\chi < \xi \quad \text{iff} \quad \chi x < \xi x, \quad \text{for } x \in X,$$

$$\chi + \xi = \lambda x{:}X.\, (\chi x + \xi x), \quad n\chi = \lambda x{:}X.\, n\chi(x), \quad \text{etc.}$$

In particular, $0\chi = \emptyset$, the *empty μ-set*, defined by: $\emptyset = \lambda x{:}X.\, 0$.

Every element x of X determines a *singleton* μ-set $\hat{x} \in \mu X$ given by:

$$\hat{x}(x') = \begin{cases} 1 & x = x' \\ 0 & \text{otherwise} \end{cases} .$$

This gives an *inclusion map* $\widehat{\ }: X \to \mu X$.

Obviously, every finitary μ-set χ has a unique formal representation: $\chi = \sum_{x \in X} \alpha_x \hat{x}$ where $\alpha_x \in \omega$, for $x \in X$, in which only finite number of coefficients α_x are non-zero. And every such representation determines a finitary μ-set.

The morphisms of μ-sets are called *multifunctions*, abbr., μ-function. A μ-function f is a function $f: \mu X \to \mu Y$ that preserves the sum operation on μ-sets, i.e., such that given $\chi, \xi \in \mu X$ we have: $f(\chi + \xi) = f\chi + f\xi$, and hence $f\emptyset = \emptyset$.

Category of finitary multisets, denoted $\mu - \mathbf{Set}$, is the subcategory of *cset* with sets of finitary μ-sets generated by a set as objects, and μ-functions as morphisms.

From the uniqueness of representation of finitary μ-sets it follows that any function $f: X \to \mu Y$ has a unique extension to a μ-function $\mu f: \mu X \to \mu Y$ s.t.: $f = \mu f \circ \widehat{\ }$. In other words the operation $\mu - _$ on sets and functions is functorial, and moreover a left adjoint to the inclusion $\mu - \mathbf{Set} \hookrightarrow \mathbf{Set}$. Hence, $\mu - \mathbf{Set}$ is a reflective subcategory of **Set**.

Also, every partial function $f: X \rightharpoonup Y$ gives rise to a μ-function $\mu f: \mu X \to \mu Y$ uniquely determined by:

$$\mu f \hat{x} = \begin{cases} \emptyset & f x\uparrow \\ \widehat{f x} & \text{otherwise} \end{cases} .$$

This determines a functor, also denoted by $\mu - _$, from **Pfn** to $\mu - \mathbf{Set}$. The reader may verify that it has a right adjoint as well.

Algebraically, finitary multisets are *abelian monoids* — simple algebras, with multifunctions being homomorphisms of these algebras. They have a constant 0 and a commutative and associative binary operation $_ + _$ s.t., $x + 0 = 0 + x = x$. The adjoint situations mentioned above extend to adjoint situations between **Set**, resp., **Pfn**, on one side, and the category of such algebras on the other.

2.3 Special Categories

2.3.1 Category of Precategories

We introduce and study properties of the category of (small) *precategories*. These are very useful later as an alternative to the category of *graphs with 'partial maps'* – the category of our main interest.

The reader is warned that the category we deal with is different from the category of small graphs and their structure preserving functions which is sometimes referred to as category of precategories, see for instance [Mac Lane 71]. In fact the latter is a subcategory of the object of our studies.

Definition 2.3.1 A *precategory* **C** is a 5-tuple $\mathbf{C} = \langle C, A, \partial^0, \partial^1, \imath \rangle$, where:

C, is a set of *objects*;

A, is a set of *arrows*;

$\partial^0, \partial^1: A \to C$, are *domain* and *codomain* maps;

$\imath: C \to A$, gives *identity arrows* corresponding to each object;
such that the following condition is satisfied:

(i) $\partial^0 \circ \imath = \partial^1 \circ \imath = id_C$

We shall refer to items building a precategory $\mathbf{C}$ by adding subscript $\mathbf{C}$ to generic term. Thus, e.g., $C_{\mathbf{C}}$ denotes the set of objects of $\mathbf{C}$, while $\partial^1_{\mathbf{C}}$ the codomain map in $\mathbf{C}$, etc.

Now, we introduce a notion of morphisms between precategories.

Definition 2.3.2 A *morphism* $f: \mathbf{C} \to \mathbf{D}$ of precategories is a pair $f = \langle f_C, f_A \rangle$ where

$f_C: C_{\mathbf{C}} \to C_{\mathbf{D}}$ and $f_A: A_{\mathbf{C}} \to A_{\mathbf{D}}$ are (total) functions,

such that

(i) $f_C \circ \partial^i_{\mathbf{C}} = \partial^i_{\mathbf{D}} \circ f_A$, for $i = 0, 1$;

(ii) $f_A \circ \imath_{\mathbf{C}} = \imath_{\mathbf{D}} \circ f_C$.

The pairs of identity functions form the *identity morphisms*.
The *composition* of morphisms is defined componentwise.

Algebraically, precategories are certain 2-sorted algebras, while their morphisms are simply homomorphisms of these algebras. The following result is therefore immediate.

Fact 2.3.3 *Precategories and their morphisms form a category, denoted* **preCat**.

Proof. Obvious. □

The relationship with Cat

The terminology chosen above is, hopefully, self-explanatory. First, let us note that every category gives a precategory. It is enough to forget about composition of morphisms. This describes an object part of a forgetful functor $\mathbf{U}: \mathbf{Cat} \to \mathbf{preCat}$.

Conversely, we can obtain a category from a given precategory if we define composition of arrows. As in the case of ordinary graphs, see e.g., [Mac Lane 71, p. 49], this can be done in the universal way. One should define the arrows as in the construction quoted above, i.e., as finite paths, with an extra requirement that the identity arrows of a precategory must become identities of the composition in the category one constructs. In effect what we construct is an object part of a left-adjoint to the forgetful functor mentioned at the beginning.

Properties of preCat

Next, we study categorical properties of **preCat**. This is done as follows. We consider the evident forgetful functor $\mathbf{U}: \mathbf{preCat} \to \mathbf{Set}^2$ and show that it creates many categorical properties of **preCat**. We can conclude that **preCat** is not only complete and cocomplete, but that the finite limits commute in **preCat** with filtered colimits as well.

Formally, we define $\mathbf{U}: \mathbf{preCat} \to \mathbf{Set}^2$:

$\mathrm{U}(\mathbf{C}) = \langle C_{\mathbf{C}}, A_{\mathbf{C}} \rangle$, on objects, and

$\mathrm{U}(f) = f$, on morphisms, i.e., we take the same pair of maps, but considered as morphisms in different categories.

Fact 2.3.4 U *defined above is a functor.*

Proof. Omitted. □

U Creates Products

Let $\mathcal{F}$ be a (small) family of precategories. We consider its image via U, i.e., the family $\mathrm{U}(\mathcal{F}) = \{\mathrm{U}(\mathbf{C}) \mid \mathbf{C} \in \mathcal{F}\}$. It has the product $\prod \mathrm{U}(\mathcal{F})$ in $\mathbf{Set}^2$ computed componentwise:

$$\prod \mathrm{U}(\mathcal{F}) = \langle C_\Pi, A_\Pi \rangle$$

where $C_\Pi = \prod_{\mathbf{C} \in \mathcal{F}} C_{\mathbf{C}}$ and $A_\Pi = \prod_{\mathbf{C} \in \mathcal{F}} A_{\mathbf{C}}$. Moreover it has usual projections:

$$\pi_{\mathbf{C}} = \langle \pi^{\mathbf{C}}_C, \pi^{\mathbf{C}}_A \rangle : \prod \mathrm{U}(\mathcal{F}) \to \mathrm{U}(\mathbf{C}),$$

where $\pi^{\mathbf{C}}_C : C_\Pi \to C_{\mathbf{C}}$ and $\pi^{\mathbf{C}}_A : A_\Pi \to A_{\mathbf{C}}$ are projections in $\mathbf{Set}$.

This data can be enriched to a precategory structure in such a way that the projections become morphisms in $\mathbf{preCat}$. We define $\partial^i_\Pi : \prod_{\mathbf{C} \in \mathcal{F}} C_{\mathbf{C}} \to \prod_{\mathbf{C} \in \mathcal{F}} A_{\mathbf{C}}$, for $i = 0, 1$ and $\mathfrak{i}_\Pi : \prod_{\mathbf{C} \in \mathcal{F}} A_{\mathbf{C}} \to \prod_{\mathbf{C} \in \mathcal{F}} C_{\mathbf{C}}$ from the categorical properties of products. Namely, we demand that they are the unique maps that satisfy:

$$\begin{array}{ccc} \prod_{\mathbf{C}\in\mathcal{F}} C_{\mathbf{C}} & \xrightarrow{\partial^i_\Pi} & \prod_{\mathbf{C}\in\mathcal{F}} A_{\mathbf{C}} \\ \downarrow{\scriptstyle \pi_{C_{\mathbf{C}}}} & & \downarrow{\scriptstyle \pi_{A_{\mathbf{C}}}} \\ C_{\mathbf{C}} & \xrightarrow[\partial^i_{\mathbf{C}}]{} & A_{\mathbf{C}} \end{array} \quad \text{and} \quad \begin{array}{ccc} \prod_{\mathbf{C}\in\mathcal{F}} C_{\mathbf{C}} & \xleftarrow{\mathfrak{i}_\Pi} & \prod_{\mathbf{C}\in\mathcal{F}} A_{\mathbf{C}} \\ \downarrow{\scriptstyle \pi_{C_{\mathbf{C}}}} & & \downarrow{\scriptstyle \pi_{A_{\mathbf{C}}}} \\ C_{\mathbf{C}} & \xleftarrow[\mathfrak{i}_{\mathbf{C}}]{} & A_{\mathbf{C}} \end{array} \quad \text{commute, for } \mathbf{C} \in \mathcal{F}.$$

Now, because of this categorical definition, the condition of the definition of a precategory, Definition 2.3.1(i), is automatically satisfied. That is we have defined a precategory:

$$\prod \mathcal{F} = \langle C_\Pi, A_\Pi, \partial^0_\Pi, \partial^1_\Pi, \mathfrak{i}_\Pi \rangle.$$

On the other hand the diagrams above say nothing else but that the pairs:

$$\pi_{\mathbf{C}} = \langle \pi_{C_{\mathbf{C}}}, \pi_{A_{\mathbf{C}}} \rangle, \quad \text{for } \mathbf{C} \in \mathcal{F}$$

are morphisms of precategories. To conclude that we have obtained a categorical product in $\mathbf{preCat}$ it is enough to check that the unique mediating pair of maps $h = \langle h_C, h_A \rangle : \mathrm{U}(\mathbf{B}) \to \mathrm{U}(\prod calFA)$ for a cone $\{g^{\mathbf{C}} : \mathbf{B} \to \mathbf{C} \mid \mathbf{C} \in \mathcal{F}\}$ is indeed a morphism of precategories. This is left to the reader.

The results of the discussion above can be collected in the form of the following lemma.

Lemma 2.3.5 $\mathbf{Set}^2$ *has small products and* U *creates them in* $\mathbf{preCat}$. *Thus,* $\mathbf{preCat}$ *has small products.*

U Creates Equalizers

Next, we turn our attention to the equalizers in **preCat**.

Let $\mathbf{C}, \mathbf{D}$ be precategories, $f, g: \mathbf{C} \to \mathbf{D}$ their morphisms. We consider two derived diagrams in **Set**:

$$f_C, g_C: C_{\mathbf{C}} \to C_{\mathbf{D}} \quad \text{and} \quad f_A, g_A: A_{\mathbf{C}} \to A_{\mathbf{D}}.$$

Their equalizers in **Set** are determined by taking:

$$C = \{c \in C_{\mathbf{C}} \mid f_C(c) = g_C(c)\} \quad \text{with } e_C: C \hookrightarrow C_{\mathbf{C}}, \text{ and}$$

$$A = \{a \in A_{\mathbf{C}} \mid f_A(a) = g_A(a)\} \quad \text{with } e_A: A \hookrightarrow A_{\mathbf{C}}.$$

We note that then: $\imath_{\mathbf{C}}(C) \subseteq A$, since

$$\begin{aligned} f_A(\imath_{\mathbf{C}}(c)) &= \imath_{\mathbf{D}}(f_C(c)) && \text{by Definition 2.3.2(ii)} \\ &= \imath_{\mathbf{D}}(g_C(c)) && \text{since } c \in C \\ &= g_A(\imath_{\mathbf{C}}(c)) && \text{by Definition 2.3.2(ii) again.} \end{aligned}$$

Similarly, $\partial^i_{\mathbf{C}}(A) \subseteq C$, for $i = 0, 1$, since

$$\begin{aligned} f_C(\partial^i_{\mathbf{C}}(a)) &= \partial^i_{\mathbf{D}}(f_A(a)) && \text{by Definition 2.3.2(i)} \\ &= \partial^i_{\mathbf{D}}(g_A(a)) && \text{since } a \in A \\ &= g_C(\partial^i_{\mathbf{C}}(a)) && \text{by Definition 2.3.2(i) again.} \end{aligned}$$

Thus, we can define $\partial^0, \partial^1: A \to C$, and $\imath: C \to A$, by:

$$\partial^i(a) = \partial^i_{\mathbf{C}}(a), \quad \text{for } a \in A, i = 0, 1, \text{ and}$$

$$\imath(c) = \imath_{\mathbf{C}}(c), \quad \text{for } c \in C.$$

It is therefore evident that $\mathbf{E} = \langle C, A, \partial^0, \partial^1, \imath \rangle$ is a precategory. It is equally evident that the pair $e = \langle e_C, e_A \rangle$ is a morphism of precategories. It equates f with g: $f \circ e = g \circ e$, since the constituent maps e_C, e_A do it in **Set** for corresponding constituents of f and g. Now, for any other morphism $h: \mathbf{B} \to \mathbf{C}$ that equates f and g there is a unique pair $k = \langle k_C, k_A \rangle$ of maps such that $e \circ k = h$ in $\mathbf{Set}^2$. One checks that this k is indeed a morphism of precategories.

As a consequence we obtain the following lemma.

Lemma 2.3.6 $\mathbf{Set}^2$ *has equalizers and* U *creates them in* **preCat**. *Thus,* **preCat** *has equalizers.*

The two lemmas above imply in turn completeness of **preCat**.

Proposition 2.3.7 **preCat** *is (small) complete.*

Proof. Follows immediately from Lemma 2.3.5 and Lemma 2.3.6, by Corollary 2 p.109 in [Mac Lane 71]. □

It is well known that **Set**, and so $\mathbf{Set}^2$, is (small) cocomplete as well as (small) complete. Below we state without proof results dual to those just presented. Their proofs are 'dual' as well. The reader should be able to carry them out consulting [Goldblatt 79] for details of construction of coproducts and coequalizers in **Set**, if necessary.

Lemma 2.3.8 $\mathbf{Set}^2$ *has small coproducts, and* U *creates them in* **preCat**. *Thus,* **preCat** *has small coproducts.*

Lemma 2.3.9 $\mathbf{Set}^2$ *has coequalizers and* U *creates them in* **preCat**. *So,* **preCat** *has coequalizers.*

As before we conclude that **preCat** is (small) cocomplete.

Proposition 2.3.10 **preCat** *is (small) cocomplete.*

Proof. Follows immediately from Lemma 2.3.8 and Lemma 2.3.9, by result dual to Corollary 2 p.109 in [Mac Lane 71]. □

There is another important consequence of U creating all small limits and colimits in **preCat**. We can 'import' another nice property of **Set**.

Proposition 2.3.11 *Finite limits commute with filtered colimits in* **preCat**.

Proof. Filtered colimits commute with finite limits in **Set**, see Proposition 2.2.5. Therefore, they commute in $\mathbf{Set}^2$ as well. This is because both limits and colimits are computed in $\mathbf{Set}^2$ componentwise.

Since U creates, in particular, all finite limits and filtered colimits in **preCat** it follows from Fact 2.2.1(iv) that they commute in **preCat**. □

2.3.2 Categories of (Directed) Graphs with 'Partial' Morphisms: pGrph and ppGrph

We have introduced the category **preCat** of precategories mainly as a tool to study the properties of **pGrph** – the category of *directed graphs with 'partial' morphism*, and **ppGrph** – the category of *pointed* directed graphs with 'partial' morphisms that preserve the 'points'. We show that the first two categories are equivalent, and so that they enjoy the same categorical properties. Category **ppGrph** is obtained from **pGrph**, and inherits many properties of the latter.

It seems that the equivalence we are about to establish sheds a new light on the category **pGrph**. What is more important the natural forgetful functor, this time $U: \mathbf{pGrph} \to \mathbf{Set} \times \mathbf{Pfn}$, does not create all limits, and products in particular. This would leave us with a tedious direct method of showing that finite products commute with ω-colimits.

Let us proceed formally.

Definition 2.3.12 A *(directed) graph* G is a 4-tuple $G = \langle C_G, A_G, \partial^0_G, \partial^1_G \rangle$ where

C_G, is a set of *nodes*;

A_G, is a set of *edges*;

$\partial^0_G, \partial^1_G \colon A_G \to C_G$, are functions prescribing *beginning/source:* ∂^0, and *end/target:* ∂^1 to each edge;

Let G, H be graphs. A *morphism* $f \colon G \to H$ is a pair $f = \langle f_C, f_A \rangle$ where

$f_C \colon C_G \to C_H$ is a (total) function;

$f_A \colon A_S \rightharpoonup A_T$ is a partial function;

such that

(i) $f_A(a)\downarrow \Rightarrow f_C(\partial^i_G(a)) = \partial^i_H(f_A(a))$ for $i = 0, 1$ and $a \in A_G$, and
$f_A(a)\uparrow \Rightarrow f_C(\partial^0_G(a)) = f_C(\partial^1_G(a))$, for $a \in A_G$;

The pairs of identity functions form the *identity morphisms.*

Their *composition* is defined componentwise.

A *pointed directed graph* G is a 5-tuple $G = \langle C_G, c^\star_G, A_G, \partial^0_G, \partial^1_G \rangle$ where

$\langle C_G, A_G, \partial^0_G, \partial^1_G \rangle$, is the *underlying* directed graph of G;

$c^\star_G$, is an *initial node.*

A morphism of pointed directed graphs is a morphism of directed graphs that preserves initial nodes: $f_C(c^\star_G) = c^\star_H$.

In other words we consider the usual directed graphs with a larger collection of morphisms. Namely, we allow an edge to be removed via a morphism f, but only when this coincides with an identification of its beginning with its end via f.

In order to stress the difference between our generalisation and the usual notion of a morphism between directed graphs we shall sometimes refer to the former as 'partial'. The adjective 'directed' is often omitted as we do not consider other kinds of graphs.

First, note that we obtain categories.

Fact 2.3.13 *Graphs with their morphisms form a category, denoted* **pGrph**. *Pointed graphs with their morphisms form a category, denoted* **ppGrph**.

Proof. Omitted. □

Our next goal is to show that the categories: **preCat** and **pGrph** are equivalent.

Proposition 2.3.14 **pGrph** *is equivalent to* **preCat**.

Proof. We content ourselves with the definitions of appropriate functors. The burden of checking that they do enjoy the right property, i.e., one is the inverse of the other up to a natural iso, is put on the reader. We think it should be evident anyway.

Define PGRPH: **preCat** $\to$ **pGrph** as follows.

$\text{PGRPH}(\mathbf{C}) = \langle C_{\mathbf{C}}, \overline{A_{\mathbf{C}}}, \overline{\partial^0_{\mathbf{C}}}, \overline{\partial^1_{\mathbf{C}}} \rangle$ where

$$\overline{A_{\mathbf{C}}} = A_{\mathbf{C}} - \imath_{\mathbf{C}}(C_{\mathbf{C}}), \qquad \text{and}$$
$$\overline{\partial^i_{\mathbf{C}}}(a) = \partial^i_{\mathbf{C}}(a) \qquad \text{for } a \in \overline{A_{\mathbf{C}}}, i = 0,1.$$

$$\textsc{pGrph}(f\colon \mathbf{C} \to \mathbf{D}) = \langle f_C, \overline{f_A}\rangle\colon \textsc{pGrph}(\mathbf{C}) \to \textsc{pGrph}(\mathbf{D}) \quad \text{where}$$
$$\overline{f_A}(a) = f_A(a), \qquad \text{for } a \in \overline{A_{\mathbf{C}}}.$$

Similarly, define $\textsc{preCat}\colon \mathbf{pGrph} \to \mathbf{preCat}$ as follows.

$$\textsc{preCat}(G) = \langle C_G, \overline{A_G}, \overline{\partial^0_G}, \overline{\partial^1_G}, \imath_G\rangle \quad \text{where}$$
$$\overline{A_G} = A_G + C_G;$$
$$\overline{\partial^i_G}(a) = \begin{cases} \partial^i_G(a) & a \in A_G \\ c & a = c \in C_G \end{cases}, \qquad \text{for } a \in \overline{A_{\mathbf{C}}}, i = 0,1;$$
$$\imath_G(c) = c, \qquad \text{for } c \in C_G.$$

$$\textsc{preCat}(f\colon G \to H) = \langle f_C, \overline{f_A}\rangle\colon \textsc{preCat}(G) \to \textsc{preCat}(H) \quad \text{where}$$
$$\overline{f_A}(a) = \begin{cases} f_A(a) & a \in A_G \\ f_C(c) & a = c \in C_G \end{cases}, \qquad \text{for } a \in \overline{A_{\mathbf{C}}}.$$

□

We conclude immediately that both categories have the same categorical properties. In particular, we can repeat the results for **preCat** from the previous section.

Corollary 2.3.15 **pGrph** *is small complete and cocomplete. Finite limits commute with filtered colimits in* **pGrph**.

Proof. One readily sees that all the properties mentioned above are categorical. □

Later we shall see that the same is true of **ppGrph**.

For the future applications it is helpful to have more concrete characterisation of certain constructions in **pGrph**. The remaining part of this section is devoted to this problem, and to characterisation of corresponding constructs in **ppGrph**.

Products in pGrph and ppGrph

Let G, H be graphs. Define $G \sqcap H = \langle C_\sqcap, A_\sqcap, \partial^0_\sqcap, \partial^1_\sqcap\rangle$, where

$$C_\sqcap = C_G \times C_H;$$
$$A_\sqcap = A_G \times C_H + A_G \times A_H + C_G \times A_H;$$
$$\partial^i_\sqcap(a,b) = \begin{cases} \langle \partial^i_G(a), d\rangle & a \in A_G, b = d \in C_H \\ \langle \partial^i_G(a), \partial^i_H(b)\rangle & a \in A_G, b \in A_H \\ \langle c, \partial^i_H(b)\rangle & a = c \in C_G, b \in A_H \end{cases} \qquad \text{for } i = 0,1.$$

Next, we define *projections*: $\pi_G\colon G \sqcap H \to G$, $\pi_H\colon G \sqcap H \to H$. To this end put $\pi_G = \langle \pi^G_C, \pi^G_A\rangle$, where

$\pi^G_C\colon C_\sqcap \to C_G$, the usual set-theoretic projection;

$\pi^G_A\colon A_\sqcap \to A_G$, is defined by:

$\pi^G_A(\langle a,b\rangle) = a$ if $a \in A_G$

Similarly in case of π_H.

One can see that the construction above is the result of translation of the (binary) product construction in **preCat** via equivalence. We chose to give a sketch of an explicit proof instead.

Fact 2.3.16 *$G \sqcap H$ together with projections: π_G and π_H, is a categorical product of G and H in* **pGrph**.

Proof. First note that $G \sqcap H$ is a graph, i.e., that the maps $\partial^0_\sqcap, \partial^1_\sqcap$ are total functions.

Our next observation is that the pair π_G, and similarly π_H, gives a morphism in **pGrph**. For if $\pi_G(a_\sqcap)\uparrow$ then $a_\sqcap = \langle c, b\rangle$, for some $c \in C_G, b \in A_H$. Thus,

$$\pi^G_C(\partial^i_\sqcap(a_\sqcap)) = \langle c, \partial^i_H(b)\rangle = c, \quad \text{for } i = 0,1, \text{ as required.}$$

If, on the other hand, $\pi_G(a_\sqcap) = a \in A_G$ then by the definitions of π^G_C, π^G_A and $\partial^i_\sqcap$ we have:

$$\pi^G_C(\partial^i_\sqcap(a_\sqcap)) = \partial^i_G(a) = \partial^i_G(\pi^G_A(a_\sqcap)), \quad \text{for } i = 0,1, \text{ again as required.}$$

Finally, we show that $G \sqcap H$ enjoys the right universal property. To this end assume a graph $H_0 = \langle C_0, A_0, \partial^0_0, \partial^1_0\rangle$ is given together with morphisms: $g\colon H_0 \to G$ and $h\colon H_0 \to H$. First, we show the uniqueness of a hypothetical mediating morphism $f\colon H_0 \to G \sqcap H$. For suppose f does satisfy

$$\pi_G \circ f = g \quad \text{and} \quad \pi_H \circ f = h.$$

We show that f must be defined in terms of g and h in a unique way.

It is immediate that $f_C = g_C \times h_C$, since $C_\sqcap$ with π^G_C, π^H_C is a categorical product of C_G and C_H in **Set**.

As for the f_A we need to consider four cases. Let $a \in A_0$.

If $g_A(a)\downarrow$ and $h_A(a)\downarrow$ then $\pi^G_C(f_A(a)) = g_A(a)$ and $\pi^H_C(f_A(a)) = h_A(a)$, so $f_A(a) = \langle g_A(a), h_A(a)\rangle$.

If $g_A(a)\downarrow$ and $h_A(a)\uparrow$ then $\pi^G_C(f_A(a)) = g_A(a)$ and $\pi^H_C(f_A(a))\uparrow$, so $f_A(a) = \langle g_A(a), d\rangle$, for some $d \in C_H$. But since f is a morphism it follows that $d = h_C(\partial^i_H(a))$, as the reader may easily check.

Case $g_A(a)\uparrow$ and $h_A(a)\downarrow$ is symmetric.

Finally, if $g_A(a)\uparrow$ and $h_A(a)\uparrow$ then $f_A(a)$ must be undefined as well.

In the light of the above discussion to show the existence of a mediating morphism one needs to show that the pair $f = \langle f_C, f_A\rangle$, where $f_C = g_C \times h_C$ while f_A is defined below, constitutes a morphism of graphs. This task is left to the reader.

$$f_A(a) = \begin{cases} \langle g_A(a), h_A(a)\rangle & g_A(a)\downarrow \wedge\, h_A(a)\downarrow \\ \langle g_A(a), h_C(\partial^0_H(a))\rangle & g_A(a)\downarrow \wedge\, h_A(a)\uparrow \\ \langle g_C(\partial^0_G(a)), h_A(a)\rangle & g_A(a)\uparrow \wedge\, h_A(a)\downarrow \end{cases}, \qquad \text{for } a \in A_0.$$

Note that by convention $f_A(a)$ is undefined in the remaining case: $g_A(a)\uparrow \wedge h_A(a)\uparrow$.

□

Remark. The reader may notice that products in **pGrph** are not created via forgetful $U: \mathbf{pGrph} \to \mathbf{Set} \times \mathbf{Pfn}$. Thus, to prove directly Corollary 2.3.15, and especially commutativity of finite limits with filtered colimits, one would have to conduct the proofs explicitly.

Now, let us take pointed graphs G, H. Small adjustments are needed to get their product from the product of their underlying graphs. We put

$$G \amalg H = \langle C_\Pi, c^\star_\Pi, A_\Pi, \partial^0_\Pi, \partial^1_\Pi \rangle,$$

where $c^\star_\Pi = \langle c^\star_G, c^\star_H \rangle$, and the rest is defined as previously.

In this way we construct the categorical product in **ppGrph** since the projections evidently preserve the initial nodes. One can be even more specific and say that the forgetful functor $U: \mathbf{ppGrph} \to \mathbf{pGrph}$ that sends a pointed graph to its underlying graph creates binary products. The generalisation to arbitrary (small) products is obvious, and we state it without proof.

Fact 2.3.17 *The forgetful functor* $U: \mathbf{ppGrph} \to \mathbf{pGrph}$ *creates (small) products.*

Terminal Objects in pGrph and ppGrph

A particular consequence of Fact 2.3.17 is that U creates terminal objects. The reader may wish to convince him/herself that $\mathbf{1}_{\mathbf{pGrph}} = \langle \mathbf{1}_{\mathbf{Set}}, \emptyset, \emptyset, \emptyset \rangle$, where $\mathbf{1}_{\mathbf{Set}} = \{\star\}$ is a set with one element, recall that $\mathbf{1}_{\mathbf{Set}}$ is a terminal object in **Set**, is a terminal object in **pGrph**. The corresponding $\mathbf{1}_{\mathbf{ppGrph}} = \langle \mathbf{1}_{\mathbf{Set}}, \star, \emptyset, \emptyset, \emptyset \rangle$ is not only terminal in **ppGrph**, but initial as well, i.e., a zero object.

Equalizers in pGrph and ppGrph

We know that **pGrph** has equalizers, since it is equivalent to a small complete category **preCat**. We state without proof the following result.

Fact 2.3.18 *The forgetful functor* $U: \mathbf{ppGrph} \to \mathbf{pGrph}$ *creates equalizers.*

pGrph and ppGrph are Small Complete

Completeness of **pGrph** was already established as a consequence of equivalence between **pGrph** and **preCat**. As for the **ppGrph** we obtain it by general means from products and equalizers.

Proposition 2.3.19 **ppGrph** *is (small) complete.*

Proof. Follows immediately from Lemma 2.3.17 and Lemma 2.3.18, by Corollary 2 p.109 in [Mac Lane 71]. □

Binary Coproducts in pGrph and ppGrph

Let G, H be graphs. Take graph $G \amalg H = \langle C_\amalg, A_\amalg, \partial^0_\amalg, \partial^1_\amalg \rangle$ given by:

$$C_\amalg = C_G + C_H;$$

$$A_\amalg = A_G + A_H;$$

$$\partial^i_\amalg = \partial^i_G + \partial^i_H, \quad \text{for } i = 0, 1.$$

Since disjoint union gives coproduct in both: **Set** and **Pfn**, it follows that '+' is a bifunctor. Hence, the definition of $\partial^i_\amalg$, for $i = 0, 1$.

The *injection morphisms* $in_G: G \to G \amalg H, in_H: H \to G \amalg H$ have appropriate injections as components.

By now it should be standard to check that $G \amalg H$ with injections is a categorical coproduct of G and H in **pGrph**. To prove it one can check, for instance, that the above is (isomorphic to) what we would get by translating a coproduct translation from **preCat**.

It is more fruitful, however, to realise that the construction presented proves more. It shows that the forgetful functor $\mathrm{U}: \mathbf{pGrph} \to \mathbf{Set} \times \mathbf{Pfn}$ creates binary coproducts.

This remark is useful when it comes to consider coproducts in **ppGrph**. This time it is not the case that they are created by $\mathrm{U}: \mathbf{ppGrph} \to \mathbf{pGrph}$. But one readily sees that they are created via forgetful functor $\mathrm{U}: \mathbf{ppGrph} \to \mathbf{Set}_\star \times \mathbf{Pfn}$. For pointed G, H their coproduct $G \amalg H$ is given by

$$C_\amalg = (C_G + C_H)/\approx, \quad \text{where } \approx \text{ identifies } c^\star_G \text{ with } c^\star_H;$$

$$c^\star_\amalg = [c^\star_G]_\approx;$$

$$A_\amalg = A_G + A_H;$$

$$\partial^i_\amalg = \partial^i_G + \partial^i_H, \quad \text{for } i = 0, 1.$$

Obviously, $\langle C_\amalg, c^\star_\amalg \rangle$ is a coproduct, 'amalgamated sum', of $\langle C_G, c^\star_G \rangle$ and $\langle C_H, c^\star_H \rangle$ in $\mathbf{Set}_\star$. Using corresponding injections we can define $\partial^i_\amalg$, for $i = 0, 1$, as previously.

This is an easy step to generalise the above to arbitrary (small) coproducts. We leave the task of filling in the details of the proof of the following fact to the reader.

Fact 2.3.20 *The forgetful functor* $\mathrm{U}: \mathbf{ppGrph} \to \mathbf{Set}_\star \times \mathbf{Pfn}$ *creates (small) coproducts. Thus,* **ppGrph** *has (small) coproducts.*

Initial Objects in pGrph and ppGrph

We can conclude from the Fact 2.3.20 that U creates initial objects. It is easy to check that $\mathbf{0}_{\mathbf{pGrph}} = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$, is an initial object in **pGrph**. Earlier we noticed that $\mathbf{1}_{\mathbf{ppGrph}} = \langle \mathbf{1}_{\mathbf{Set}}, \star, \emptyset, \emptyset, \emptyset \rangle$ is a zero object in **ppGrph**.

Coequalizers in pGrph and ppGrph

pGrph has coequalizers, since it is equivalent to a small cocomplete category **preCat**. We state without proof the following result.

Fact 2.3.21 *The forgetful functor* U: **ppGrph** → **pGrph** *creates coequalizers.*

pGrph and ppGrph are Small Cocomplete

Cocompleteness of **pGrph** was already established, see Corollary 2.3.15. For **ppGrph** it follows by general means from the existence of coproducts and coequalizers.

Proposition 2.3.22 **ppGrph** *is (small) cocomplete.*

Proof. Follows immediately from Lemma 2.3.20 and Lemma 2.3.18, by dual to Corollary 2 p.109 in [Mac Lane 71]. □

Finite Products Commute with ω-colimits in ppGrph

We end this Chapter by showing that in **ppGrph** finite products commute with ω-colimits. Presumably, one could aim at a stronger result. Namely, that finite limits commute with filtered colimits in **ppGrph**.

The result will follow by an application of Fact 2.2.1(iv). Therefore, we need to know that the forgetful functor U: **ppGrph** → **pGrph** creates ω-colimits. Unfortunately, it does not create all colimits, vide coproducts.

Let $\Gamma = G_0 \xrightarrow{f_0} G_1 \xrightarrow{f_1} \dots$ be a ω-diagram in **pGrph**. Then a colimiting cocone $\phi\colon \Gamma \to G_\rightarrow$ may be constructed as follows.

Assume $G_n = \langle C_n, A_n, \partial^0_n, \partial^1_n \rangle$, and $f_n = \langle f^n_C, f^n_A \rangle$, for $n \in \omega$. Put

$$G_\rightarrow = \langle C_\rightarrow, A_\rightarrow, \partial^0_\rightarrow, \partial^1_\rightarrow \rangle \quad \text{where}$$

$C_\rightarrow = (\coprod_{n\in\omega} C_n)/\approx_C$ where $\approx_C$ is an equivalence relation generated by the set $\{\langle c, f^n_C(c)\rangle \mid n \in \omega, c \in C_n\}$;

$A_\rightarrow = (\coprod_{n\in\omega} A_n)/\approx_A$ where $\approx_A$ is a partial equivalence relation given by:

$$a \in \mathrm{dom}(\approx_A) \quad \text{iff} \quad f^{m+n}_A \circ \dots \circ f^n_A(a)\downarrow \quad \text{for } m \in \omega, a \in A_n;$$

$$a \approx_A b \quad \text{iff} \quad a, b \in \mathrm{dom}(\approx_A) \wedge f^k_A \circ \dots \circ f^m_A(a) = f^k_A \circ \dots \circ f^n_A(b),$$
$$\text{for } m, n \in \omega, a \in A_m, b \in A_n, \text{ for some } k \geq m, n;$$

$$\partial^i_\rightarrow([a]_{\approx_A}) = [\partial^i_n(a)]_{\approx_C} \qquad \text{for } a \in A_n \cap \mathrm{dom}(\approx_A), i = 0, 1.$$

And let $\phi_n\colon G_n \to G_\rightarrow$ be defined by:

$$\phi^n_C(c) = [c]_{\approx_C}, \qquad \text{for } c \in C_n;$$

$$\phi^n_A(a) = [a]_{\approx_A}, \qquad \text{for } a \in A_n.$$

Note that $\phi^n_A(a)$ is undefined iff $a \notin \mathrm{dom}(\approx_A)$.

Fact 2.3.23 $\phi\colon \Gamma \to G_\rightarrow$ *is a colimiting cocone. It is created via forgetful functor* U: **pGrph** → **Set** × **Pfn**.

Proof. Elementary. All the data necessary to conduct it is available. The reader should be able to fill in the details. □

A quick inspection reveals that if we start with Γ — a ω-diagram in **ppGrph** then only cosmetic changes are necessary in order to get a colimiting cocone in **ppGrph** from the above construction. For let $G_{\rightarrow} = \langle C_{\rightarrow}, c^*_{\rightarrow}, A_{\rightarrow}, \partial^0_{\rightarrow}, \partial^1_{\rightarrow} \rangle$, where the only new item is defined by: $c^*_{\rightarrow} = [c^*_0]_{\approx_C}$.

Then indeed the maps ϕ_n preserve the initial nodes, as $c^*_0 \approx_C c^*_n$, for $n \in \omega$. Therefore, $\phi: \Gamma \rightarrow G_{\rightarrow}$ is colimiting. Form the above discussion it is evident that the following fact holds. The details of its proof are left to the reader.

Fact 2.3.24 *The forgetful functor* U: **ppGrph** $\rightarrow$ **pGrph** *creates* ω*-colimits.*

Thus, finally, we obtain

Corollary 2.3.25 *Binary products commute with* ω*-colimits in* **ppGrph**.

Proof. Both binary products and ω-colimits are created in **ppGrph** via forgetful U: **ppGrph** $\rightarrow$ **pGrph**, by Fact 2.3.17 and Fact 2.3.24, respectively. They exist and commute in **pGrph** as a consequence of Corollary 2.3.15. Thus, they exist and commute in **ppGrph**, by Fact 2.2.1(iv). □

Part I

Generalised Transition Systems

Chapter 3

Petri Nets, Petri Systems

Petri nets are a concrete kind of concurrent systems — the *basic* model for asynchrony in our opinion. They are perhaps best thought of as a generalisation of Keller's *transition systems* in which the states were distributed in order to give a more elementary account of the control structure. This was achieved by introduction of the notion of *conditions*, or *places*.

The conditions result in a natural graphical representation of Petri nets. Obviously, the ability to *visualise* concurrent systems and their behaviour is crucial for any model for concurrency.

At the same time the same conditions become a handicap when nets grow large. Even more so when the problem of constructing big systems from small ones is considered. The common factor is abundance of the elements that form the description of systems. To avoid this complexity a variety of more abstract models was developed.

The same factor appears also in another form. What we are mostly interested in is *dynamic behaviour* of systems. The problem with the formalism of Petri nets is that it provides many descriptions of systems with intuitively the same behaviour. This prompts the question of a choice of a (universal) representative of the class of nets with the same behaviour.

We would like to see *Petri systems* defined here as a crude yet natural first approximation of the notion of dynamic behaviour. Then the above discussion can be encapsulated in the form of a more general question.

> Does there exist a (universal) procedure that given a behaviour, i.e., a Petri system, produces its *realisation*, that is a Petri net that gives rise to this Petri system?

It is shown that the answer to this question is negative. But, it seems, any notion of behaviour that does away with places must be seen as a form of collapsing of (the category of) Petri systems. This strongly suggests that the problem of a uniform realisation of abstract models in the general Petri nets is insoluble in general as well.

This argument is taken as an excuse for rejection of both models as too general from a practical point of view. It seems essential that one is able to go back and forth between abstract descriptions of concurrent systems, and their concrete realisations in Petri nets.

The Chapter is organised as follows.

The first section introduces the definition of Petri nets. Their graphical representation, notation and some standard notions are introduced. Finally, the category **PNet** of Petri nets is defined. The notion of morphism is taken from Winskel, see [Winskel 84].

In the second section the category **PSys** of Petri systems is introduced. It is shown that every Petri net N gives rise to a Petri system SYS N constructed by forgetting about concrete details of presentation of the nets. Specifically, one forgets the way in which conditions build markings. This process of abstraction extends to a functor SYS: **PNet** $\rightarrow$ **PSys**.

Finally, in the last section we show that neither of the categories can be thought of as a (co)reflective subcategory of the other with SYS as one of the functors of the (co)reflection. We take this as a formalisation of non-existence of a uniform realisation of Petri systems by means of Petri nets.

3.1 Petri Nets: the Concrete Model of Asynchronous Computations

Net theory was initiated by C. A. Petri in 1962, see [Petri 62], and has since developed into a large subject with numerous applications. The reader should be aware that the author concentrates on a specific branch of the theory that deals with application of nets to modelling of dynamically changing systems. Other branches, like the structure theory of nets, are left untouched. For a more representative account on the theory of Petri nets the reader is asked to consult [APN'86/I].

It may be historically wrong, but perhaps didactically correct to start the presentation of Petri nets by comparison with a version of transition systems introduced by Keller, see [Keller 76].

3.1.1 Transition Systems

Both models intend to formalise *dynamic systems*. Originally, a *transition system* S, abbr.: ts, was just a pair $S = \langle C, \rightarrow \rangle$ where $c, d : C$ was a non-empty set of *states*, or *configurations*, and $_ \rightarrow _ \subseteq C \times C$ a *transition relation*. Intuitively, $c \rightarrow d$ captures the ability of S to *evolve* from the state c to the state d.

We shall always designate a particular state $c^* \in C$ as *initial*. Then the pair becomes a triple $\langle C, c^*, \rightarrow \rangle$.

To capture the idea that each transition of a state has some *cause* the triple is expanded to a 4-tuple $\langle C, c^*, A, \rightarrow \rangle$, by adding a set $a : A$ of *actions*. The transition relation changes its type to $_ \stackrel{_}{\rightarrow} _ \subseteq C \times A \times C$. Accordingly, $c \stackrel{a}{\rightarrow} d$ means that the system S from the state c can evolve to the state d with occurrence of the action a being the cause of the transition.

So far it is very easy to model systems that exhibit a *non-deterministic* behaviour. For instance, if for an action a, and some states c, d, d' we have $c \stackrel{a}{\rightarrow} d$ and $c \stackrel{a}{\rightarrow} d'$ then one can think that the result of action a at c is indeterminate, if $d \neq d'$. For reasons that will become apparent later we want to consider transition systems in which the above cannot

take place, i.e., in the situation described above they will always satisfy $d = d'$. Additionally, we demand that each action a can be seen as *happening* somewhere in the system, i.e., that $c \xrightarrow{a} d$ hold, for some $c, d \in C$.

The 4-tuples as above were originally termed *labelled transition systems*. With the extra provisos they could be called *deterministic labelled ts-s*. Nevertheless, we will call them, simply, transition systems. In this convention we follow others, e.g., [Winskel 84b].

We do not want to loose the ability to model systems that exhibit non-deterministic behaviour. The solution proposed is to say that certain actions exhibit the same attributes to the outside world, and hence are indistinguishable from each other. This can be formalised by making S into a 6-tuple $S = \langle C, c^*, A, \rightarrow, M, l \rangle$ where M is a set of *labels*, i.e., externally observable attributes of actions, and $l: A \rightharpoonup M$ is a partial *labelling function*. Intuitively, $l(a)$ is the protrusion of the action a to the external world. We would like to think of those actions that do not have labels associated with them as *internal* or *unobservable*.

In the already mentioned paper [Winskel 84b] Winskel studies the category of labelled transition systems more thoroughly, although he allows only total labellings. We feel free to take it as a basic intuitive and informal model, and refer to its categorical properties only when necessary. Let us remark that with morphisms as defined below the category of transition systems becomes equivalent to the category **ppGrph** studied in Chapter 2.

As morphisms of transition systems we take pairs of functions that preserve their structure. Thus, $f = \langle f_C, f_A \rangle: S_0 \rightarrow S_1$ where $S_0 = \langle C_0, c_0^*, A_0, \rightarrow \rangle$, and $S_1 = \langle C_1, c_1^*, A_1, \rightarrow \rangle$ comprises: $f_C: \langle C_0, c_0^* \rangle \rightarrow \langle C_1, c_1^* \rangle$ and $f_A: A_0 \rightharpoonup A_1$. Thus, we demand that f_C be a morphism in $\mathbf{Set}_*$, i.e., a (total) function that preserves the initial states, and allow f_A to be a partial function. Moreover, they should somehow preserve the transition relation. Intuitively, morphisms f should reflect the evolution possibilities of S_0 in S_1. Assume that $c \xrightarrow{a} d$ holds in S_0. If $f_A(a)\downarrow$ then we simply demand that $f_C(c) \xrightarrow{f_A(a)} f_C(d)$ in S_1. If $f_A(a)\uparrow$ then we can think of action a as being *wiped out* by f, and so the result of it happening should be undone. Therefore we stipulate $f_C(c) = f_C(d)$ in this case.

So far there was no place for modelling concurrency in the above set-up. The idea is to abandon the image of states and actions as solid entities, first class objects. Petri says that we should view them as second class citizens that have internal structure, and are actually definable from more elementary items called *conditions* or *places*, and *events* or *transitions*, respectively. The labelling part is left aside for a while as it plays no role in the discussion.

Every event has a specified *collection* of conditions on which it *depends*, and also another collection of conditions which *depend on it*. The transition relation is definable from this data as well. em States are collections of conditions. Then for an event to be *enabled* in a given state the collection of the conditions it depends on must be a sub-collection of the state. When an enabled event *occurs* it affects the state by removing from it the collection of places on which it depended, and adding to it the collection of places which depended on it.

There are many ways in which the ideas described above can be formalised. And indeed

many, sometimes intriguing formalisations can be found in the literature. Many of the definitions available are equivalent. Our intention is to chose something simple for a presentation, and also familiar from mathematical point of view.

Some examples can be found in a monograph by Reisig, [Reisig 85], and the reader interested in the theory of Petri nets should look there for more on the subject. There is also a series of volumes of Advances in Petri Nets in the Springer's series of Lecture Notes in Computer Science. They are devoted not only to Petri nets but also to related topics, and appear annually. The recently published volumes [APN'86/I] and [APN'86/II] are a good reference for anybody interested in the subject.

3.1.2 Petri Nets — a Definition

Here is a formal definition of Petri nets. The definitions of (finitary) multisets, abbr.: μ-sets, and multifunctions, abbr.: μ-functions, are given in the Chapter 2.

Definition 3.1.1 A *Petri net* N is a 5-tuple $N = \langle B, E, {}^\bullet(_), (_)^\bullet, M_0 \rangle$ where

$b{:}\,B$ is a non-empty set of *places*, or *conditions*,

$e{:}\,E$ is a set of *events*, or *transitions*,

${}^\bullet(_), (_)^\bullet{:}\, E \to \mu B$ are *pre-* and *postcondition functions*,

$M_0 \in \mu B$ is called *initial marking*.

Items building a net will be denoted by their generic names, e.g., B for the set of places of N, with the same superscripts as the net in question. Hence, E' will denote the set of events of N'. This does not apply to the functions ${}^\bullet(_)$ and $(_)^\bullet$. It is hoped that the relevant information can be always recovered from the context.

Thus, our definition is quite liberal, more general than Winskel's in [Winskel 87] as we do not impose any restrictions. We do, however, admit only finitary μ-sets, while Winskel allows for the non-finitary μ-sets as well. This decision was taken to make the presentation simpler. For instance, finitary μ-sets, and μ-functions between them form a category. This is not so in general, for examples see [Winskel 87].

In [Winskel 87] the author also shows that his definition has an equivalent more standard presentation of Petri nets that employs the notion of so called *flow relation*. His argument is valid also in our case, and we feel relieved from providing a similar result.

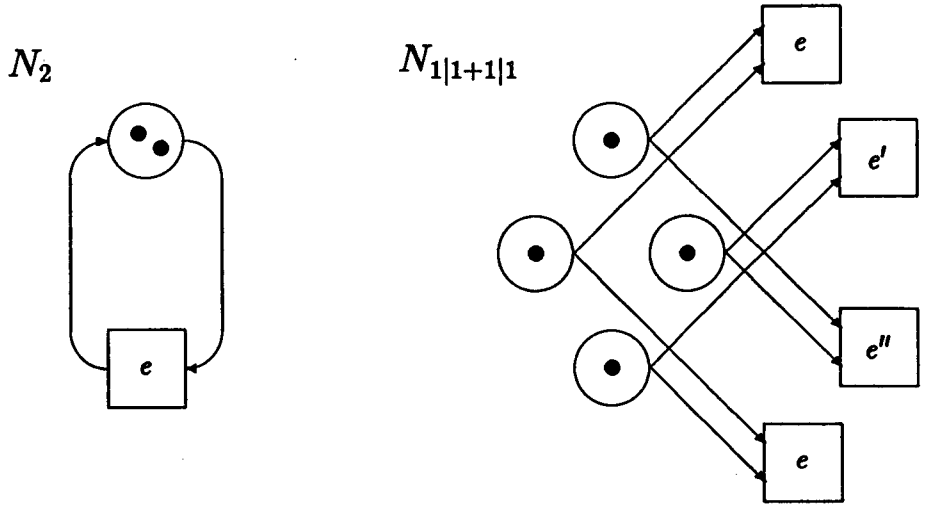
There is a still more general class of Petri nets known as *predicate/transition systems*, abbr.: P/T systems, see [APN'86/I]. Nets from this class have *capacities* associated with each place. Our nets can be viewed as P/T systems with each place having infinite capacity. We are not interested in considering this generalisation. On the contrary, we shall argue that the category **PNet**, to be defined, is already too large from the "practical" point of view. Categorically, it is not cocomplete enough.

Petri nets are admired for their intuitive graphical representation. In accord with the usual convention we shall draw conditions as circles, and transitions as boxes. The μ-set ${}^\bullet(e)$ of conditions on which a transition e depends is represented as a set of arrows leading

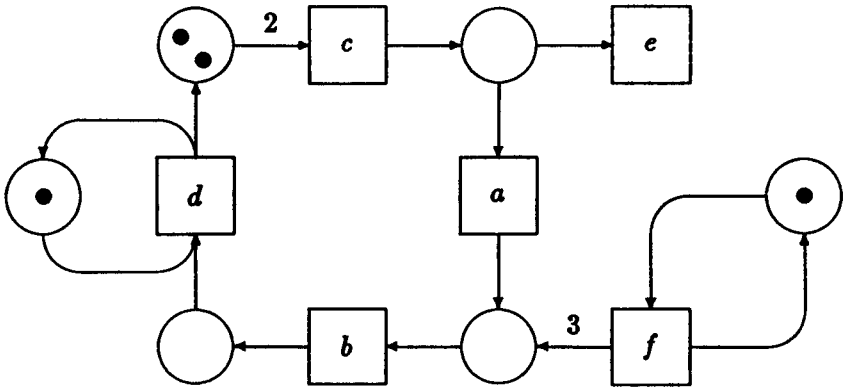
from each condition b s.t., $\widehat{b} \leq {}^\bullet(e)$, to e. Each arrow from b has a *weight* associated with it and equal: ${}^\bullet(e)b$. The μ-sets $(e)^\bullet$, for $e \in E$ are represented analogously.

The initial marking is represented as a set of blobs in conditions, the number of blobs in a condition b equal $M_0 b$, for $b \in B$.

We give several examples of nets. Arrows without a weight associated with them are assumed to have the weight equal 1.



And now somewhat more elaborate example.



Dynamic Behaviour of Petri Nets

States of Petri nets are represented as *markings*, i.e., finitary μ-sets of conditions. Petri Net Theory comes with an illustrative terminology and intuitive descriptions. For instance we are urged to think about places as a kind of stores. Markings represent then some kind of resources, pictorially, the blobs. A *firing* transition consumes resources from one places, and produces them in the other — the blobs disappear from some circles only to reappear in the other. The amount of resources consumed and produced are described by the functions

${}^\bullet(_)$, and $(_)^\bullet$. It is only natural, but not everybody does it, to allow an event to occur with some multiplicity as long as there are enough resources around. This play is known as *token game.*

Let us now introduce some notation to capture the above phenomena. Let $N = \langle B, E, {}^\bullet(_), (_)^\bullet, M_0\rangle$ be a net, and $M, M' \in \mu B$ be markings, i.e., finitary μ-sets of conditions, while $A \in \mu E$ a finitary μ-set of events in N.

A is called *enabled* at M, and denoted $M \xrightarrow{A}$, iff $\mu^\bullet(A) \leq M$. Here, $\mu^\bullet(_)$ denotes the extension of ${}^\bullet(_)$ to a μ-function, as defined in Chapter 2.

The *transition relation* $_ \xrightarrow{} _ \subset \mu B \times \mu E \times \mu B$ is defined by:

$$M \xrightarrow{A} M' \quad \text{iff} \quad \mu^\bullet(A) \leq M \wedge M' = M - \mu^\bullet(A) + \mu(A)^\bullet.$$

In effect, we have recovered a transition system $\langle \mu B, M_0, \mu E, \rightarrow\rangle$ underlying every Petri net.

Set $\mathrm{Mark}_N \subseteq \mu B$ of *reachable markings* of a net N is defined as the least one that satisfies:

(i) $M_0 \in \mathrm{Mark}_N$, and

(ii) $M \xrightarrow{A} M' \wedge M \in \mathrm{Mark}_N$ implies $M' \in \mathrm{Mark}_N$, for $M, M' \in \mu B$, $A \in \mu E$.

Evidently, a marking is reachable iff it can be arrived at while playing the token game from the initial marking.

3.1.3 A Category of Petri Nets

Morphisms of Petri Nets

It was W. Reisig who suggested that perhaps Petri nets can be best viewed as certain algebras.

Every net N determines a structure $\mu N = \langle \mu B, \mu E, \mu^\bullet(_), \mu(_)^\bullet, M_0\rangle$. From the algebraic point of view the latter is nothing other than a certain *2-sorted algebra* in the category $\mu - \mathbf{Set}$ of μ-sets. One can say that Petri nets are presentations of such algebras.

This analogy becomes even more apparent when we realise that a notion of morphism of Petri nets is also a presentation of a morphism, i.e., a homomorphism of such algebras.

Definition 3.1.2 (Winskel) A *morphism* $f: N \rightarrow N'$ of Petri nets is a pair $f = \langle f_B, f_E\rangle$ where

$$f_B: B \rightarrow \mu B' \quad \text{and} \quad f_E: E \rightarrow E',$$

are such that:

(i)

$$\begin{array}{ccc} \mu E & \xrightarrow{\mu f_E} & \mu E' \\ {\scriptstyle \mu^\bullet(_)}\downarrow & & \downarrow{\scriptstyle \mu^\bullet(_)} \\ \mu B & \xrightarrow[\mu f_B]{} & \mu B' \end{array} \qquad \begin{array}{ccc} \mu E & \xrightarrow{\mu f_E} & \mu E' \\ {\scriptstyle \mu(_)^\bullet}\downarrow & & \downarrow{\scriptstyle \mu(_)^\bullet} \\ \mu B & \xrightarrow[\mu f_B]{} & \mu B' \end{array}$$

(ii) $\mu f_B M_0 = M_0'$.

The *composition* $g \circ f$ of morphisms $f: N \to N'$ and $g: N' \to N''$ is defined by: $g \circ f = \langle \mu g_B \circ f_B, g_E \circ f_E \rangle$. The *identity* morphism id_N is equal $id_N = \langle \hat{}, id_E \rangle$.

Obviously, $f: N \to N'$ is a morphism of nets iff the pair $\mu f = \langle \mu f_B, \mu f_E \rangle$ is a homomorphism of the corresponding algebras. As a consequence we see that Petri nets with above morphisms form a category.

Fact 3.1.3 *Petri nets with their morphisms form a category, denoted* **PNet**.

Proof. Omitted. □

Morphisms as defined above preserve the transition relation.

Proposition 3.1.4 *Let $f: N \to N'$ be a morphism of nets. Then for $M, M' \in \mu B$, $A \in \mu E$ we have:*

$$M \xrightarrow{A} M' \quad \textit{implies} \quad \mu f_B M \xrightarrow{\mu f_E A} \mu f_B M'.$$

Proof. Let us assume that $M \xrightarrow{A} M'$ in N.

Then $M \geq \mu^\bullet(A)$ implies $\mu f_B M \geq \mu f_B \mu^\bullet(A) = \mu^\bullet(\mu f_E A)$.

Therefore,

$$\begin{aligned} \mu f_B M' &= \mu f_B M - \mu f_B \mu^\bullet(A) + \mu f_B \mu(A)^\bullet, && \text{by linearity of } f_B \\ &= \mu f_B M - \mu^\bullet(\mu f_E A) + \mu(\mu f_E A)^\bullet, && \text{by 3.1.2(i).} \end{aligned}$$

as required. □

One of the consequences of the above Proposition is that reachable markings are mapped onto reachable markings.

Corollary 3.1.5 *If $f: N \to N'$ is a morphism of Petri nets then $\mu f_B(\mathrm{Mark}_N) \subseteq \mathrm{Mark}_{N'}$.*

Proof. Let $\mathcal{M}$ be a set of all markings that are mapped onto $\mathrm{Mark}_{N'}$. Then $\mathcal{M}$ satisfies both conditions that define Mark_N: the first by definition, the second by Proposition 3.1.4. The result follows. □

Let us finally remark that the procedure of sending a net N to a transition system $\mu N = \langle \mu B, M_0, \mu E, \to \rangle$, and a net morphism f to the morphism $\mu f = \langle \mu f_B, \mu f_E \rangle$ of the corresponding ts-s is a functor. As a consequence an isomorphism $f: N \to N'$ of nets must satisfy:

(i) f_E is (total) bijection;

(ii) $f_B = \hat{} \circ g$, for some bijection $g: B \to B'$.

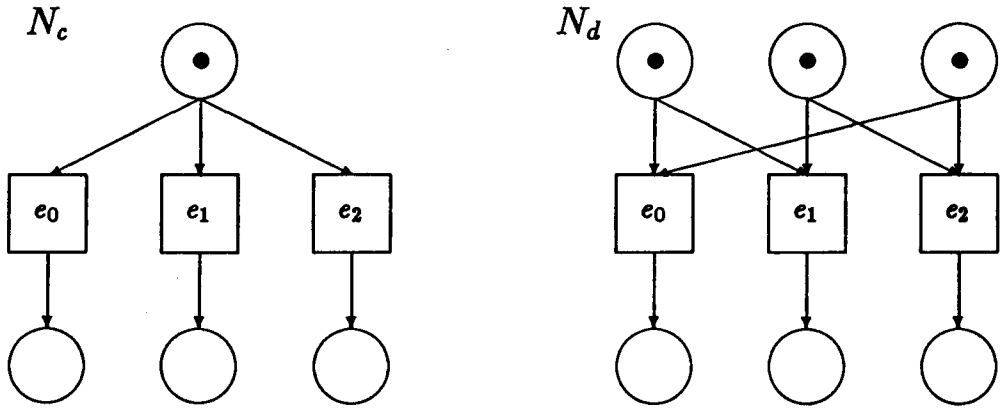
One can verify that these two conditions are sufficient as well.

3.2 Petri Systems

The analogy between blobs at a place and some kind of resources is not the one that we would like to pursue very hard. It might be all right for some applications of Petri nets in which, say, places are to model some real life stores, the blobs stand for physical goods, and events represent their transformation in factories. For us, though, places and the blobs in them are merely means of control with which one achieves required behaviour of the system. Therefore the exact method in which a particular behaviour is achieved is a secondary issue.

Rather than spend more time arguing let us consider an example.

Example 3.2.1 Let N_c and N_d be the following Petri nets:



Intuitively, there is no difference in their dynamic behaviour: both can fire exactly one among events: e_0, e_1 and e_2, and afterwards do nothing. The only difference is in the way it is achieved. In N_c there is only one place at which the conflict between the three events is resolved. In N_d the resolution of the conflict is distributed.

The reader should note that the nets represent a *sequential* behaviour, i.e., no two events can ever happen simultaneously, and no event can ever happen with a multiplicity greater than one. ◇

We take the elements of what was referred to as *dynamic behaviour* of nets to build an approximation of the notion *behaviour* of concurrent systems.

Definition 3.2.2 A *Petri system* S is a 4-tuple $S = \langle C, A, \rightarrow, c^* \rangle$ where:

$c, d: C$ is a set of *configurations* or *states*,

$c^* \in C$ is an *initial configuration*,

$a, b: A$ is a set of *elementary actions*.

$_ \stackrel{\cdot}{\rightarrow} _ \subset C \times \mu A \times C$ is a *transition relation*,

and such that for $c, d, d' \in C$, $a \in A$, $\alpha, \beta \in \mu A$ the following are satisfied:

(i) $c \xrightarrow{\emptyset} c$,

(ii) $\overline{c} \xrightarrow{a} \overline{d}$, for some $\overline{c}, \overline{d} \in C$,

(iii) $c \xrightarrow{\alpha} d \wedge c \xrightarrow{\alpha} d'$ implies $d = d'$,

(iv) $c \xrightarrow{\alpha+\beta} d$ implies $c \xrightarrow{\alpha} c' \xrightarrow{\beta} d$, for some $c' \in C$.

Evidently, the 4-tuple $\langle C, c^*, \mu A, \rightarrow \rangle$ is a transition system.

The usual conventions are used to refer to items building a Petri system, e.g., C_S denotes the set of configurations of S, etc. The only exception is the transition relation. We do not stick any labels to it. The corresponding transition system should be always clear from the context.

We shall also consider an extension of the transition relation onto $C \times (\mu A)^* \times C$ by allowing $c \xrightarrow{v} d$ for some $c, d \in C$, $v = \alpha_1 \ldots \alpha_n \in (\mu A)^*$ iff $c_0 \xrightarrow{\alpha_1} c_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} c_n$, for some $c = c_0, \ldots, c_n = d \in C$.

A typical Petri system SYS N is constructed from a Petri net N. One should take the set of all markings as configurations with M_0 as initial, events as elementary actions, and the transition relation as defined between markings of N. Altogether, we obtain a Petri system SYS $N = \langle \mathrm{Mark}_N, r\,E, \rightarrow, M_0 \rangle$ where $r\,E$ is the set of those events that can occur at a reachable marking. Let us remark that by their nature Petri nets give rise to *deterministic* systems, whether Petri systems, or transition systems. This is the main reason we have chosen to work with them at the cost of having a complex 2-level method of representation of non-determinism.

Next, we define morphisms between Petri systems. It should be evident that, similarly to the nets themselves, the morphisms of Petri nets give rise to the morphisms of the corresponding Petri systems.

Definition 3.2.3 A *morphism* $f: S \rightarrow S'$ of Petri systems S and S' is a pair $f = \langle f_C, f_A \rangle$ where:

$$f_C: C \rightarrow C' \quad \text{and} \quad f_A: A \rightarrow A',$$

such that for $c, d \in C_S$ the following holds:

(i) $c \xrightarrow{\alpha} d$ implies $f_C(c) \xrightarrow{\mu f_A(\alpha)} f_C(d)$.

It is not difficult to check that we have defined a category of Petri systems. Moreover, the translation of Petri nets into Petri systems is functorial.

Fact 3.2.4 *Petri systems with their morphisms constitute a category, denoted* PSys. *The construction of a Petri system* SYS N *from a net* N *extends to a functor* SYS: PNet $\rightarrow$ PSys.

Proof. The proof of the first claim is standard and therefore omitted.

As far as the second is concerned we note that SYS N is a Petri system. It obviously satisfies conditions 3.2.2(i)–(iii). Condition 3.2.2(iv) follows from the linearity of μ-functions:

$$\begin{aligned} M \xrightarrow{A+B} M' &\iff M' = M - \mu^\bullet(A+B) + \mu(A+B)^\bullet \\ &\iff M' = M - \mu^\bullet(A) - \mu^\bullet(B) + \mu(A)^\bullet + \mu(B)^\bullet \\ &\iff M' = (M - \mu^\bullet(A) + \mu(A)^\bullet) - \mu^\bullet(B) + \mu(B)^\bullet. \end{aligned}$$

Finally, from Proposition 3.1.4 and Corollary 3.1.5 it follows that morphisms of Petri nets are translated to morphisms of Petri systems. □

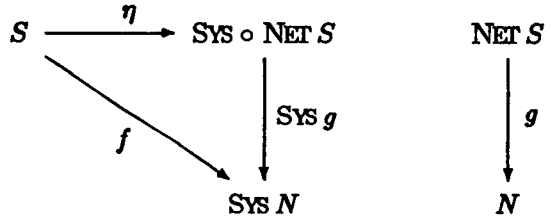
3.3 A Negative Result

Petri systems arise from Petri nets in a process of abstraction. It is natural therefore to think of the former as giving a semantics to Petri nets. We stressed already that they are intended to capture a notion of dynamic behaviour of nets.

From the practical point of view any formalism must provide a versatile and manageable descriptions of objects. We have seen in Example 3.2.1 that Petri nets can provide different descriptions of concurrent systems with the same dynamic behaviour. This prompts the question of means to cope with this variety. Ideally, there should exist a *universal* net among those that exhibit a given abstract behaviour.

The above is an instance of a more general problem. Namely, one can ask if there exists a universal way in which abstract behaviours can be realised by means of Petri nets. The more general problem can be formalised as the problem of existence of a (co)reflection between **PNet** and **PSys** with the functor SYS as one of the adjoints.

Let us try to explain in more elementary terms what the existence of such a situation would mean. For the purpose of argument let us consider the case of SYS being the right adjoint in a coreflection. Then we would have a functor NET: **PSys** → **PNet** s.t., for any net N, system S, and a morphism $f: S \to \text{SYS}\, N$ the following diagram would commute:



for a unique morphism g.

Moreover, as a consequence of Theorem 2.2.3, the morphism η would be an isomorphism. This means that the functor NET would construct nets with given behaviour, and the initial among them since any other net N with $f: S \xrightarrow{\cong} \text{SYS}\, N$ would be an image of NET S via (a unique) morphism.

Unfortunately, this does not take place. We will show that neither of the categories can be thought of as a (co)reflective subcategory of the other with SYS as one of the functors of the adjunction. Thus, the problem of finding a universal realisation of a given behaviour in general Petri nets is insoluble. In fact we will show more. Namely, that there are Petri systems that are not realisable as Petri nets at all.

3.3.1 PNet is not (Co)reflective in PSys

Both negative results are based on Theorem 2.2.3. Specifically, we show that the functor SYS has neither left, nor right inverse.

Proposition 3.3.1 SYS *has not a left-inverse.*

Proof. Assume to the contrary, and let the left-inverse be NET: **PSys** → **PNet**.

Since SYS $N_c \simeq$ SYS N_d, for N_c, N_d as in the Example 3.2.1 then, since functors preserve isomorphisms we obtain $N_c \simeq$ NET ∘ SYS $N_c \simeq$ NET ∘ SYS $N_d \simeq N_d$. But this is clearly impossible since the two nets are not isomorphic. In fact, the reader may easily verify, there are no morphisms between them that would have a bijective event part. □

Corollary 3.3.2 PNet *is not (co)reflective in* **PSys**.

Proof. From Theorem 2.2.3 it follows that **PNet** is not coreflective in **PSys**. Dual to Theorem 2.2.3 implies that **PNet** is not reflective in **PSys**. □

3.3.2 PSys is not (Co)reflective in PNet

The previous result is not very surprising. Intuitively, the functor SYS forgets a lot of structural information. Therefore, one would rather expect to find **PSys** isomorphic to a subcategory of **PNet**. Mathematical practice gives many examples of forgetful functors. Consider functor U: **Monoid** → **Set** from the category of monoids and their homomorphisms to the category of sets. It sends a monoid to its carrier, and forgets that the functions preserved the monoidal structure. It is well known that this functor, as well as many others, has a left adjoint. In fact the left adjoint to U sends a set X to the monoid X^*, and a function $f: X \to Y$ to the monoid homomorphism $f^*: X^* \to Y^*$.

By analogy, we would rather expect to find NET: **PSys** → **PNet** as a left adjoint to SYS in the coreflection. This would mean that **PSys** can be identified with a (full) coreflective subcategory of **PNet**.

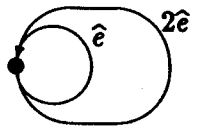
It is, however, impossible.

Proposition 3.3.3 SYS *has not a right-inverse.*

Proof. Let us assume to the contrary that NET: **PSys** → **PNet** is a right inverse to SYS.

Next, let us take the net N_2 drawn as the first example. The corresponding system $S_2 =$ SYS N_2 can be drawn like this:

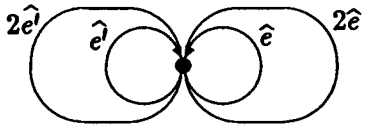
S_2



Above the black disc denotes the unique state of the system while the transition relation is represented by arrows labelled with actions = multisets of elementary actions.

It can be shown that the category **PSys** has (binary) coproducts, and that the system S_{2+2} below is a coproduct of two copies of S_2.

S_{2+2}



By assumption it follows that SYS $\circ$ NET $S_{2+2} \simeq S_{2+2}$. The crucial observation is that no net can give rise to S_{2+2} via SYS, hence the result follows.

For assume that such a net, N, exists. It must have at least two transitions, let us call them a and b, and a non-empty set of places. Since S_{2+2} has only one configuration it follows that M_0 is the only reachable marking of N, and also that in N we have:

$$M_0 \xrightarrow{\hat{a}} M_0, \quad M_0 \xrightarrow{2\hat{a}} M_0, \quad M_0 \xrightarrow{\hat{b}} M_0 \quad \text{and} \quad M_0 \xrightarrow{2\hat{b}} M_0,$$

$$\text{but not } M_0 \xrightarrow{\hat{a}+\hat{b}} M_0.$$

In other words for M_0 we have:

$$M_0 \geq 2^\bullet(a) \geq {}^\bullet(a) \quad \text{and} \quad M_0 \geq 2^\bullet(b) \geq {}^\bullet(b) \quad \text{and} \quad M_0 \not\geq {}^\bullet(a) + {}^\bullet(b).$$

The last condition implies that for some place $x \in B$ we have: ${}^\bullet(a)\,x + {}^\bullet(b)\,x > M_0\,x$. But then we obtain:

$$M_0\,x \geq \max(2^\bullet(a)\,x, 2^\bullet(b)\,x) \geq {}^\bullet(a)\,x + {}^\bullet(b)\,x > M_0\,x.$$

Contradiction. □

Corollary 3.3.4 **PSys** *is not (co)reflective in* **PNet**.

Proof. Same as the proof of Corollary 3.3.2 □

Unfortunately, we cannot offer a stronger result that SYS is not an adjoint.

Let us finish this short Chapter by drawing some conclusions.

In general a binary coproduct in **PSys** is constructed by simply gluing together the initial configurations of the systems involved. The coproduct operation will play a vital role later in interpreting the nondeterministic *sum* operator of *CCS*-like languages. Therefore, the existence of coproducts in **PSys** makes the category nicer than **PNet** in which coproducts do not exist. The last was shown by Winskel in [Winskel 87]. His proof remains valid in the category of Petri nets we consider.

The example on which the proof of the last Proposition is based is of slightly different nature than Winskel's result.

The generality of nets considered creates problems. To avoid them it seems necessary to restrict attention to a better, more cocomplete and regularly behaving sub-categories of Petri nets and Petri systems. This will be the subject of the next Chapters, although, the main attention will be devoted to the subcategories of **PSys**.

Chapter 4

Asynchronous Systems

In the previous Chapter it was argued that the general category of Petri nets considered there is too large. The same applies therefore to the related category of Petri systems. *Asynchronous systems* are introduced here to take place of Petri systems as more abstract, simpler objects. In this way we mimic a similar trend in the theory of Petri nets where most results are obtained for *safe nets* and their subclasses.

Asynchronous systems are closely related to a variety of other models for concurrency. First, we find that the *traces* of Mazurkiewicz are a natural choice for *computations* of asynchronous systems. Other links with safe Petri nets, and Winskel's event structures will be developed in the next Chapter.

This Chapter is organised as follows.

The first section introduces the category **AS** of asynchronous systems. We proceed to establish links between asynchronous systems and Petri systems. To this end a full subcategory **APSys** of **PSys** is defined, and we show that asynchronous systems are abstractions of the systems from **APSys**. In fact **AS** can be regarded as a reflective, or a coreflective subcategory of **APSys**.

Also in this section a full coreflective subcategory **seqAS** of **AS** is defined. It contains systems that are intuitively *sequential*, and it happens to be in disguise the category of transition systems defined informally in Chapter 3.

The section ends with an introduction of the category **LAS** of *labelled asynchronous systems*. A category equivalent to **LAS** is studied more intensively in Part II where it plays central role in the studies of the non-interleaving semantics of languages for synchronisation. Therefore, there is no need to study categorical properties of asynchronous systems here.

In the second section traces are introduced as computations of asynchronous systems, and their elementary properties are studied. Traces are equivalence classes of a congruence relation on the free monoid of sequences. Therefore, they form a quotient monoid. The algebraic structure of this monoid determines a natural partial order relation on traces called *prefix*. Properties of the resulting partially ordered set of traces are studied. The results reported here are not new, but are given for the sake of completeness of the presentation.

Moreover, it would be difficult to point to a reference in which all of them can be found.

In the last section two representation theorems are established. First, we show that the set of computations of an asynchronous system is a non-empty and prefix-closed *trace language*, i.e., set of traces. Conversely, every such set determines an asynchronous system that accepts it. This gives rise to a coreflection between **AS**, and a suitably chosen category of trace languages.

This connection is shown to cut down to forward stable asynchronous systems on one side, and non-empty *ideals* of traces on the other.

4.1 Category of Asynchronous Systems

4.1.1 The Definition

Asynchronous systems were introduced independently by Mike Shields and the author during British Theoretical Computer Science Colloquium at Warwick in 1986. It is Shields's terminology that the author has adopted for the purpose of presentation.

Definition 4.1.1 An *asynchronous system* S, abbr., *asys*, is a 5-tuple $S = \langle C, c^*, A, I, \rightarrow \rangle$ where

$c, d : C$ is a non-empty set of *configurations* of S,

$c^* \in C$ is an *initial configuration*,

$a, b : A$ is a set of *actions* of S,

$_\ I\ _ \subseteq A^2$ is an *independency* of S, i.e., a symmetric and irreflexive relation.

$_ \rightarrow _ \subseteq C \times A \times C$ is a graph of a partial function from $C \times A$ to C called *transition relation* of S,

such that for $a, b \in A$, $c, c', d \in C$ the following holds:

(i) $c \xrightarrow{a} c' \xrightarrow{b} d \wedge a\ I\ b$ implies $c \xrightarrow{b} d' \xrightarrow{a} d$, for some $d' \in C$.

An asynchronous system S is called *forward stable* iff, additionally,

(ii) $c \xrightarrow{a} c' \wedge c \xrightarrow{b} d \wedge a\ I\ b$ implies $c' \xrightarrow{b} d' \wedge d \xrightarrow{a} d'$, for some $d' \in C$.

Obviously, the 4-tuple $\langle C, c^*, A, \rightarrow \rangle$ is a transition system. It is called the underlying ts of S, and is denoted by S as well.

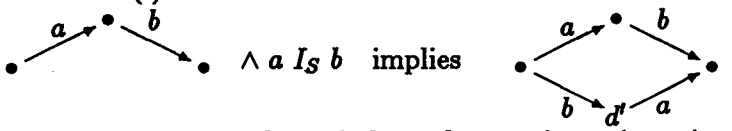
We shall refer to the items building an asynchronous system S by adding subscript S to the generic term. Thus, e.g., C_S denotes the set of configurations of S, while c^*_S its initial configuration, etc. There is only one exception to this rule, namely, we shall not mark the transition relation in any way.

By convention $_ \rightarrow _$ will also denote an extension of the transition relation onto $C \times A^* \times C$, defined in the same way as the one for Petri systems. If $c \xrightarrow{v} d$ in S for some $v = a_1 \ldots a_n \in A^*$ then we will say that v is a *path* from c to d in S. Every path determines and is determined by a unique sequence of configurations $c = c_0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} c_n = d$. In particular, $c \xrightarrow{\varepsilon} c$.

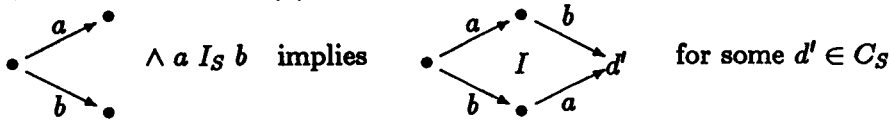
It is better still to develop a graphical presentation for objects of our discourse. Each asynchronous system S can be drawn as a graph with certain additional information.

Configurations on the pictures are represented either by their names, or as blobs of ink if we are not interested in their identity. The transition relation is represented by arrows labelled with names of actions.

Visual representation is helpful in understanding the conditions of the Definition 4.1.1. Condition 4.1.1(i) can be restated as follows:



Thus, any sequence of two independent actions gives rise to a little parallelogram. To stress that the actions involved are independent we draw the letter I in the middle of it. Hence, the condition 4.1.1(ii) can be re-written as:



Now, we introduce an appropriate notion of morphisms between asynchronous systems.

Definition 4.1.2 Let S, T be asynchronous system-s. A *morphism* f from S to T, denoted $f: S \to T$, is a pair $f = \langle f_C, f_A \rangle$, where

$f_C: (C_S, c_S^*) \to (C_T, c_T^*)$ is a morphism in $\mathbf{Set}_*$, i.e., a (total) function that preserves the distinguished points,

$f_A: A_S \rightharpoonup A_T$ is a partial function,

such that for $a, b \in A_S$ the following conditions are satisfied:

(i) $f_A(a)\downarrow \wedge\, c \xrightarrow{a} d$ implies $f_C(c) \xrightarrow{f_A(a)} f_C(c)$, and
$f_A(a)\uparrow \wedge\, c \xrightarrow{a} d$ implies $f_C(c) = f_C(c)$,

(ii) $f_A(a)\downarrow \wedge\, f_A(b)\downarrow \wedge\, a\, I_S\, b$ implies $f_A(a)\, I_T\, f_A(b)$,

The pairs of identity functions form the *identity* morphisms.
The *composition* of morphisms is defined componentwise.

Thus, the morphisms simply preserve the structure of asynchronous system-s. The following result is immediate.

Fact 4.1.3 *Asynchronous systems and their morphisms form a category, denoted* **AS**.

Proof. It is a simple exercise to check that the axioms of a category are satisfied. □

By **fsAS** we denote the full subcategory of **AS** with forward stable systems.

4.1.2 Relationship with Petri Systems

Next, we want to argue that asynchronous systems can be seen as a (full) subcategory of **PSys** — the category of Petri systems. In fact it is possible to define such inclusion in at least two ways. This shows that there is a substantial loss of information involved when the attention is restricted to **AS** — the category of asynchronous systems. This is illustrated with appropriate examples.

Let us consider a full subcategory **APSys** of **PSys**. It contains only those Petri systems S in which concurrency is *global* or *uniform*, and *flat*, i.e., no action occurs with multiplicity greater than one. Formally, for all $a, a' \in A_S$ the following holds:

$$\exists c, c' \in C_S \,.\, c \stackrel{\widehat{a} + \widehat{a'}}{\longrightarrow} c' \;\Rightarrow\; a \neq a' \wedge \forall d, d', d'' \in C_S \,.\, (d \stackrel{\widehat{a}}{\longrightarrow} d' \stackrel{\widehat{a'}}{\longrightarrow} d'' \;\Rightarrow\; d \stackrel{\widehat{a} + \widehat{a'}}{\longrightarrow} d'').$$

The first observation is that the systems from this subcategory give rise to asynchronous system in a natural way. We can define a functor U: **APSys** $\rightarrow$ **AS** as follows.

On objects: $U\,S = \langle C_S, c^*_S, A_S, I, \rightsquigarrow \rangle$ where

$a \; I \; a'$ iff $c \stackrel{\widehat{a} + \widehat{a'}}{\longrightarrow} d$, for some $c, d \in C_S$, and

$c \stackrel{a}{\rightsquigarrow} d$ iff $c \stackrel{\widehat{a}}{\longrightarrow} d$.

On morphisms: $U\,f = f$, i.e., we take the same pair of maps only consider it in a different category.

The task of showing that we have defined a functor is left to the reader.

In order to show that **AS** can be seen as a subcategory of **PSys** we indicate that U defined above has both left and right adjoints.

Let F, G: **AS** $\rightarrow$ **APSys** be functors defined as follows.

On objects: $F\,S = \langle C_S, c^*_S, A_S, \rightsquigarrow \rangle$ where

$$c \stackrel{\alpha}{\rightsquigarrow} d \;\text{ iff }\; \exists v = a_1 \ldots a_n \in A^*_S \,.\, (i \neq j \;\Rightarrow\; a_i I_S a_j) \wedge c \stackrel{v}{\longrightarrow} d \wedge \alpha = \sum\{\widehat{a_i} \mid i \leq n\}.$$

In other words we allow $c \stackrel{\alpha}{\rightsquigarrow} d$ only by a μ-set that results from a finite permutation v of a set of pairwise independent actions of A_S, and such that $c \stackrel{v}{\longrightarrow} d$ in S.

On morphisms: $F\,f = f$.

The definition of G differs from that of F only by allowing the permutations to have the length of at most 2.

We leave without proof the claim that both F and G are functors.

Let us expand a little on the merits of the above constructions. Both of them try to expand the transition relation of S onto $C_S \times \mu A_S \times C_S$ in a way that would give back S after application of U to the resulting Petri system. It should be obvious that $U \circ F = U \circ G = id_{\mathbf{AS}}$.

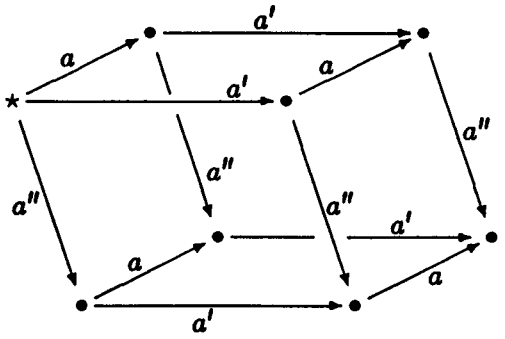
Functor F constructs a Petri system in which there is as much concurrency as possible. Functor G gives the minimally concurrent Petri system.

Finally, we remark that the pairs of identity maps: $\phi_S : S \to F \circ U\, S$ and $\psi_S : G \circ U\, S \to S$ form the unit and the counit of the adjunctions $G \dashv U \dashv F$, respectively. The reader should be able to fill in the details easily.

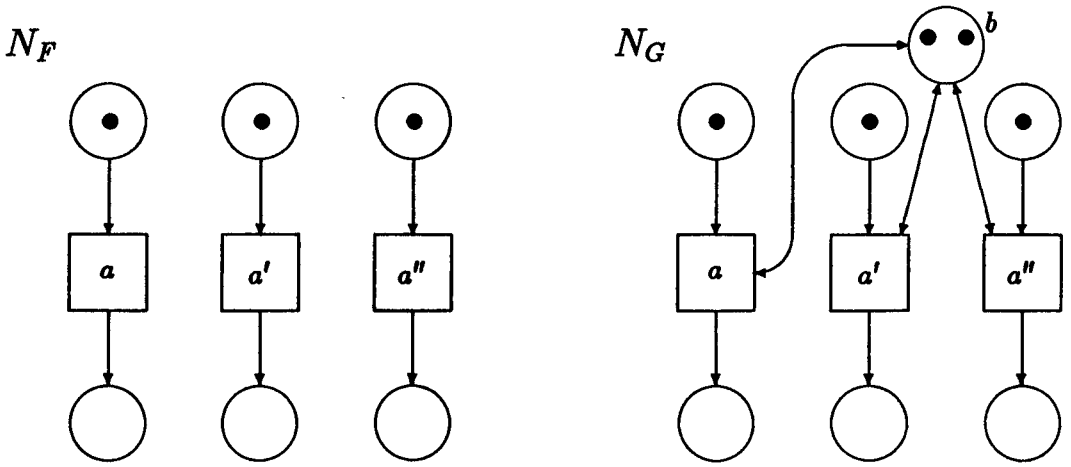
Thus, we can think of **AS** as a reflective, or coreflective subcategory of **APSys**.

To make things more concrete let us consider a simple example.

Example 4.1.4 Let S be an asynchronous described below with exception of the independency relation omitted for clarity. We assume that $a\ I_S\ a'\ I_S\ a''\ I_S\ a$ holds in S.



The corresponding Petri systems $F\,S$ and $G\,S$ are presented by Petri nets N_F and N_G, respectively. That is, we claim, the systems are isomorphic to the results of application SYS to corresponding nets.



Above, the arrows going in both directions abbreviate two arrows going in the opposite directions, or more formally: $\widehat{b} \leq {}^\bullet(a) \wedge \widehat{b} \leq (a)^\bullet$, etc.

The effect of the extra place b with 2 blobs and connections to and from any transition makes all the difference. As a result in the net N_G we cannot observe all the three actions happening at the same time. Therefore, one can argue, the two systems, or nets do exhibit different behaviour. Since we prefer to work with asynchronous systems we implicitly choose not to distinguish between them. ◇

4.1.3 Sequential Systems

There is a natural place among asynchronous systems for the systems that do not exhibit concurrency. We will call them *sequential.*

A sequential system S is one in which $c \xrightarrow{a} c' \xrightarrow{b} d$ does not hold for actions a and b s.t., $a\ I_S\ b$. Obviously, every asynchronous system that satisfies: $I_S = \emptyset$, is sequential.

Actually, we can consider a category **seqAS** of sequential asynchronous systems defined as the full subcategory of **AS** in which every system has the empty independency relation. The reader should notice that this category is isomorphic to the category of transition systems described informally in Chapter 3, and also to the category **ppGrph** introduced in Chapter 2.

Every asynchronous system S can be *sequentialised*. Let $\mathbb{SEQ}\, S = \langle C_S, c^*_S, A_S, \emptyset, \rightarrow \rangle$, i.e., the independency relation of S is replaced with $\emptyset$. It is left as an exercise to verify that the construction is functorial with $\mathbb{SEQ}\, f = f$, and that the resulting functor is right adjoint to the inclusion: **seqAS** $\hookrightarrow$ **AS**.

Therefore, the category of transition systems is a coreflective subcategory of **AS**.

4.1.4 Labelled Asynchronous Systems

We finish this section by introducing formally labelled asynchronous systems. A category equivalent to the category of labelled asynchronous systems will be extensively studied in Part II.

Definition 4.1.5 A *labelled asys* S, abbr.: *las*, is a 7-tuple $S = \langle C, c^*, A, I, \rightarrow, M, l \rangle$ where

$\langle C, c^*, A, , I, \rightarrow \rangle$ is the underlying asys of S, also denoted by S,
M is a set of *labels*, and
$l: A \rightharpoonup M$ is a partial *labelling* function.

Morphisms of labelled asynchronous systems must cope additionally with labelling. To this end an extra partial function is added, and a requirement that the labelling is to be preserved. The formal definition follows.

Definition 4.1.6 Let S, T be labelled asynchronous system-s. A *morphism* f from S to T, denoted $f: S \rightarrow T$, is a triple $f = \langle f_C, f_A, f_M \rangle$ where

$f = \langle f_C, f_A \rangle$ is a morphism of the underlying asynchronous systems,
$f_M: M_S \rightharpoonup M_T$ is a partial function,

such that:

(iii) $f_M \circ l_S = l_T \circ f_A$.

The triples of identity functions form the *identity* morphisms.
The *composition* of morphisms is defined componentwise.

The following result is then standard.

Fact 4.1.7 *Labelled asynchronous systems with their morphisms form a category, denoted* **LAS**.

The full subcategory of **LAS** with forward stable systems is denoted **fsLAS**.

4.2 Mazurkiewicz Traces — Computations of Asynchronous Systems

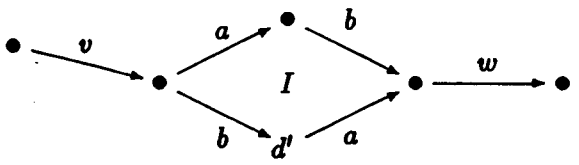
Readers unfamiliar with Mazurkiewicz's ideas should be aware — the notion central to this section should not be confused with the meaning prescribed to the word "trace" by researches in Oxford. It remains a total mystery to the author why one should introduce another name for such an old good term as "sequence".

The labelling part of the systems is omitted from considerations, and we talk about asynchronous systems only. The following discussion can be obviously conducted with the labelling around. It would be, however, a fruitless exercise, as the labelling part would not contribute to the subject otherwise than by complicating the presentation.

4.2.1 Motivation

Let $S = \langle C, c^*, A, I, \rightarrow \rangle$ be an asynchronous system fixed for the purpose of this section. And let $c'' \xrightarrow{v} c \xrightarrow{a} c' \xrightarrow{b} d \xrightarrow{w} d''$ in S. Thus, $c'' \xrightarrow{vabw} d''$, and so $vabw$ is a path from c'' to d'' in S.

Now, assume that $a\ I\ b$ holds. Then from the definition of asynchronous systems it follows that for some $d' \in C_S$ we have:



This situation can be given the following interpretation.

In S the actions a and b are independent. This, by the nature of asynchronous systems, means that the change caused by a followed by a change caused by b has the same effect as the changes performed in the inverse order. Therefore, the order in which they happen is irrelevant. If we could see better perhaps we would see them happening simultaneously, compare $c \overset{\widehat{a}+\widehat{b}}{\rightsquigarrow} d$ in FS or GS!

Thus, the two paths: $vabw$ and $vbaw$ can be thought of as representing the same *computation*, and hence are equivalent.

This type of argument must have led Mazurkiewicz when he introduced the notion of a *trace* as an equivalence class of sequences where the equivalence relation was the least one that identifies $vabw$ with $vbaw$ above.

Formally, let $=_I \subseteq A \times A$ be defined as follows:

$=_I = (-_I)^*$, i.e., a reflexive and transitive closure of relation $-_I$ defined by:

$$-_I = \{\langle vabw, vbaw\rangle \mid a\ I\ b \wedge a, b \in A \wedge v, w \in A^*\}.$$

It is obvious that in this way an equivalence relation is defined. In fact, $=_I$ is a congruence w.r.t., operation of concatenation of sequences. Algebraically, $A^* = \langle A^*, _\cdot_, \varepsilon\rangle$ is a monoid, where $_\cdot_$ denotes the (usually denoted as juxtaposition) concatenation operation, and ε the empty sequence. The resulting quotient monoid $A^*/=_I$ will be denoted $t{:}\, A_I^*$ and referred to as the *monoid of traces* on A w.r.t., independency I.

The trace represented by a sequence v will be denoted by $[v]$.

From the above discussion it follows that the following important property holds.

Proposition 4.2.1 *Let v be a path from c to d in S, and let $w =_I v$. Then w is a path from c to d in S.*

Proof. By definition of $=_I$ we have $w -_I^n v$, for some $n \in \omega$, where $-_I^n = -_I \circ \ldots \circ -_I$ n times. The validity of the result for $n = 0$ is trivial, while for $n = 1$ follows from the discussion. The proof of the general case follows by induction. □

As a consequence of the above Proposition we can take traces as computations of asynchronous systems. Namely, an 'extension' of the transition relation onto $C \times A_I^* \times C$ given by: $c \xrightarrow{[v]} d$ iff $c \xrightarrow{v} d$, is well defined.

The set L_S of *traces accepted by* S is defined by: $L_S = \{t \in A_I^* \mid \exists c \in C\,.\, c^* \xrightarrow{t} c\}$.

There is one more question we want to ask before we turn to more systematic study of traces and their languages. Assume that $f{:}\, S \to S'$ is a morphism of asynchronous systems S and $S' = \langle C', c'^*, A', I', \to\rangle$. The question is what is the relationship between the trace languages accepted by S and S'.

First, we note that every partial function, here $f_A{:}\, A \rightharpoonup A'$, determines a homomorphism of monoids $f_A^*{:}\, A^* \to A'^*$ defined uniquely if we demand that the following properties are satisfied:

$$f_A^*(\langle a\rangle) = \begin{cases} \langle f_A(a)\rangle & f_A(a)\downarrow \\ \varepsilon & \textit{otherwise} \end{cases}.$$

Above, for clarity, we have refrained from using the usual convention, and omitting the square brackets around the sequences, here singletons.

Now, we can prove the following.

Proposition 4.2.2 *Let A, A' be sets, and I, I' independency relations on A, and A', respectively. If $f{:}\, A \rightharpoonup A'$ satisfies $a, b \in \mathrm{dom} f \;\wedge\; a\, I\, b \;\Rightarrow\; f(a)\, I'\, f(b)$ then*

$v =_I w$ *implies* $f_A^*(v) =_{I'} f_A^*(w)$ *for* $v, w \in A^*$.

Such f determines a homomorphism of monoids $f^{\natural}{:}\, A_I^* \to A_I'^*$ given by: $f_A^{\natural}([v]) = [f_A^*(v)]$. Finally, if $f{:}\, S \to S'$ is a morphisms of asynchronous systems then $f_A^{\natural}(L_S) \subseteq L_{S'}$.

Proof. If f preserves independency relation in the above sense then $v -_I w$ implies $f^*(v) -_{I'} f^*(w)$ or $f^*(v) = f^*(w)$.

Let us note that $c \xrightarrow{v} d$ in S implies $f_C(c) \xrightarrow{f_A^*(v)} f_C(d)$ in S', can be proved easily by induction on the length of v. Hence, the second property follows. □

The discussion above reflects, it is hoped, the spirit of the original paper in which Mazurkiewicz introduced the notion of a trace, see [Mazurkiewicz 77]. The idea was to use traces as computations of safe nets.

One should add that there are other Computer Science and algebraic motivations to study the monoids of traces, also known as *partially commutative monoids*. The reader can find relevant informations in an overview paper [Aalbersberg&Rozenberg 86]. See also Mazurkiewicz's article in [APN'86/II]

4.2.2 Elementary Properties of Traces

Most of the results presented are part of the folklore of the subject. Some of them can be found in [Shields 85], although Shields works with a presentations of traces as *vectors of strings*. Citation [Aalbersberg&Rozenberg 86] presents most, if not all of them, unfortunately, without proofs. We present them all with enough hints to the reader to enable her/him recover missing details.

Let a set A, and an independency $I \subseteq A \times A$ on it be fixed for the purpose of this paragraph. By convention the same notation will be used for the empty sequence ε, and the empty trace $[\varepsilon] = \{\varepsilon\}$. Similarly, we write a for an element of A, but also for the sequence $\langle a \rangle$ consisting of this element, and the trace $[\langle a \rangle]$.

For the purpose of this section only we let p, q, r, s and t denote traces in A_I^*. Letters u, v and w will be used to denote sequences A^*.

It was already mentioned that $=_I$ is a congruence of monoids. This follows immediately from the observation that $v -_I^k v'$ and $w -_I^l w$ implies $vw -_I^{k+l} v'w'$, for $v, v', w, w' \in A^*$. As a result the composition in the quotient monoid A_I^*, i.e., *concatenation* of traces, is defined by: $[v][w] = [vw]$, for $v, w \in A^*$.

Only sequences of the same length can compose a trace. Therefore, the notion of a *length* of a trace is inherited by taking the length of any of its representatives. As a consequence the composition of traces is length increasing. This shows that the *prefix* relation defined on traces by: $t \preceq t'$ iff $t' = tt''$, for some $t'' \in A_I^*$, is a partial ordering.

In other words the monoidal structure on traces makes it into a poset. We will be interested mainly in the properties of this poset. One of them can be stated already.

Lemma 4.2.3 *Poset* $\langle A_I^*, \preceq \rangle$ *is* finitary, *i.e.,* $\{p \in A_I^* \mid p \preceq t\} \in \mathcal{F}in\, A_I^*$, *for each* $t \in A_I^*$.

Proof. We can give $(n+1)!$ as an upper bound of the number of prefixes of any trace of length n, since the function

$$T(n) = \text{the maximal number of prefixes a trace of length } n \text{ may have}$$

satisfies the following inequalities:

$$T(0) = 1, \quad \text{and}$$
$$T(n+1) \leq (n+1)T(n) + 1, \quad \text{for } n \in \omega.$$

□

We do need, however, to investigate some of the properties of the monoid. Most of the technical results are collected in the following Lemma. The notation $(-_I)^{\circledast}$ is used as an abbreviation for $\bigcup_{i=0}^{n}(-_I)^i$.

Lemma 4.2.4 *For $a,b \in A$, $u,v \in A^*$, $p,q,t,t' \in A_I^*$, $n \in \omega$ the following hold:*

(i) $pq = \varepsilon$ *implies* $p = \varepsilon = q$,

(ii) $au(-_I)^{\circledast}bv$ *implies* $a = b \wedge u(-_I)^{\circledast}v$ *or* $a\ I\ b \wedge u(-_I)^{\circledast}bw \wedge v(-_I)^{\circledast}aw$, *for some* $w \in A^*$, $k,l \in \omega$, $k+l < n$,

(iii) $ap = bq$ *implies* $a = b \wedge p = q$ *or* $a\ I\ b \wedge p = bt \wedge q = at$, *for some* $t \in A_I^*$,

(iv) $pa = qb$ *implies* $a = b \wedge p = q$ *or* $a\ I\ b \wedge p = tb \wedge q = ta$, *for some* $t \in A_I^*$,

(v) **(Cancellation Law)** $ptq = pt'q$ *implies* $t = t'$.

Proof. (i) Trivial.

(ii) By induction on n. While conceptually easy the proof is long as there are many cases to consider.

(iii) Immediate consequence of (ii).

(iv) The proof is a symmetric version of the proof of (iii).

(v) By induction of the length of p and q from (iii) and (iv). □

Any subset L of A_I^* will be referred to as a *trace language*. The concatenation is extended on trace languages by: $LL' = \{tt' \in A_I^* \mid t \in L \wedge t' \in L'\}$. If $L = \{a\}$ we will write simply aL'.

If $t \preceq t'$ then there is a unique trace t'' s.t., $tt'' = t'$, as a consequence of the Cancellation Law. This trace will be called *derivative of t' w.r.t., t*, and denoted: t'/t.

Operation of taking a derivative of a trace language L w.r.t., a trace $t \in A_I^*$ is defined by: $L/t = \{t' \in A_I^* \mid tt' \in L\}$. It is easy to verify that $a\ I\ b$ implies $(L/a)/b = L/[ab] = (L/b)/a$.

Every non-empty set of traces has a greatest lower bound. To show this a notion of a rank of a non-empty trace language is introduced. A *rank* of L, denoted rkL, is the length of the shortest trace in L.

Next, let *glb*: $\mathcal{P}_+A_I^* \to \mathcal{P}_+A_I^*$ be defined by:

$$glb\,L = \begin{cases} \{\varepsilon\} & L \neq a(L/a), \text{ for } a \in A \\ a\,glb(L/a) & otherwise \end{cases}.$$

We can prove the following result.

Lemma 4.2.5 *For every non-empty $L \subseteq A_I^*$ there exists $t \in A_I^*$ s.t., $glbL = \{t\}$, and t is a greatest lower bound of L.*

Proof. By induction on the rank of L. There are two possibilities.

$\boxed{L \neq a(L/a), \text{ for } a \in A}$ This includes the basic induction case where $rkL = 0$. Then ε is the only lower bound of L. Hence, the greatest.

$\boxed{L = a(L/a), \text{ for some } a \in A}$ Let $A' = \{a \in A \mid L = a(L/a)\}$. Then for $a, b \in A'$ either $a = b$ or $a \; I \; b$ follows from 4.2.4(iii). Since $rk(L/a) < rkL$ it follows by induction that $glb(L/a)$ contains the greatest lower bound of L/a only. Let us verify that $a \, glb(L/a)$ does not depend on the choice of a.

If $a, b \in A'$, $a \neq b$ then $a \; I \; b$, so $L/a = b((L/a)/b)$, by Lemma 4.2.4. Whence, $a \, glb(L/a) = [ab] \, glb((L/a)/b) = [ab] \, glb(L/[ab]) = [ab] \, glb((L/b)/a) = b \, glb(L/b)$. The reader is asked to verify that $glb(L/a) = \{t\}$, t — a greatest lower bound of L/a implies at — a greatest lower bound of L. □

Above result has important consequences.

A poset $\mathbf{P} = \langle P, \leq \rangle$ is called *meet semilattice* iff it has greatest lower bounds of all pairs of elements. It is called ***consistently complete*** iff every bounded subset of A_I^* has a least upper bound.

Corollary 4.2.6 *Poset $\langle A_I^*, \preceq \rangle$ is a consistently complete meet semilattice.*

Proof. The proof is standard. □

The greatest lower bound of set $\{t, t'\}$ will be denoted $t \sqcap t$. Here are other consequences of the last Lemma.

Corollary 4.2.7 *(i)* $t(t' \sqcap t'') = tt' \sqcap tt''$, *for* $t, t', t'' \in A_I^*$,

(ii) $(t/(t \sqcap t')) \sqcap (t'/(t \sqcap t')) = \varepsilon$, *for* $t, t', t'' \in A_I^*$.

Proof. (i) $t(t' \sqcap t'') = tglb\{t', t''\} = glb\{tt', tt''\} = tt' \sqcap tt''$.

(ii) Let us assume to the contrary: $a \preceq (t/t \sqcap t') \sqcap (t'/t \sqcap t')$, for some $a \in A$. Then $(t \sqcap t')a \preceq t$, and $(t \sqcap t')a \preceq t'$. So, $(t \sqcap t')a \preceq t \sqcap t'$ — contradiction, since the length function is order preserving. □

With the help of greatest lower bounds we can characterise least upper bounds in A_I^*. Let us start from binary lub-s.

First, let us define $I^{\natural} \subseteq A_I^* \times A_I^*$, an extension of the independency relation as follows:

$$p \; I^{\natural} \; q \quad \text{iff} \quad p = rar' \wedge q = sbs' \Rightarrow a \; I \; b, \quad \text{for } a, b \in A, \; r, r's, s' \in A_I^*.$$

In other words two traces are independent iff all actions of one are independent of all actions of the other. As a consequence we see that the relation $I^{\natural}$ is a conservative extension of I onto the set of traces: $a \; I^{\natural} \; b$ iff $a \; I \; b$, for $a, b \in A$. This extension is obviously symmetric, and almost irreflexive: $t \; I^{\natural} \; t$ iff $t = \varepsilon$. Another easy to verify consequence is that $p \; I^{\natural} \; q$ implies $pq = qp$.

As a preliminary step to characterisation of binary lub-s in the poset of traces we offer the following technical Lemma.

Lemma 4.2.8 *For all* $a, b \in A$, $p, q, t \in A_I^*$

(i) $pa \sqcap b = \varepsilon \wedge pa, b \preceq t$ *implies* $a\ I\ b \wedge pab = bpa \preceq t$,

(ii) $pa \sqcap qb = \varepsilon \wedge pa, qb \preceq t$ *implies* $a\ I\ b \wedge qbpa = paqb \preceq t$,

(iii) $p \sqcap q = \varepsilon \wedge p, q \preceq t$ *iff* $p\ I^{\natural}\ q \wedge pq \preceq t$.

Proof. (i) By induction on the length of p. We assume that $r, s \in A_I^*$ are such that $par = t = bs$.

$\boxed{p = \varepsilon}$ Basic case of induction. Then $a \sqcap b = \varepsilon$ implies $a \neq b$. Both results follow now from $ar = bs$, by 4.2.4(iii).

$\boxed{p = a'p'}$ We show that in this case $p' \sqcap b = \varepsilon$, and $p'a, b \preceq t'$, for $t' = p'ar$. The result follows then by induction.

First, $a'p'a \sqcap b = \varepsilon$ implies $a' \neq b$. From $a'p'ar = bs$ it follows by 4.2.4(iii) that $a'\ I\ b$, and $p'ar = bs'$, for some $s' \in A_I^*$. Thus, $p'a, b \preceq t' = bs'$.

Let us assume that $p'a \sqcap b \neq \varepsilon$, i.e., $p'a \sqcap b = b$. Then $b \preceq p'a$, so that $p'a = br'$, for some $r' \in A_I^*$. But then $pa \sqcap b = a'br' \sqcap b = ba'r' \sqcap b = b$. Contradiction.

(ii) By induction on the length of q. We assume that $r, s \in A_I^*$ are such that $par = t = qbs$.

$\boxed{q = \varepsilon}$ Basic case of induction. The problem reduces to (i).

$\boxed{q = b'q'}$ We show that in this case $pa \sqcap q'b = \varepsilon$, and $pa, q'b \preceq t'$, for some $t' \in A_I^*$. The result follows then by induction.

First, $pa \sqcap b'q'b = \varepsilon$ implies $pa \sqcap b' = \varepsilon$. From $par = b'q'bs$ it follows by (i) and Cancellation Law that $r = b'r'$ and $pab'r' = b'par'$. Hence, $pa, q'b \preceq t' = par'$.

The proof of $pa \sqcap q'b = \varepsilon$ is indirect, and very similar to the corresponding one in (i). We leave it to the reader.

(iii) The "if" part is trivial. The "only' if' follows from (ii). □

The lub of traces p and q, if exists, will be denoted by $p \sqcup q$. The following result gives us information on the conditions under which they exist, and the form of binary least upper bounds in the poset of traces.

Lemma 4.2.9 (Binary lub-s) *Let* $t_0, t_1, t, t' \in A_I^*$ *and* $t = t_0 \sqcap t_1$. *Then*

(i) $t_0, t_1 \preceq t'$ *implies* $(t_0/t)\ I^{\natural}\ (t_1/t)$ *and* $t(t_0/t)(t_1/t) \preceq t'$,

(ii) $(t_0/t)\ I^{\natural}\ (t_1/t)$ *implies* $t_0 \sqcup t_1 = t(t_0/t)(t_1/t)$.

Proof. (i) Since $t_0 = t(t_0/t) \preceq t'$ and $t_1 = t(t_1/t) \preceq t'$ then $t_0/t \preceq t'/t$ and $t_1/t \preceq t'/t$.

From Corollary 4.2.7(ii) it follows that $t_0/t \sqcap t_1/t = \varepsilon$.

So, $(t_0/t)\ I^{\natural}\ (t_1/t)$ and $(t_0/t)(t_1/t) \preceq (t'/t)$, by Lemma 4.2.8(iii).

Hence, $t(t_0/t)(t_1/t) \preceq t(t'/t) = t'$.

(ii) If $(t_0/t)\ I^{\natural}\ (t_1/t)$ then $t_0, t_1 \preceq t(t_0/t)(t_1/t)$. The result follows by (i). □

As a corollary we obtain a well known and often quoted result. It is, however, difficult to find its proof in the literature.

Corollary 4.2.10 (Levi's Lemma) *Let $t, t', t'', t''' \in A_I^*$. Then*

$$tt' = t''t''' \quad \textit{implies} \quad t = pq \wedge t' = rs \wedge t'' = pr \wedge t''' = qs, \quad \textit{for some } p, q, r, s \in A_I^*.$$

Moreover, we can assume that q and s above satisfy $q \; I^\natural \; r$.

Proof. Take $p = t \sqcap t''$, and let $q = t/p$ and $r = t''/p$. Then $q \; I^\natural \; r$, and $pqr \preceq tt'$, by Lemma 4.2.9(i). Since $qr = rq$ we can take $s = tt'/pqr$. □

Our next goal is to characterise arbitrary lub-s in the poset of traces. First, let us introduce yet another notion.

A meet semilattice $\mathbf{P} = \langle P, \leq \rangle$ is called *distributive* iff $q \sqcup r \downarrow$ implies $p \sqcap (q \sqcup r) = (p \sqcap q) \sqcup (p \sqcap r)$.

Lemma 4.2.11 *Poset $\langle A_I^*, \preceq \rangle$ is distributive.*

Proof. Let $t, t', t'' \in A_I^*$ and $t' \sqcup t'' \downarrow$.

First, let us notice that $t' \sqcup t'' \downarrow$ implies that $(t \sqcap t') \sqcup (t \sqcap t'') \downarrow$ by consistency completeness of $\langle A_I^*, \preceq \rangle$, since the set $\{t \sqcap t', t \sqcap t''\}$ has both t and $t' \sqcup t''$ as upper bounds. Hence, $(t \sqcap t') \sqcup (t \sqcap t'') \preceq t \sqcap (t' \sqcup t'')$. Let us turn to the converse.

From Lemma 4.2.9 it follows that $t' \sqcup t'' = pqr$ where $p = t' \sqcap t''$, $q = t'/p$, $r = t''/p$ and $q \; I^\natural \; r$. If $s' = t \sqcap (t' \sqcup t'')$ then obviously $pqr = s's''$, for some $s'' \in A_I^*$. From Levi's Lemma applied twice it follows that there are some traces $p', p'', q', q'', r', r''$ such that $p = p'p''$, $q = q'q''$, $r = r'r''$, $s' = p'q'r'$, $s'' = p''q''r''$, and moreover $p'' \; I^\natural \; q'r'$ and $q'q'' \; I^\natural \; r'r''$.

So, $q' \; I^\natural \; r'$ in particular, and therefore $t \sqcap (t' \sqcup t'') = s' = p'q' \sqcup p'r'$, by Lemma 4.2.9.

Since $p'' \; I^\natural \; q'$ implies $p''q' = q'p''$ we obtain $p'q' \preceq p'q'p''q'' = p'p''q'q'' = pq = t'$, and $p'q' \preceq p'q'r' = s' \preceq t$.

So, $p'q' \preceq t \sqcap t'$.

Similarly, $p'r' \preceq t \sqcap t''$. The result follows. □

The distributivity plays an important part in the proof of the following result. We will characterise first ternary lub-s, and subsequently arbitrary lub-s in the poset of traces.

Lemma 4.2.12 *(i) Let $p, q, r \in A_I^*$. Then $\bigsqcup\{p, q, r\}\downarrow$ iff $p \sqcup q\downarrow \wedge p \sqcup r\downarrow \wedge q \sqcup r\downarrow$.*

(ii) Let $L \subseteq A_I^*$. Then $\bigsqcup L\downarrow$ iff $L \in \mathcal{F}in\, A_I^*$ and $\forall p, q \in L \,.\, p \sqcup q\downarrow$.

Proof. (i) The "only if" part follows immediately from Corollary 4.2.6. So, let $p \sqcup q\downarrow$, $p \sqcup r\downarrow$ and $q \sqcup r\downarrow$. All we need is, also by Corollary 4.2.6, an upper bound on p, q and r.

Let us consider $p' = p \sqcup (q \sqcap r)$, $q' = q \sqcup (p \sqcap r)$ and $r' = r \sqcup (p \sqcap q)$. If we show that an upper bound of p', q' and r' exists then the result obviously follows.

Firstly, they are defined as a consequence of Corollary 4.2.6. For example p' since $p, q \sqcap r \preceq p \sqcup q$.

By distributivity $\Delta = p' \sqcap q' = (p \sqcup (q \sqcap r)) \sqcap (q \sqcup (p \sqcap r)) = (p \sqcap q) \sqcup (p \sqcap r) \sqcup (q \sqcap r)$, and similarly $(p' \sqcap r') = \Delta = (q' \sqcap r')$.

Moreover, $p' \sqcup q'\downarrow \wedge p' \sqcup r'\downarrow \wedge q' \sqcup r'\downarrow$, again as a consequence of Corollary 4.2.6. In fact $p' \sqcup q' = p \sqcup q$, but it does not play any role in the argument. What is important, however, we infer that $(p'/\Delta)\ I^\natural\ (q'/\Delta)\ I^\natural\ (r'/\Delta)\ I^\natural\ (p'/\Delta)$. Therefore, $(p'/\Delta)(q'/\Delta) = (q'/\Delta)(p'/\Delta)$, etc.

Hence, $p', q', r' \preceq \Delta(p'/\Delta)(q'/\Delta)(r'/\Delta)$.

(ii) "only if" Let $\bigsqcup L\downarrow$. Then $L \subseteq \{t \in A_I^* \mid t \preceq \bigsqcup L\} \in \mathcal{F}in\, A_I^*$, by Lemma 4.2.3. The rest is a consequence of consistent completeness of $\langle A_I^*, \preceq \rangle$.

"if" Let $L = \{t_0, \ldots, t_n\}$. The proof goes by induction on n. For $n \leq 3$ the result follows from (i). If $n > 3$ then also by (i) $(t_0 \sqcup t_1) \sqcup t_i\downarrow$, for $i = 2, \ldots, n$, and the problem is reduced to the set $L' = \{t_0 \sqcup t_1, \ldots, t_n\}$. Clearly, $\bigsqcup L = \bigsqcup L'$, and the result follows by induction. □

It will appear important to have a characterisation of traces that are bounded by lub-s of some sets of traces.

Lemma 4.2.13 *(i) Let $t \preceq t_0 \sqcup t_1$. Then $t = t_0' \sqcup t_1'$, for some $t_0' \preceq t_0$, $t_1' \preceq t_1$,*

(ii) Let $t \preceq t_0 \sqcup \ldots \sqcup t_n$. Then $t = t_0' \sqcup \ldots \sqcup t_n'$, for some $t_i' \preceq t_i$, $i = 0, \ldots, n$.

Proof. (i) The proof follows by induction on s where $ts = t_0 \sqcup t_1 = pqr$ with $p = t_0 \sqcap t_1$, $q = t_0/p$ and $r = t_1/p$.

For $s = \varepsilon$ we can take $t_0' = t_0$ and $t_1' = t_1$.

Otherwise, let $s = s'a$. As in the proof of the Lemma 4.2.11 we can find a decomposition $p = p'p''$, $q = q'q''$, $r = r'r''$, $ts' = p'q'r'$ and $a = p''q''r''$ s.t., $q'\ I^\natural\ p''\ I^\natural\ r'\ I^\natural\ q'$.

From $q'\ I^\natural\ r'$ it follows that $t \preceq p'q'r' = p'q' \sqcup p'r'$, by Lemma 4.2.9. Moreover, $p'q' \preceq p'q'p''q'' = p'p''q'q'' = pq = t_0$. Similarly, $p'r' \preceq t_1$. The result follows now by induction.

(ii) Follows from (i) and 4.2.12(ii). □

Let A, A' be sets, and I, I' independency relations on A, and A', respectively.

If $f: A \rightharpoonup A'$ is independency preserving, i.e., for $a, b \in \mathrm{dom}\, f$ assumption $a\ I\ b$ implies $f(a)\ I'\ f(b)$, then according to Proposition 4.2.2 it gives rise to a homomorphism of monoids of traces: $f^\natural: A_I^* \to A_I'^*$ given by: $f_A^\natural([v]) = [f_A^*(v)]$.

The properties of the homomorphism as a morphism between the posets of traces are subject of the next result.

Lemma 4.2.14 *Let A, A' be sets, and I, I' independency relations on A, and A', respectively. If $f: A \rightharpoonup A'$ preserves the independency then $f^\natural: A_I^* \to A_I'^*$ is order-preserving. Moreover, it preserves existing lub-s in $\langle A_I^*, \preceq \rangle$: $\bigsqcup X\downarrow$ implies $f^\natural(\bigsqcup X) = \bigsqcup f^\natural(X)$, for $X \subseteq A_I^*$.*

Proof. $f^{\natural}$ a homomorphism of monoids implies that it is order preserving:

$$p \preceq q \quad \text{implies} \quad f^{\natural}(q) = f^{\natural}(p)f^{\natural}(q/p) \quad \text{iff} \quad f^{\natural}(p) \preceq f^{\natural}(q).$$

It is also *continuous* in the sense that it preserves existing lub-s. For if $t \sqcup t'\downarrow$ then $t \sqcup t' = pqr$ with $p = t \sqcap t'$, $q = t/p$, $r = t'/p$ and $q \; I^{\natural} \; r$. But then $f^{\natural}(q) \; I^{\natural} \; f^{\natural}(r)$, since f preserves independency. Hence, $f^{\natural}(t \sqcup t') = f^{\natural}(p)f^{\natural}(q)f^{\natural}(r) = f^{\natural}(pq) \sqcup f^{\natural}(pr)$ follows easily with the help of Lemma 4.2.9. Preservation of arbitrary existing lub-s follows from the above by induction by Lemma 4.2.12. □

A language L of traces is called *prefix-closed* iff $t \preceq t' \in L$ implies $t \in L$. This notion is a generalisation of a notion of an ideal in a lattice to a poset of traces.

A *lattice* $\mathbf{P}$ is such a poset $\mathbf{P} = \langle P, \leq \rangle$ in which every two elements have a least upper bound, and a greatest lower bound. An *ideal* X in a lattice $\mathbf{P}$ is a subset of P that satisfies:

(i) $p \leq q \in X$ implies $p \in X$, and

(ii) $p, q \in X$ implies $p \sqcup q \in X$.

Those subsets of a poset that satisfy the first condition are often referred to as *lower-sets*. Hence, prefix-closed languages of traces are just lower-sets in a poset of traces.

Lower-sets play an important role in the studies of the posets. There are, however, at least two more faithful generalisations of the notion of ideal from lattices to arbitrary posets.

An *ideal* X in a poset $\mathbf{P}$ is a subset of P that satisfies:

(i) $p \leq q \in X$ implies $p \in X$, and

(ii) $p, q \in X \wedge p \sqcup q\downarrow$ implies $p \sqcup q \in X$.

A *strong ideal* X in a poset $\mathbf{P}$ is a subset of P that satisfies:

(i) $p \leq q \in X$ implies $p \in X$, and

(ii) $p, q \in X$ implies $p \sqcup q\downarrow \wedge p \sqcup q \in X$.

When specialised to lattices they both coincide with the standard notion. The first of them will be used in the next section.

4.3 Representation Theorems

We have argued that the traces are an acceptable candidates for *computations* of asynchronous systems. In this section an attempt is made for a stronger claim that they are the right choice. Namely, we will show that there is a nice correspondence between the asynchronous systems and non-empty prefix-closed trace languages. The relationship can be expressed in the form of coreflection between the corresponding categories. It cuts down to subcategories of forward stable systems on the one hand, and non-empty *ideals* of traces on the other.

4.3.1 Asynchronous Systems

It is easy to see that the set of traces accepted by a transition system has certain simple properties. Later, we will prove that every such set is non-empty, and prefix-closed. But first, let us note that the converse also takes place. Namely, every non-empty and prefix-closed set of traces $L \subseteq A_I^*$ determines an asynchronous system AS L s.t., the set of traces accepted by AS L is L itself. Below, $alph(L) = \{a \in A \mid \exists t, t' \in A_I^*.\, tat' \in L\}$ denotes the *alphabet* of L.

Let AS $L = \langle L, \varepsilon, alph(L), I \cap alph(L) \times alph(L), \rightarrow \rangle$ where

$t \xrightarrow{a} t'$ iff $t' = ta$, for $t, t' \in L$, $a \in alph(L)$.

Let us prove the claims made so far.

Proposition 4.3.1 *(i) If S is an asynchronous system then L_S is a non-empty and prefix-closed language of traces.*

(ii) If L is a non-empty and prefix-closed language of traces then $S_L =$ AS L is an asynchronous system, and $L_{S_L} = L$.

Proof. (i) Since $c_S^* \xrightarrow{\varepsilon} c_S^*$ it follows that $[\varepsilon] \in L_S \neq \emptyset$.

Next, let $t \preceq t' \in L_S$. Then $c_S^* \xrightarrow{v'} c$ in S for some $v' \in t'$, $c \in C_S$. If $t' = tt''$, $t = [v]$, and $t'' = [v'']$ then $v' =_I vv''$. From Proposition 4.2.1 it follows that $c_S^* \xrightarrow{vv''} c$ in S, and hence $c_S^* \xrightarrow{v} d \xrightarrow{v''} c$ for some $d \in C_S$. Therefore, $t \in L_S$.

(ii) First, let us note that $\langle L, \varepsilon, alph(L), \rightarrow \rangle$ is a transition system. The determinancy of $\rightarrow$ is obvious. Also the other condition is satisfied, by the definition of $alph(L)$. The condition 4.1.1(i) is also satisfied, for if $t \xrightarrow{a} t' \xrightarrow{b} t''$ and $a\ I\ b$ then $t'' = t'b = tab = tba$. Since L is prefix-closed we obtain $tb \in L$, and hence $t \xrightarrow{b} tb \xrightarrow{a} tba = t''$, as required.

The last property follows easily, since by induction on the length of $v \in alph(L)^*$ one shows that:

$\varepsilon \xrightarrow{v} t$ in AS L iff $v \in t$. □

It is possible to give a more categorical account of the situation. Let us consider the category of trace languages defined as follows.

Definition 4.3.2 **TLan** has as objects, referred to as *trace languages*, triples of the form $\mathrm{L} = \langle L, A, I \rangle$ where:

A is a set,

$I \subseteq A \times A$ is an independency on A,

$L \subseteq A_I^*$ is non-empty prefix-closed set of traces s.t., $alph(L) = A$.

A *morphism* $f: \langle L, A, I \rangle \rightarrow \langle L', A', I' \rangle$ in **TLan** is a partial function $f: A \rightharpoonup A'$ such that:

(i) $f_A(a)\downarrow \wedge f_A(b)\downarrow \wedge\ a\ I_S\ b$ implies $f_A(a)\ I_T\ f_A(b)$, for $a, b \in A$,

(ii) $f^!(L) \subseteq L'$.

Identities, and composition of morphisms are inherited from **Pfn**.

Condition 4.3.2(i) is sufficient, by Proposition 4.2.2, but also necessary for $f^\natural$ to be well-defined homomorphism of monoids $f^\natural: A_I^* \to A_{I'}'^*$. Otherwise, it should be evident that the above definition gives a category, and we state this fact without a proof.

Fact 4.3.3 **TLan** *is a category.*

It is more interesting to notice that the two constructions we have discussed are functorial. Moreover, they give rise to a coreflection between **AS**, and **TLan**.

Let TLAN: **AS** $\to$ **TLan** be given by:

TLAN $S = \mathrm{L}_S = \langle L_S, A_S, I_S \rangle$, on objects, and

TLAN $f = f_A$, on morphisms.

Similarly, the construction AS is extended to AS: **TLan** $\to$ **AS** by:

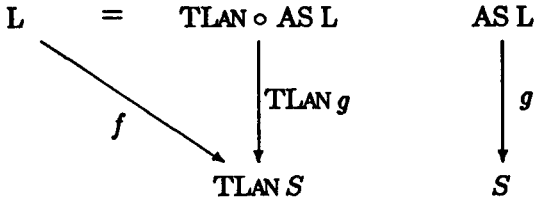
AS $f = \langle f^\natural, f \rangle$.

Theorem 4.3.4 TLAN *and* AS *are functors. Moreover,* TLAN $\circ$ AS $= id_{\mathbf{TLan}}$, *and* TLAN *is a right adjoint to* AS. *Hence,* **TLan** *is a coreflective subcategory of* **AS**.

Proof. It should be evident that AS is a functor. As far as TLAN is concerned it follows from Proposition 4.2.2.

It is also obvious that TLAN $\circ$ AS $= id_{\mathbf{TLan}}$.

Let us establish the adjunction between the functors. Let $L \in$ **TLan**, $S \in$ **AS** and $f: L \to$ TLAN S in **TLan**. We want to show that there exists a unique morphism g in **AS** s.t., the following diagram commutes:



Uniqueness. Let us suppose that such a morphism exists. We show that it is uniquely determined by f.

First, let us note that TLAN $g = f$ implies $g_A = f$. As for the second component of g it follows from g being a morphism that it satisfies the following:

$$g_C(\varepsilon) = c_S^*, \quad \text{and}$$

$$g_C(ta) = \begin{cases} g_C(t) & g_A(a)\uparrow \iff f(a)\uparrow \\ c & f(a)\downarrow \wedge\, g_C(t) \xrightarrow{f(a)} c \quad \text{in } S \end{cases}.$$

Thus, g is uniquely determined by f, if it exists.

Existence. Let $g = \langle g_C, f \rangle$ where g_C is defined as above. It is easy to see that $f^{\natural}(L) \subseteq L_S$ implies: $c = g_C(t)$ iff $c_S^* \stackrel{f^{\natural}(t)}{\longrightarrow} c$ in S. Therefore, g_C is well defined, by Proposition 4.2.1.

Almost trivially, it satisfies the conditions of a morphism in **AS**. □

Remark. There is also another method of constructing an asynchronous system that accepts a given non-empty and prefix-closed trace language $\mathrm{L} = \langle L, A, I \rangle \in \mathbf{TLan}$.

Let $\mathcal{L} = \mathcal{L}(L)$ be the least family of non-empty prefix-closed trace languages that contains L and is closed under ***derivation*** relation, i.e., the least one that satisfies:

(i) $L \in \mathcal{L}$, and

(ii) $a \in L' \in \mathcal{L}$ implies $L'/a = \{t \in A_I^* \mid at \in L'\} \in \mathcal{L}$.

It is easy to verify that $m\mathrm{AS}\,L = \langle \mathcal{L}, L, A, I, \rightsquigarrow \rangle$ where: $L' \stackrel{a}{\rightsquigarrow} L''$ iff $L'' = L'/a$, is an asynchronous system s.t., $L_{m\mathrm{AS}\,L} = L$. This system is in fact ***minimal*** among those which accept L.

To see this it is enough to restrict attention to c^*-***generated*** systems S, i.e., those in which every state c is ***reachable*** from the initial configuration: $c_S^* \stackrel{v}{\longrightarrow} c$, for some $v \in A_S^*$. Then we can define a morphism $\eta_S : S \to m\mathrm{AS}\,L_S$ by:

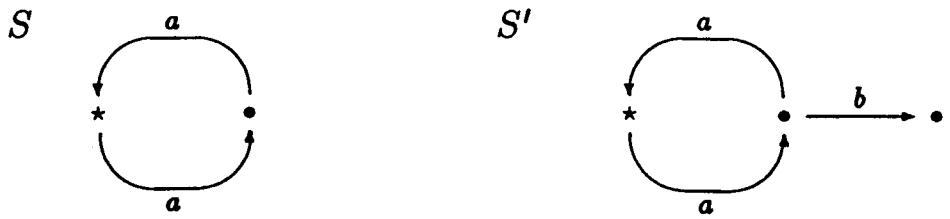
$$\eta_C(c) = L_c, \quad \text{for } c \in C_S, \text{ and}$$
$$\eta_A = id_A.$$

where $L_c = \{t \in A_{S_I}^* \mid \exists d \in C_S \,.\, c \stackrel{t}{\longrightarrow} d\}$ is the set of traces acceptable in S from c.

The above procedure shows, by analogy with theory of finite automata, that each asynchronous system can be ***minimised.*** All one has to do is to remove configurations unreachable from the initial one (together with actions that lost all their origins), and then apply the construction presented.

Thus, apparently, all the ingredients for establishing a reflection are here. All but one. It appears the the procedure of minimisation is <u>not</u> functorial. More specifically, it cannot be made functorial so that we would obtain a reflection.

Let us consider the following (sequential) asynchronous systems.



Intuitively, S is a ***subsystem*** of S', i.e., the pair of inclusions is a morphism. The system S' is already reduced, while S not. This is the cause of problems.

One can, however, restrict the class of morphisms considered so that the construction becomes functorial, and we obtain a reflection. The author decided to stop the discussion here as it would lead too far from the main subject of discourse.

4.3.2 Forward Stable Asynchronous Systems

Our intention is to use non-empty ideals of traces to characterise forward stable systems. To this end a category **iTLan** is defined as the full subcategory of **TLan** with objects whose first components are non-empty ideals of traces.

First, let us relate **iTLan** and **TLan** more closely. We show that the inclusion functor **iTLan** $\hookrightarrow$ **TLan** has a left adjoint.

Theorem 4.3.5 **iTLan** *is a reflective subcategory of* **TLan**.

Proof. For $\mathrm{L} = \langle L, A, I\rangle \in$ **TLan** let $\widehat{\mathrm{L}} = \langle \widehat{L}, A, I\rangle$ where

$$\widehat{L} = \{t \in A_I^* \mid \exists X \in L\,.\, t = \bigsqcup X\}.$$

Obviously, $L \subseteq \widehat{L}$. From Lemma 4.2.12 it follows that $L = \widehat{L}$, for L an ideal.

We show that $\widehat{L}$ is an ideal. Let $p, q \in \widehat{L}$ and $p \sqcup q\!\downarrow$ in A_I^*. Then $p = \bigsqcup X$, and $q = \bigsqcup Y$ for some finite $X, Y \subseteq L$. Thus, $p \sqcup q = \bigsqcup(X \cup Y) \in \widehat{L}$ as a consequence of Lemma 4.2.12.

Let $p \preceq q = \bigsqcup X \in \widehat{L}$. Then X finite, by Lemma 4.2.12. Lemma 4.2.13 and assumption that L is prefix-closed imply $p \in \widehat{L}$.

Next, let us verify that the above operation is functorial. Let $f\colon \mathrm{L} \to \mathrm{L}'$ be a morphism in **TLan**.

Let us put $\widehat{f} = f$. The only thing that needs verification is $f^{\natural}(\widehat{L}) \subseteq \widehat{L}'$. But this follows from $f^{\natural}(L) \subseteq L'$ by continuity of $f^{\natural}$, Lemma 4.2.14.

Details of the proof that we have obtained a functor are left to the reader.

Finally, the reflection. Its existence is an immediate consequence of the way the morphism part of the functor was defined, and its easy proof is omitted as well. □

And now let us turn to the characterisation of forward stable asynchronous systems.

Theorem 4.3.6 *(i) If S is a forward stable asynchronous system then L_S is a non-empty ideal of traces.*

(ii) If L is a non-empty ideal of traces then $S_L = \mathrm{AS}\, L$ is a forward stable asynchronous system, and $L_{S_L} = L$.

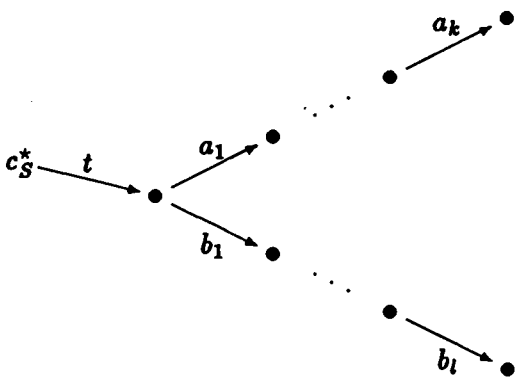
(iii) **iTLan** *is a coreflective subcategory of* **fsAS**.

Proof. (i) We must show that for any two traces with a least upper bound, if they are in L_S then their lub is there as well.

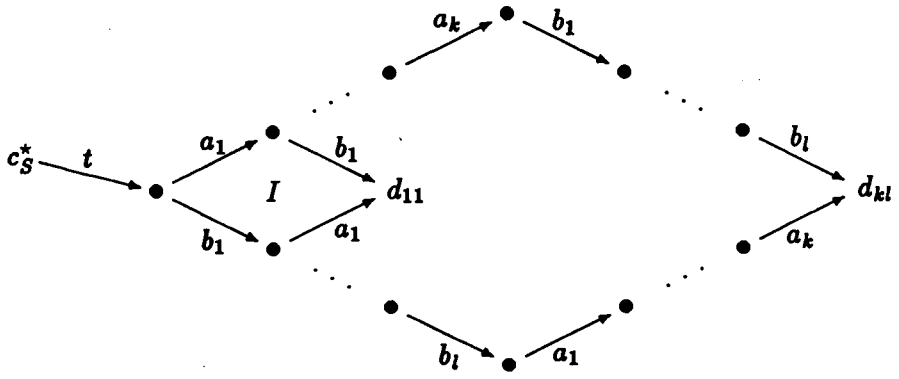
Let $t', t'' \in L_S$. Then $t' \sqcup t'' = t\bar{t}'\bar{t}'' = t\bar{t}''\bar{t}'$ where $t = t' \sqcap t''$, $\bar{t}' = t'/t$ and $\bar{t}'' = t''/t$.

We know also that then $\bar{t}' \; I_S{}^! \; \bar{t}''$, i.e., if $\bar{t}' = a_1 \ldots a_k$, and $\bar{t}'' = b_1 \ldots b_l$ then $a_i \; I_S \; b_j$, for $i = 1 \ldots k$, $j = 1 \ldots l$.

The situation is best described pictorially:

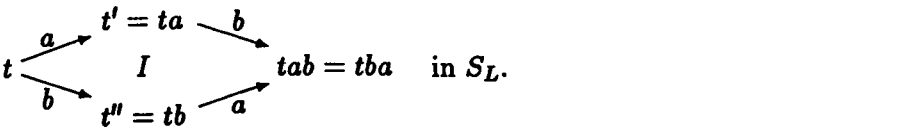


We can also present a pictorial proof:



Existence of configuration d_{11} above with appropriate arrows to it follows from forward stability of S. The result follows by evident induction.

(ii) Let $t \xrightarrow{a} t'$ and $t \xrightarrow{b} t''$ in S_L, i.e., $t' = ta$ and $t'' = tb$, for some $t, t', t'' \in L$. Let $a \; I \; b$ hold as well. Then $t' \sqcup t''$ exists, and $t' \sqcup t'' = tba = tab$. Since L is an ideal we obtain $tab \in L$. Hence,



(iii) Is an immediate consequence of Theorem 4.3.4, and (i) and (ii) above. □

We finish this Chapter with a remark on the role various kinds of ideals of traces play in the studies of asynchronous systems.

We have seen that non-empty lower-sets of traces characterise asynchronous systems, Theorem 4.3.1.

Next, non-empty ideals of traces characterise forward stable asynchronous systems, Theorem 4.3.6.

These results strengthen our conviction that traces are the right candidate for (finite) computations of asynchronous systems. It is then natural to take non-empty upwards directed lower-sets of traces as finite and infinite computations of systems, see e.g., [Mazurkiewicz 84]. From the properties of joins in a poset of traces it follows that such sets coincide with strong ideals of traces.

Chapter 5

Asynchronous Systems vs Other Models

The connection between asynchronous systems and Mazurkiewicz's traces and their languages, was established in the previous Chapter. There are many papers that link traces to other models. *Dependency graphs* are related to traces e.g., in [Aalbersberg&Rozenberg 86]. In [Mazurkiewicz 84] Mazurkiewicz makes a connection between traces on the one hand, and a variety of models including dependency graphs, and also Shields's *string vectors* which he calls *histories*, etc.

Here, the relationship between the *categories* of asynchronous systems and other categories of models for concurrency is established. In his work, see [Winskel 82], [Winskel 84], [Winskel 84a], [Winskel 84b], [Winskel 87] and article in [APN'86/II], Winskel secured links between varieties of models and categories of *event structures*. Therefore, we can build on his work. A connection is made between categories of asynchronous systems and his categories of *prime event structures*.

Some remarks are also made on the relationship between forward stable asynchronous systems, and *safe* Petri nets. In the light of the non-existence of a coreflection between **PSys** and **PNet** we ask whether or not a coreflection exists on this lower level. This problem, to which we do not have a ready answer, is deemed to be the fundamental problem in the development of a unified theory for Petri nets, and other models.

The Chapter is organised as follows.

The first section contains the main results. A coreflection between **AS** and **PES** — the category of prime event structures, is established. To this end we prove that the latter is a coreflective subcategory of **TLan**. This coreflection is shown to specialise to a coreflection between full reflective subcategories of *ideal* prime event structures, and **iTLan**. A notion of a *prime* trace is introduced as an indispensable technical tool.

In the second section the definition of safe nets is quoted, and the category **Safe** of safe nets defined. It is shown that every safe net gives rise to a forward stable asynchronous system via a functor FSAS:**Safe** $\rightarrow$ **fsAS**. The main remaining problem in relating the variety of models for concurrency is given the form of a question of whether the functor

FSAS is a left-inverse right adjoint.

The section, and the last Chapter of the Part I, ends with a diagram representing the various connections.

5.1 Prime Event Structures

Event structures are a natural model in which to express such ideas as *causal dependency*, and *conflict*. They grew up from the work of Nielsen, Plotkin and Winskel, [Nielsen et al 81]. Later, Winskel developed considerably the idea and the theory of event structures. He applied them to give denotational semantics to *CCS*-like languages [Winskel 82]. Certain category of event structures was found closely related to the work by Berry and others, [Berry et al 85], on the problem of full abstraction for PCF, see e.g., Winskel's article in [APN'86/II]. The relationship between event structures and domain theory was realised already in [Nielsen et al 81], but was further developed by Winskel, [Winskel 83], see also [APN'86/II].

For other connections, and more details the reader is strongly advised to consult Winskel's survey paper in [APN'86/II]. From now on, unless otherwise stated, all references to [APN'86/II] below apply to this paper.

In the meantime, the categories considered grew larger. For the sake of this presentation we take the initial idea, essentially from [Nielsen et al 81].

5.1.1 Category of Prime Event Structures

We quote Winskel's definitions and results with changes confined to the differences in the notation.

Definition 5.1.1 ([APN'86/II]) A *prime event structure* E, abbr.: *pes*, is a triple $\mathrm{E} = \langle E, Con, \leq \rangle$ where

E is a set of *events*,

$Con \subseteq \mathcal{F}in\, E$ is family of *consistent* finite subsets of E, and

$\leq$ is a partial order of *causal dependency* on E.

The above data are subject to the following restrictions:

(i) $\{e' \in E \mid e' \leq e\} \in \mathcal{F}in\, E$, for $e \in E$,

(ii) $\{e\} \in Con$, for $e \in E$,

(iii) $Y \subseteq X \in Con$ implies $Y \in Con$, for $Y, X \in \mathcal{F}in\, E$,

(iv) $X \in Con \wedge \exists e' \in X . e' \leq e$ implies $X \cup \{e\} \in Con$, for $e \in E$, $X \in \mathcal{F}in\, E$.

We shall use the natural conventions when referring to the items building a pes E: it will have E as the set of events, and Con_{E}, $\leq_{\mathrm{E}}$, as the consistency predicate, and the causal dependency relation, respectively.

The terminology suggests intended meaning and applications. Let E be a prime event structure.

Notation $(X]$ is used for the lower-set $\{e' \in E \mid \exists e \in X . e' \leq e\}$ *generated* by X, and $(e]$ for $(\{e\}]$. The first clause of the Definition above says that $(e]$ is finite. Posets with this property are called *finitary*. In Chapter 4 it was shown that the posets of traces are finitary. This, as we shall see is not a coincidence.

A set of events is called *consistent* iff every all its finite subset are in Con_{E}. Consistent lower-sets of events are called *states* or *configurations*. As a consequence of 5.1.1(iii) consistent sets form a lower-set in $\mathcal{P} E$ ordered by inclusion.

We will be interested only in finite states of E. Their set is denoted $State_{\mathrm{E}}$. There is an intimate relationship between states and consistent subsets of events.

Fact 5.1.2 $X \in Con_{\mathrm{E}}$ *iff* $(X] \in State_{\mathrm{E}}$, *for* $X \in \mathcal{F}in\, E$.

Proof. "if" Follows by 5.1.1(iii).

"only if" By induction from 5.1.1(iv) in the presence of 5.1.1(i). □

Next, we quote the definition of morphisms between pes-s.

Definition 5.1.3 ([APN'86/II]) A *morphism* f from E to E' where E, E' are pes-s, is a partial function $f: E \rightharpoonup E'$ s.t., for all $e, e' \in E$, $X \in \mathcal{F}in\, E$:

(i) $f(e)\downarrow$ implies $(f(e)] \subseteq f(e]$,

(ii) $X \in Con_{\mathrm{E}}$ implies $f(X) \in Con_{\mathrm{E}'}$,

(iii) $\{e, e'\} \in Con_{\mathrm{E}} \wedge f(e)\downarrow \wedge f(e) = f(e')$ implies $e = e'$.

Composition and *identity* morphisms are inherited from **Pfn**.

From [APN'86/II] we quote without proof the following result.

Fact 5.1.4 *Prime event structures and their morphisms form a category, denoted* **PES**.

Special kind of prime event structures will be of interest later.

Definition 5.1.5 A prime event structure E is called *ideal* iff for any $X \in \mathcal{F}in\, E$

$X \in Con_{\mathrm{E}}$ iff $\forall e, e' \in X . \{e, e'\} \in Con_{\mathrm{E}}$.

Full subcategory of **PES** with ideal event structures is denoted by **iPES**.

In other words an ideal prime event structure is one in which the consistency predicate is determined on a pairwise basis. Therefore, ideal prime event structures can be represented with a help of a binary *conflict* relation on events: $e \#_{\mathrm{E}} e'$ iff $\{e, e'\} \notin Con_{\mathrm{E}}$. It can be shown that the triple $\mathrm{E}^{\#} = \langle E, \#_{\mathrm{E}}, \leq_{\mathrm{E}} \rangle$ enjoys the following property:

$e \#_{\mathrm{E}} e' \leq_{\mathrm{E}} e''$ implies $e \#_{\mathrm{E}} e''$, for $e, e', e'' \in E$.

And conversely, any such triple $\mathrm{E} = \langle E, \#, \leq \rangle$ in which $\langle E, \leq \rangle$ is a finitary poset, and which has the above property determines an ideal prime event structure $\mathrm{E}^C = \langle E, Con_\#, \leq \rangle$ where $Con_\# = \{X \in \mathcal{F}in\, E \mid \forall e, e' \in X \,.\, \neg\, e \# e'\}$. Moreover, $(\mathrm{E}^\#)^C = \mathrm{E}$ and $(\mathrm{E}^C)^\# = \mathrm{E}$.

Reader familiar with [Nielsen et al 81] will recognise that ideal prime event structures are a finitary version of what was referred to as event structures there. In [APN'86/II] Winskel uses notation **P** for our category of prime event structures **PES**, and $\mathbf{P}^\#$ for **iPES**.

We leave the above claims for the reader to prove as an exercise, and quote without proof from [APN'86/II] the following result.

Proposition 5.1.6 **iPES** *is a reflective subcategory of* **PES**.

5.1.2 From Prime Event Structures to Trace Languages ...

Our ultimate goal is to show that every prime event structure can be seen as an asynchronous system, and that every asynchronous system can be *unfolded* into a prime event structure. We also want to show the connection to be a coreflection. Then the unfolding of an asynchronous system built from a pes E coincides with E (up to isomorphism).

Actually, we will use trace languages as an intermediary between the categories **PES** and **AS**. If we show that the former is (equivalent to) a coreflective subcategory of **TLan** then it will follow by Theorem 4.3.4 that it is (equivalent to) a coreflective subcategory of **AS**, since coreflections compose.

Let $\mathrm{E} = \langle E, Con, \leq \rangle$ be a prime event structure — fixed for the sake of this paragraph. Let I_E be an independency on E defined by:

$$e\, I_\mathrm{E}\, e' \quad \text{iff} \quad \{e, e'\} \in Con \;\wedge\; e \not\leq e' \;\wedge\; e' \not\leq e, \qquad \text{for } e, e' \in E.$$

As usual $=_{I_\mathrm{E}}$ stands for the congruence on E^* determined by the independency I_E.

Let $A \subseteq E$ be a set of events of E and $v \in E^*$ its linear ordering, i.e., all and only elements of A appear in v without repetitions. By convention $v = \varepsilon$ is the unique linear ordering of $A = \emptyset$. We are interested in a special class of linear orderings of sets of events.

Definition 5.1.7 A linear ordering $v = e_1 \ldots e_n$, $n \in \omega$, of a set $A \subseteq E$ of events is called *admissible* iff $\{e_1, \ldots, e_i\} = (\{e_1, \ldots, e_i\}]$, for $i = 1, \ldots, n$.

Basic properties of admissible linear orderings and their connection to the independency I_E are the subject of the following Lemma.

Lemma 5.1.8 *(i) ε is an admissible linear ordering of $\emptyset$.*

(ii) If a set of events A has an admissible linear ordering then it is a lower-set.

(iii) Let $A \subseteq B$ be finite lower-sets of events in E. Then every admissible linear ordering v of A has an extension to an admissible linear ordering w of B. Hence, every lower-set of events has an admissible linear ordering.

(iv) Let v, w be two admissible linear orderings of $X \in State_\mathrm{E}$. Then $v =_{I_\mathrm{E}} w$.

(v) Let $A \subseteq E$ be a set of events of E, *and let $v \in E^*$ be its admissible linear ordering. Then $w \in E^*$ and $v =_{I_E} w$ implies w is an admissible linear ordering of A.*

Proof. (i) and (ii) are trivial.

(iii) By induction on the cardinality of $B - A$. If $B - A = \emptyset$ then $v = w$ has required properties. Otherwise, let $e \in B - A$ be minimal in $B - A$. Then $w = ve$ is admissible, and $A \cup \{e\}$ is a lower-set, so we can invoke induction.

As for the second claim it follows from the above by (i).

(iv) By induction on the cardinality of X. If $X = \emptyset$ then $v = \varepsilon = w$, so $v =_{I_E} w$. Otherwise, $v = v'e$, $w = w'e'$, for some $e, e' \in E$, $v', w' \in E^*$.

If $e = e'$ then $v' =_{I_E} w'$ by induction, since v', w' are admissible. So, $v =_{I_E} w$.

If $e \neq e'$ then $X - \{e, e'\}$ is a lower-set, and so there is by (iii) its admissible linear ordering u. Obviously, uee' and $ue'e$ are admissible linear orderings of X, and we have $ue'e -_{I_E} uee'$. Moreover, $v'e =_{I_E} ue'e$ and $w'e' =_{I_E} uee'$ by the previous argument. The result follows.

(v) Let v be an admissible linear ordering of A, and let $v -_{I_E} w$. Then for some $e, e' \in E$, $v', w'' \in E^*$ we have $v = v'ee'w'$, $w = v'e'ew'$ and $e\ I_E\ e'$. But then neither $e \leq e'$ nor $e' \leq e$. Hence, w is admissible as well.

The result follows by induction on k where $v -^k_{I_E} w$. □

As a result we can extend the notion of an admissible linear ordering onto traces in E^*_I, and say that $p \in E^*_I$ is one of them iff every, equivalently any, of its elements is an admissible linear ordering.

The reader may have noticed that only in the fourth clause above it was required that the set of events involved was consistent.

A trace language $\mathrm{L_E} = \langle L_{\mathrm{E}}, E, I_{\mathrm{E}} \rangle$ *generated* by E is given by:

$L_{\mathrm{E}} = \{[v] \in E^*_{I_{\mathrm{E}}} \mid v \text{ is an admissible linear ordering of a consistent set}\}$.

The above definition gives not only a non-empty prefix-closed trace language, but is functorial as well. Let TL: **PES** $\to$ **TLan** be defined by:

$\mathrm{TL\,E} = \mathrm{L_E}$, and

$\mathrm{TL}\, f = f$, i.e., we take the **TLan**-morphism determined by f.

We obtain an expected result.

Proposition 5.1.9 TL: **PES** $\to$ **TLan** *defined above is a functor. Moreover, it cuts down to a functor* TL: **iPES** $\to$ **iTLan**.

Proof. First, $\mathrm{TL\,E}$ is a non-empty trace language, since obviously $\varepsilon \in L_{\mathrm{E}}$. It is also prefix-closed as a consequence of Lemma 5.1.8. For let $t \preceq t' \in L_{\mathrm{E}}$. Then $vv' \in t'$ for some $v, v' \in E^*$ s.t., $v \in t$. From 5.1.8(v) it follows that vv' is admissible, but clearly, admissibility is inherited by prefixes. The result now follows from 5.1.1(iii).

Next, let $f: \mathrm{E} \to \mathrm{E}'$ be a morphism of prime event structures. First, we show that f preserves the independency relation. If $f(e)\downarrow$, $f(e')\downarrow$ and $e\ I_{\mathrm{E}}\ e'$ then $\{e, e'\} \in Con_{\mathrm{E}}$, and neither $e \leq_{\mathrm{E}} e'$ nor $e' \leq_{\mathrm{E}} e$.

We show that $f(e)\ I_{\mathrm{E}'}\ f(e')$.

From 5.1.3(ii) it follows that $\{f(e), f(e')\} \in Con_{\mathrm{E}'}$.

Now, we prove that $f(e) \not\leq_{\mathrm{E}'} f(e')$. Let us assume to the contrary. From 5.1.3(i) it follows that $f(e) = f(e'')$ for some $e'' \in (e']$. From 5.1.1(iv), and then 5.1.1(iii) it follows that $\{e, e''\} \in Con_{\mathrm{E}}$, so that $e = e''$, by 5.1.3(iii). Thus, $e \leq_{\mathrm{E}} e'$. Contradiction.

The proof of $f(e) \not\leq_{\mathrm{E}'} f(e')$ is symmetric.

It remains to show that $f^{\natural}(L_{\mathrm{E}}) \subseteq L_{\mathrm{E}'}$. If $t \in L_{\mathrm{E}}$ then $t = [v]$ for some admissible linear ordering $v = e_1 \ldots e_n$ of a set $A = \{e_1, \ldots, e_n\} \in Con_{\mathrm{E}}$. All we need to know is that $f^*(v)$ is an admissible linear ordering of a set $A' \in Con_{\mathrm{E}'}$.

Let $A' = f(A)$. Then $A' \in Con_{\mathrm{E}'}$, by 5.1.3(ii), and also $f^*(v)$ is a linear ordering of A' by 5.1.3(iii). Now, let $A_i = \{e_1, \ldots, e_i\}$, $A'_i = f(A_i)$, for $i = 0, \ldots, n$. The result will follow if we show that $A'_i = (A'_i]$, for $i = 0, \ldots, n$. It is obvious that $A'_i \subseteq (A'_i]$, so let us concentrate on the inverse inclusion. Let $e' \in (A'_i]$. Then $e' \leq_{\mathrm{E}'} f(e_k)$ and $f(e_k)\downarrow$, for some $k \leq i$. Hence, $e' \in (f(e_k)] \subseteq f(e_k]$, by 5.1.3(i), i.e., $e' = f(e)$, for some $e \in (e_k]$. But $e \in (e_k] \subseteq (\{e_1, \ldots, e_i\}] = \{e_1, \ldots, e_i\} = A_i$, since v admissible. And so, $e' \in f(A_i) = A'_i$, as required.

Finally, let E be an ideal pes, and let $t, t' \in L_{\mathrm{E}}$ be such that $t \sqcup t'$ exists in $E^*_{I_{\mathrm{E}}}$. Then $t \sqcup t' = pqr$ where $p = t \sqcap t'$, $q = t/p$, $r = t'/p$ and $q\ I^{\natural}\ r$, Lemma 4.2.9. By the definition of L_{E}, and from the Lemma 5.1.8 it follows that if $u \in p$, $v \in q$ and $w \in r$ then uv, uw and u are admissible linear orderings of certain consistent lower-sets B, B' and $A = B \cap B'$, respectively. Moreover, v is a linear ordering of $B - A$, and w is a linear ordering of $B' - A$.

From $q\ I_{\mathrm{E}}^{\natural}\ r$ it follows that for any $e \in B - A$, $e' \in B' - A$ we have $e\ I_{\mathrm{E}}\ e'$, and hence $\{e, e'\} \in Con_{\mathrm{E}}$. Thus, for all $e \in B$, $e' \in B'$: $\{e, e'\} \in Con_{\mathrm{E}}$. Therefore, $B \cup B' \in Con_{\mathrm{E}}$, since E is ideal. It is easy to verify now that uvw is an admissible linear ordering of $B \cup B'$. So, $t \sqcup t' \in L_{\mathrm{E}}$. □

5.1.3 ... and Back

There is also an inverse construction of a prime event structure from a given non-empty and prefix-close trace language. First, however, we need to introduce some new concepts related to trace languages.

Let $\mathrm{L} = \langle L, A, I \rangle$ be in **TLan**.

Definition 5.1.10 A trace $t \in A_I^*$ is *prime* iff $\varepsilon \neq t$, and for all $a, b \in A$, $p, q \in A_I^*$ if $pa = t = qb$ then $a = b$.

As a consequence of the definition each prime trace has a unique decomposition $t = \underline{\underline{t}}\overline{\overline{t}}$, where $\overline{\overline{t}} \in A$. A prime trace t with $\overline{\overline{t}} = a$ is often referred to as an *a-prime*. The set of all prime traces in a set of traces L is denoted $pr\, L$. Notation $pr\, t$ is used to denote $pr(t]$.

Remark. The concept of a completely prime (prime) element of a poset is well-established. If $\mathbf{P} = \langle P, \leq \rangle$ is a poset then $p \in P$ is called *completely prime (prime)* iff for every (finite) $X \subseteq P$ it is the case that $p \leq \bigsqcup X$ implies $p \leq x$, for some $x \in X$.

Since in $\langle A_I^*, \preceq \rangle$ only finite sets admit lub-s, and then they are determined by binary lub-s of their elements, it follows that the two concepts reduce to a notion of a 2-prime. A *2-prime* is an element of P s.t., $p \leq r \sqcup s$ implies $p \leq r$ or $p \leq s$.

The reader should encounter no problems in showing that a trace is prime according to Definition 5.1.10 iff it is 2-prime, and hence completely prime.

Prime traces have the property of being the building bricks of all traces, as the following result shows. It says that in $\langle A_I^*, \preceq \rangle$ every trace is a lub of all its prime prefixes.

Lemma 5.1.11 $t = \bigsqcup pr\, t$, *for* $t \in A_I^*$.

Proof. The proof is by induction on the length of t. It is trivial if $t = \epsilon$ or $t \in pr\, A_I^*$.

If t is not prime then $t = sab = sba$, for some $s \in A_I^*$, $a, b \in A$ such that $a\, I\, b$. Then $t = sa \sqcup sb$, by Lemma 4.2.9. Also, $pr\, t = pr\, sa \cup pr\, sb$, and by induction $sa = \bigsqcup pr\, sa$, $sb = \bigsqcup pr\, sb$. Then $t = sa \sqcup sb = \bigsqcup pr\, sa \sqcup \bigsqcup pr\, sb = \bigsqcup (pr\, sa \cup pr\, sb) = \bigsqcup pr\, t$ where the various lub-s exist by Corollary 4.2.6, and the equalities follow from Lemma 4.2.12. □

Now, let $ta \in A_I^*$. Then there exists the largest a-prime among prefixes of ta. To see this let $\phi_a : \{ta \in A_I^* \mid t \in A_I^*\} \to pr\, A_I^*$ be defined by:

$$\phi_a(ta) = \begin{cases} ta & ta \in pr\, A_I^* \\ \phi_a(s) & ta = sb \ \wedge \ a \neq b \end{cases}.$$

Fact 5.1.12 *$\phi_a(ta)$ is the largest a-prime among prefixes of ta. Also, $\phi_a(ta) \prec \phi_a(tat'a)$, for all $a \in A$, $t, t' \in A_I^*$.*

Proof. First, let us see that ϕ_a is well defined. The proof goes by induction, and the only non-trivial case is when $ta = sb = s'b'$ and $a \neq b \neq b' \neq a$. Then $a\, I\, b\, I\, b'\, I\, a$, and $s = rb'$, $s' = rb$, for some $r \in A_I^*$, by Lemma 4.2.4(iii). By induction hypothesis $\phi_a(s)$ and $\phi_a(s')$ are well-defined, and so $\phi_a(s) = \phi_a(r) = \phi_a(s')$, as required.

It is obvious that $\phi_a(ta) \preceq ta$. It is also obvious that $\phi_a(ta)$ is an a-prime.

Let p be an a-prime s.t., $p \preceq ta$. We show by induction on t that $p \preceq \phi_a(ta)$.

Case $ta \in pr\, A_I^*$ includes the basic case of induction, and the result holds trivially.

So assume that $ta = sb$ with $a\, I\, b$. Then $\phi_a(ta) = \phi_a(s)$, but from Levi's Lemma it follows that $p \preceq s$. The result follows by induction.

The second claim follows by induction on the sum of lengths of t and t'.

The case when $tat'a$ is prime contains the basic case of induction, and is trivial.

So, assume $tat'a = rb$ and $a\ I\ b$. Then $tat' = sb$ and $r = sa$, for some $s \in A_I^*$ follows by 4.2.4(iv). By Levi's Lemma we can find $p, p', q, q' \in A_I^*$ s.t., $ta = pp'$, $t' = qq'$, $s = pq$, $b = p'q'$ and $p'q = qp'$.

If $p' = b$ then $q' = \varepsilon$, $ta = pb$ and $t' = q$. Hence, $tat'a = pbqa = paqa = pqba = pqab$. Therefore, $\phi_a(tat'a) = \phi_a(pqa)$, and $\phi_a(ta) = \phi_a(p)$. The result follows by induction since $p = s'a$ for some $s' \in A_I^*$.

If $q' = b$ then $p' = \varepsilon$, $ta = p$ and $t' = qb$. Hence, $tat'a = taqba = taqab$, and so $\phi_a(tat'a) = \phi_a(taqa)$. We can apply induction again. □

Let A, A' have independencies I, I', resp., and let $f: A \rightharpoonup A'$ be independency preserving. We can define partial function $f^\circ: pr\, A_I^* \rightharpoonup pr\, A_{I'}'^*$ as follows:

$$f^\circ(p) = \phi_{f(\overline{p})}(f^\natural(p)), \quad \text{for } p \in pr\, A_I^*.$$

According to our convention $f^\circ(p)\downarrow$ iff $f(\overline{p})\downarrow$, for $p \in pr\, A_I^*$.

We obtain an important technical result.

Lemma 5.1.13 *$pr\, f^\natural(t) = f^\circ(pr\, t)$, for $t \in A_I^*$.*

Proof. "$\supseteq$" This part is easy. If $q = f^\circ(p)$ and $p \in pr\, t$ then, by Lemma 4.2.14

$$\phi_{f(\overline{p})}(f^\natural(p)) \preceq f^\natural(p) \preceq f^\natural(t).$$

"$\subseteq$" First, let us show that for any $q \in pr\, A_I^*$

$$q \preceq f^\natural t \quad \text{implies} \quad f(a) = \overline{q} \ \wedge\ ra \preceq t \ \wedge\ q = \phi_{\overline{q}}(f^\natural(ra)), \quad \text{for some } r \in A_I^*,\ a \in A.$$

The proof goes by induction on t. If $t = \varepsilon$ the result holds trivially, while $t = sb$ and $f(b)\uparrow$ follows easily by induction. Thus, let us assume $t = sb \ \wedge\ f(b)\downarrow$. Then $qp = f^\natural(s)f(b)$, for some $p \in A_I'^*$.

By Levi's Lemma we can find $t', t'', t''', t'''' \in A_I'^*$ s.t., $q = t't''$, $p = t'''t''''$, $f^\natural(s) = t't'''$, $f(b) = t''t''''$ and $t''\ I'^\natural\ t'''$.

There are two cases to consider. If $t'''' = f(b)$ then $t'' = \varepsilon$, and so $q = t' \preceq t't''' = f^\natural(s)$. The result follows by induction.

If $t'' = f(b)$ then $t'''' = \varepsilon$, and so $q = t'f(b)$. Thus, $\overline{q} = f(b)$. From $f(b) = t''\ I'^\natural\ t''' = p$ it follows easily that $\phi_{\overline{q}}(f^\natural(t)) = \phi_{\overline{q}}(q) = q$. Hence, we can take $a = b$ and $r = s$.

Finally, it can be shown that the pair r, a as above can be taken so that ra is prime. For if $ra = sb$ with $a\ I\ b$ then $ra = r'ab = r'ba$, for some $r' \in A_I^*$. Since f preserves independency it follows that either $f(b)\uparrow$, or $f(a)\ I'\ f(b)$. In the first case the size of the problem reduces right away. In the second we have $q = \phi_{f(a)}f^\natural(r'ab) = \phi_{f(a)}f^\natural(r'a)$, and so the result follows by induction as well. □

We are ready to define construction inverse to TL.

Let $\mathrm{L} = \langle L, A, I \rangle$ be a trace language in **TLan**.

Let $\mathrm{PES\,L} = \mathrm{E_L} = \langle pr L, Con_{\mathrm{L}}, \preceq \rangle$ where

$$Con_{\mathrm{L}} = \{X \subseteq pr\, L \mid \exists t \in L \forall p \in X \,.\, p \preceq t\}.$$

In other words we take the set of prime traces of L ordered by prefix relation. Consistent sets of primes are those with an upper bound in L.

Let $f: \mathrm{L} \to \mathrm{L}'$ be a morphism of trace languages.

PES $f: E_{\mathrm{L}} \rightharpoonup E_{\mathrm{L}'}$ is defined by

$$\mathrm{PES}\, f = f^{\circ}.$$

Proposition 5.1.14 PES *defined above is a functor* $\mathrm{PES}: \mathbf{TLan} \to \mathbf{PES}$. *Moreover, it cuts down to* $\mathrm{PES}: \mathbf{iTLan} \to \mathbf{iPES}$.

Proof. Let $\mathrm{L} = \langle L, A, I \rangle$ be a trace language in **TLan**. Then $\mathrm{E_L}$ is a prime event structure. It satisfies 5.1.1(i), since the poset $\langle A_I^*, \preceq \rangle$ is finitary. The other three conditions in the Definition 5.1.1 are satisfied trivially.

Next, let $\mathrm{L} = \langle L, A, I \rangle$ and $\mathrm{L}' = \langle L', A', I' \rangle$ be trace languages, and let $f: \mathrm{L} \to \mathrm{L}'$ be a morphism in **TLan**. Then PES f is a morphism of prime event structures.

It satisfies 5.1.3(i). For assume that $f^{\circ}(p)\downarrow$. Then $(f^{\circ}(p)] = pr\, f^{\circ}(p) \subseteq pr\, f^{\natural}(p) = f^{\circ}(pr\, p) = f^{\circ}(p]$, by Lemma 5.1.12, Lemma 5.1.13

Also, it satisfies 5.1.3(ii). This follows from the observation that $t \preceq t'$ implies that $f^{\natural}(t) \preceq f^{\natural}(t')$, since $f^{\natural}(L) \subseteq L'$.

Finally, condition 5.1.3(iii). Let us assume that $t, t' \in pr\, L$, $\{t, t'\} \in Con_{\mathrm{L}}$, PES $f\, t\downarrow$ and PES $f\, t =$ PES $f\, t'$. Then $f(\bar{t})\downarrow$ and $f(\bar{t}) = f(\overline{t'})$. As a result it is not the case that $\bar{\bar{t}}\; I\; \overline{\overline{t'}}$. From Lemma 4.2.9 it follows that $t \sqcup t'$ exists, since $\{t, t'\} \in Con_{\mathrm{L}}$, and that $t \sqcup t' = pqr$ where $p = t \sqcap t'$, $q = t/p$ and $r = t'/p$, with $q\; I^{\natural}\; r$. We need to show that $q = \varepsilon = r$. Assume to the contrary.

Let $q \neq \varepsilon \neq r$. Since a non-empty postfix of an a-prime trace is an a-prime trace, by Cancellation Law, it follows from $q\; I^{\natural}\; r$ that $\overline{\overline{q}}\; I\; \overline{\overline{r}}$ — contradiction.

Let $q = \varepsilon \neq r$. Then $t' = \underline{\underline{t}} t s \overline{\overline{t'}}$, for some $s \in A_I^*$. Thus, PES $f\, t = \phi_{f(\bar{t})}(f^{\natural}(\underline{\underline{t}}) f(\bar{t})) \prec \phi_{f(\bar{t})}(f^{\natural}(\underline{\underline{t}}) f(\bar{t}) f^{\natural}(s) f(\overline{\overline{t'}})) =$ PES $f\, t'$, by Fact 5.1.12. Contradiction.

Assumption $q \neq \varepsilon = r$ leads to contradiction by argument symmetric to the above.

It remains to show that L - ideal implies $\mathrm{E_L}$ -ideal.

Let $X \subseteq pr\, L$ be such a finite set that $p, q \in X$ implies $\{p, q\} \in Con_{\mathrm{L}}$. By definition it means that $p \sqcup q\downarrow$, for $p, q \in X$. From Lemma 4.2.12 it follows that $\bigsqcup X$ exists, and that it can be computed in a pairwise fashion: $\bigsqcup X = (\ldots(p_1 \sqcup p_2) \sqcup \ldots \sqcup p_n)$ where $X = \{p_1, \ldots, p_n\}$. Since L is an ideal it follows by induction that each $(\ldots(p_1 \sqcup p_2) \sqcup \ldots \sqcup p_i)$ is in L, for $i = 2, \ldots, n$. Hence, $\bigsqcup X \in L$, i.e., $X \in Con_{\mathrm{L}}$, by definition. □

5.1.4 Coreflection

Finally, we show that in fact the two functors defined in previous paragraphs are adjoint to each other. More specifically, functor TL is left adjoint to PES, and their composition PES $\circ$ TL is naturally isomorphic to $id_{\mathbf{PES}}$.

Theorem 5.1.15 TL $\dashv$ PES, *and* PES$\circ$TL $\cong id_{\mathbf{PES}}$, *i.e.,* **PES** *is a coreflective subcategory of* **TLan**. *This result cuts down to a coreflection between* **iPES** *and* **iTLan**.

Proof. The proof consists of several stages.

Let E be a pes. By 5.1.8(iii) every finite lower-set of events of E admits an admissible linear ordering. In particular, for $e \in E$, such a linear ordering, v_e say, of $(e]$ determines an e-prime trace. Hence, we can define a function $\eta_{\mathbf{E}}: E \to pr\, L_{\mathbf{E}}$ by

$$\eta_{\mathbf{E}}(e) = [v_e], \quad \text{for } e \in E.$$

It is well-defined by 5.1.8(iv).

Let us see that in fact it is an isomorphism of posets $\langle E, \leq_{\mathbf{E}} \rangle$ and $\langle pr\, L_{\mathbf{E}}, \preceq \rangle$.

From 5.1.8(iii) it follows that $\eta_{\mathbf{E}}$ is order-preserving.

Evidently, it is order-reflecting: if p is a linear ordering of A while q of B, $A, B \subseteq E$, then $p \preceq q$ means $A \subseteq B$.

Finally, if $p \in pr\, L_{\mathbf{E}}$ then $p = \eta_{\mathbf{E}}(\overline{p})$, so $\eta_{\mathbf{E}}$ is surjective. These three properties imply that $\eta_{\mathbf{E}}$ is an isomorphism of posets.

Next, we show that $\eta_{\mathbf{E}}: \mathbf{E} \to \mathbf{E}_{\mathbf{L}_{\mathbf{E}}}$ is an isomorphism of prime event structures, i.e., that both: $\eta_{\mathbf{E}}$ and $\eta_{\mathbf{E}}^{-1}$ satisfy conditions of the Definition 5.1.3.

As for 5.1.3(i) it follows from the fact that $\eta_{\mathbf{E}}$ is an isomorphism of the underlying posets. The same applies to 5.1.3(iii) — more precisely it follows from the fact that $\eta_{\mathbf{E}}$ is a bijective function.

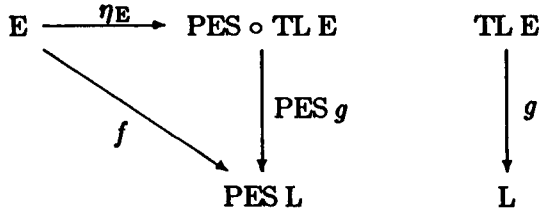
They satisfy also 5.1.3(ii), since we can show that the following holds:

$$X \in Con_{\mathbf{E}} \quad \text{iff} \quad \eta_{\mathbf{E}}(X) \in Con_{\mathbf{L}_{\mathbf{E}}}, \text{ for } X \in \mathcal{F}in\, E.$$

Indeed, if $X \in Con_{\mathbf{E}}$ then $(X] \in Con_{\mathbf{E}}$, by Fact 5.1.2. As a consistent lower-set it has an admissible linear ordering $t \in L_{\mathbf{E}}$, by 5.1.8(iii) and definition of $L_{\mathbf{E}}$. Also by 5.1.8(iii) it follows that $\eta_{\mathbf{E}}(e) \preceq t$, for $e \in X$. Hence, $\eta_{\mathbf{E}}(X) \in Con_{\mathbf{L}_{\mathbf{E}}}$, by definition of $Con_{\mathbf{L}_{\mathbf{E}}}$.

The converse has a simpler proof.

The intention is to use $\eta_{\mathbf{E}}$, for $\mathbf{E} \in \mathbf{PES}$, as the unit of coreflection. Thus, we need to show that for any $\mathbf{E} = \langle E, Com, \leq \rangle \in \mathbf{PES}$, $\mathbf{L} = \langle L, A, I \rangle \in \mathbf{TLan}$, and $f: \mathbf{E} \to \mathrm{PES}\,\mathbf{L}$ the diagram:



commutes for a unique g.

Uniqueness. Let us assume that there exists a morphism $g: L_E \to L$ that makes the diagram above commute. Then it will be uniquely determined by f.

Since η_E is total it follows that $f(e)\downarrow$ iff $g(e)\downarrow$, for $e \in E$. Next, if $f(e)\downarrow$ then $f(e) = g^\diamond \circ \eta_E(e) = \phi_{g(e)}(g^\sharp(\eta(e)))$. Hence, $g(e) = \overline{\overline{f(e)}}$.

Existence. Let us define g by: $g(e) = \overline{\overline{f(e)}}$, for $e \in E$. We will show first that g preserves independency, next that $f = g^\diamond \circ \eta_E$ in **Pfn**, and finally that $g^\sharp(L_E) \subseteq L$. Therefore that g is the required morphism.

g preserves independency. Let us assume that $g(e)\downarrow$, $g(e')\downarrow$ and $e\ I_E\ e'$. Then $f(e)\downarrow$, $f(e)\downarrow$ and $\{e, e'\} \in Con_E$, and neither $e \leq e'$ nor $e' \leq e$ — by definitions of the items involved.

Then $\{f(e), f(e')\} \in Con_L$, since f is a morphism of event structures. Hence, $f(e) \sqcup f(e')$ exists, by definition of Con_L and Lemma 4.2.9, and equals $f(e) \sqcup f(e') = pqr$, where $p = f(e) \sqcap f(e')$, $q = f(e)/p$ and $r = f(e')/p$, with $q\ I^\sharp\ r$. Since both traces $f(e)$, $f(e')$ are prime it follows that so are their proper postfixes. Thus, either $g(e) = \overline{\overline{f(e)}} = \overline{\overline{q}}\ I\ \overline{\overline{r}} = \overline{\overline{f(e')}} = g(e')$ — as required, or one of them is a prefix of the other.

We show that the second cannot take place. For assume to the contrary, that e.g., $f(e) \preceq f(e')$. From 5.1.3(i) it then follows that $f(e) = f(e'')$ for some $e'' \leq e'$. Then $\{e, e''\} \in Con_E$, by 5.1.1(iv), 5.1.1(iii). So, $e = e''$ follows from 5.1.3(iii). This contradicts assumption $e\ I_E\ e'$.

$f = g^\diamond \circ \eta_E$. Let $e_0 \in E$. The proof follows by induction on cardinality of $(e_0]$.

If $(e_0] = \{e_0\}$ then the result evidently holds.

In general we need to take into account only the case where $f(e_0)\downarrow$, and so $g(e_0)\downarrow$ as well. Let $p_0 = \underline{\underline{\eta_E(e_0)}}$, $q_0 = \underline{\underline{f(e_0)}}$, and $a = \overline{\overline{f(e_0)}}$. Obviously, $\overline{\overline{\eta_E(e_0)}} = e_0$. Then $q_0 = \bigsqcup pr\ q_0$, by Lemma 5.1.11. For any prime prefix q of q_0 it follows from 5.1.3(i) that $q = f(e)$, for some $e < e_0$. Let $Y = \{e < e_0 \mid f(e) \prec f(e_0)\}$.

We show that $e < e_0$ and $e \notin Y$ implies that $f(e)\uparrow$ or $g(e)\ I\ g(e_0)$. For assume that $f(e)\downarrow$. Then, as once already, it can be shown that $\{f(e), f(e_0)\} \in Con_L$, since f is a morphism of prime event structures, and so $f(e) \sqcup f(e_0)$ exists, and $f(e) \sqcup f(e_0) = pqr$, where $p = f(e) \sqcap f(e_0)$, $q = f(e)/p$ and $r = f(e_0)/p$, with $q\ I^\sharp\ r$. A priori there are four cases to consider. If $q \neq \varepsilon \neq r$ we are done since $g(e) = \overline{\overline{f(e)}} = \overline{\overline{q}}\ I\ \overline{\overline{r}} = \overline{\overline{f(e_0)}} = g(e_0)$. It remains to show that none of the other takes place.

Case $q = \varepsilon \neq r$ is excluded by assumption $e \notin Y$. So, let $r = \varepsilon$. Then $f(e_0) \preceq f(e)$ so that $e_0 \leq e$ follows in the established already way from assumption that f is a morphism. This contradicts assumption $e < e_0$.

Since g preserves independency it follows that:

$$\begin{array}{lcll} q_0 & = & \bigsqcup\{f(e) \mid e \in Y\} & \text{Lemma 5.1.11} \\ & = & \bigsqcup\{g^\diamond \circ \eta_{\mathbf{E}}(e) \mid e \in Y\} & \text{by induction} \\ & \preceq & \bigsqcup\{g^\diamond \circ \eta_{\mathbf{E}}(e) \mid e < e_0\} & \\ & \preceq & \bigsqcup\{g^\sharp \circ \eta_{\mathbf{E}}(e) \mid e < e_0\} & \text{Lemma 5.1.12} \\ & = & g^\sharp(\bigsqcup\{\eta_{\mathbf{E}}(e) \mid e < e_0\}) & \text{Lemma 4.2.14} \\ & = & g^\sharp(\underline{\underline{\eta_{\mathbf{E}}(e_0)}}) = g^\sharp(p_0). & \end{array}$$

So, $q_0 s = g^\sharp(p_0)$, for some s. But, for every $e < e_0$, if $g(e)\downarrow$ and $g(e)$ occurs in s then $g(e)\ I\ g(e_0) = a$, as shown above. Hence, $as = sa$, and $f(e_0) = q_0 a \preceq q_0 as = q_0 sa = g^\sharp(\eta_{\mathbf{E}}(e_0))$. Also, it is evident that $q_0 a$ is the largest a-prime of $g^\sharp(\eta_{\mathbf{E}}(e_0))$, so $f(e_0) = g^\diamond \circ \eta_{\mathbf{E}}(e_0)$.

$g^\sharp(L_{\mathbf{E}}) \subseteq L$. Let $t \in L_{\mathbf{E}}$, and let B be the lower-set of events that occur in t. Then

$$\begin{array}{lcll} g^\sharp(t) & = & \bigsqcup pr\, g^\sharp(t) & \text{Lemma 5.1.11} \\ & = & \bigsqcup g^\diamond(pr\, t) & \text{Lemma 5.1.13} \\ & = & \bigsqcup g^\diamond(\{\eta_{\mathbf{E}}(e) \mid e \in B\}) & \\ & = & \bigsqcup\{f(e) \mid e \in B\}. & \end{array}$$

Now, $B \in Con_{\mathbf{E}}$ implies $f(B) \in Con_{\mathbf{L}}$, since f a morphism. Thus, $f(B)$ has an upper bound $s \in L$. Therefore, $g^\sharp(t) \preceq s \in L$, and $g^\sharp(t) \in L$ follows since L is prefix-closed.

Clearly, the coreflection cuts down to the subcategories since so do both functors involved. □

5.2 General Picture

5.2.1 Safe Petri Nets

It was already remarked that most of the research into the theory of Petri nets concentrates on the study of special subcategories. Again most of them are included in the category of safe nets. The main advantage of working with safe nets is that one can work with sets rather than with μ-sets. In what follows by a (finite) set we mean a μ-set of the form $\sum\{\hat{x} \mid x \in X\}$, for some finite set X. Also, the usual set theoretic operations are used to denote corresponding operations on multisets, e.g., $\chi \cup \xi$ stands for the maximum of χ and ξ, while $\chi \cap \xi$ denotes their minimum, etc.

Definition 5.2.1 ([Winskel 87]) A Petri net N is *safe* iff all μ–sets of the form: ${}^\bullet(e)$, $(e)^\bullet$, M, for $e \in E$, $M \in \mathrm{Mark}_N$, are non-empty sets.

The full subcategory of **PNet** with safe nets will be denoted by **Safe**.

Our first observation is that safe Petri nets determine forward stable asynchronous systems in a natural way. Let $N = \langle B, E, {}^\bullet(_), (_)^\bullet, M_0\rangle$ be a safe Petri net.

Define FSAS $N = \langle \mathrm{Mark}_N, M_0, r\,E, I_N, \rightarrow\rangle$ where

$r\,E = \{e \in E \mid \exists M, M' \in \mathrm{Mark}_N \,.\, M \xrightarrow{\widehat{e}} M'\}$,

$M \xrightarrow{e} M'$ iff $M \xrightarrow{\widehat{e}} M'$ in N, and

$e\ I_N\ e'$ iff $({}^\bullet(e) \cup (e)^\bullet) \cap ({}^\bullet(e') \cup (e')^\bullet) = \emptyset$ and $M \xrightarrow{\widehat{e} + \widehat{e'}} M'$,

for some $M, M' \in \mathrm{Mark}_N$.

The above construction extends to a functor if we put FSAS $f = \langle g_C, f_E\rangle$, for a morphism $f: N \rightarrow N'$ of safe nets where $g_C: \mathrm{Mark}_N \rightarrow \mathrm{Mark}_{N'}$ is a restriction of μf_B.

Lemma 5.2.2 FSAS *is a functor* FSAS: **Safe** $\rightarrow$ **fsAS**.

Proof. It should be obvious that the 4-tuple $\langle \mathrm{Mark}_N, M_0, E, \rightarrow\rangle$ is a transition system.

It is obvious that I_N is irreflexive. Moreover, FSAS N is a forward stable asynchronous system. To see that it satisfies 4.1.1(i) let us assume that $M \xrightarrow{e} M' \xrightarrow{e'} M''$ in FSAS N, where $e\ I_N\ e'$. Then $({}^\bullet(e)) \cap ({}^\bullet(e') \cup (e')^\bullet) = \emptyset$ implies that ${}^\bullet(e') \leq M$, and the result follows easily.

The proof that the condition of forward stability holds as well is similar.

Next, let $f: N \rightarrow N'$ be a morphism between safe nets N, N'. Then FSAS f preserves the initial states (markings), and from Corollary 3.1.5 it follows that μf_B sends reachable markings to reachable markings. FSAS f satisfies condition 4.1.2(i) as a consequence of f satisfying 3.1.2(i). Let us see that it also satisfies 4.1.2(ii), i.e., that f_E preserves independency.

Let $e, e' \in E$ be such that $f(e)\downarrow$, $f(e')\downarrow$ and $e\ I_N\ e'$.

Then $({}^\bullet(e) \cup (e)^\bullet) \cap ({}^\bullet(e') \cup (e')^\bullet) = \emptyset$, and $M \xrightarrow{\widehat{e} + \widehat{e'}} M'$, for some $M, M' \in \mathrm{Mark}_N$. From Proposition 3.1.4 it follows that $\mu f_B(M) \xrightarrow{\widehat{f(e)} + \widehat{f(e')}} \mu f_B(M')$. So it remains to show that $({}^\bullet(f_E(e)) \cup (f_E(e))^\bullet) \cap ({}^\bullet(f_E(e')) \cup (f_E(e'))^\bullet) = \emptyset$.

To see that $(f_E(e))^\bullet \cap {}^\bullet(f_E(e')) = \emptyset$ it is enough to realise that $(e)^\bullet + {}^\bullet(e') \leq M'''$, for a reachable marking $M''' = M - {}^\bullet(e) + (e)^\bullet$. Then $(f_E(e))^\bullet + {}^\bullet(f_E(e')) \leq \mu f_B(M''')$, and so $(f_E(e))^\bullet + {}^\bullet(f_E(e'))$ a set. Hence, $(f_E(e))^\bullet \cap {}^\bullet(f_E(e')) = \emptyset$, as required.

The three remaining cases follow by symmetric arguments. □

It may seem that a simpler definition of independency I_N between the transitions of the safe net N would do. The proof above demonstrates that the assumption that two transitions to be independent must somewhere happen concurrently was needed to show that the construction is functorial. It is easy to come up with examples showing that the assumption is also necessary.

We note also that the functor extended by one of the functors embedding **AS** into **PSys** gives the same result as going directly via SYS. All inclusion functors involved in the process are tacitly omitted. Also omitted is an easy proof.

Proposition 5.2.3 $F \circ \text{FSAS} = \text{SYS}$.

Functor FSAS composed with others yields descriptions of a safe net by means of an ideal trace language, ideal prime event structure, and so on.

5.2.2 Fundamental Problem of Net Theory

We have seen how nicely a variety of categories of models for concurrency is interrelated via (co)reflections. There is only one exception in this rosy picture. We have seen that the general category of Petri nets does not contain **PSys** as a (co)reflective subcategory, and vice versa.

On the other hand we do believe that Petri nets are most important as the most concrete model of asynchronous behaviours. Therefore our quest for a left-inverse adjoint to SYS has the meaning of the quest for ***universal realisation*** of a behaviour by means of Petri nets.

To pursue this idea we have restricted attention to the study of nicer categories: asynchronous and forward stable asynchronous systems on one hand, and safe nets on the other. The same question can be asked now.

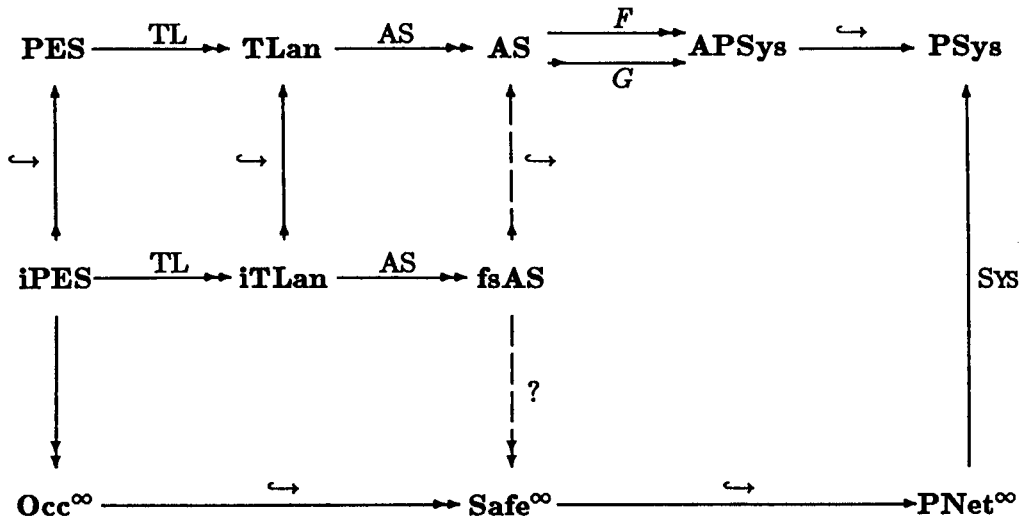
Does FSAS: **Safe** $\rightarrow$ **fsAS** *have a right-inverse left adjoint?.*

The author has spent a lot of time trying to answer the question positively — so far without success.

In the framework of this presentation it should be noticed that the expected coreflection can take place only between the subcategories of ***finite*** forward stable asynchronous systems, and ***finite*** safe nets. By finite we mean here objects which have finite representations: sets of states/places and actions/events are finite. The reason is that the general result would have to extend the only known result in this direction. Namely, Winskel's construction of a coreflection between ideal prime event structures and ***occurrence nets*** — the class of nets often used to give the semantics of safe nets in this coreflective subcategory of nets, see [APN'86/II]. The point now is that in his construction non-finitary initial markings appear naturally. Hence, one has to consider a wider class of morphisms than that admitted here.

This is not to say that such a construction would be worthless if carried only for finite objects. After all some finite safe nets have infinite behaviours, e.g., corresponding trace languages.

Among other benefits we would be able to transform the semantics of, essentially, labelled forward stable asynchronous systems for *CCS*-like languages considered in Part II to the realisation in safe nets. At least, for the sublanguage without recursion.



Above:

——→ — denotes a coreflection, and

↛ — denotes a reflection.

Figure 5.1: General picture

5.2.3 General Picture

We end this Chapter, and the first Part of the thesis with a diagram illustrating the various categories, and links between them established so far.

The (co)reflections are depicted in the direction of the embedding functors.

Above, **Occ** stands for mentioned above category of occurrence nets. The superscript "∞" attached to categories of nets indicates that we are interested in more general categories of nets with non-finitary markings, for instance as in [APN'86/II].

The broken arrows denote suspected, but not yet established connections.

The one between **fsAS** and **AS** is expected to be a reflection.

Although the second is suggested as a coreflection between **fsAS** and **Safe**$^\infty$ the author would be quite happy to see the real connection being established between certain subcategories of the two. For instance, each safe net is mapped via FSAS to a forward stable system in which all configurations are reachable along a path from the initial configuration. Thus, one should restrict attention to a subcategory of forward stable systems that have this property. In Part II such subcategory of labelled skeletal systems will be studied, and proved coreflective in **LSS**. Note that the functor AS:**TLan** $\rightarrow$ **AS**, and also its restriction to **iTLan** and **fsAS**, has this property.

Similar changes can be expected on the other side.

Part II

Non-interleaving Semantics of CCS–like Languages

Chapter 6

Recursive Languages: Their Syntax and Semantics

In this Chapter basic notions and apparatus used to give denotational semantics to recursive languages is reviewed. The Chapter is organised as follows.

In the first section we introduce *signatures*, and other basic notions used to give the syntax of recursive languages.

The second section surveys quickly the standard way in which recursive languages are given denotational semantics in *ω-complete pointed posets*, abbr.: ω-cppo.

This in turn is generalised in the last section to Category Theory. We trade ω-cppo-s for ω-categories and continuous functions for ω-functors. In this part we relay heavily on the pioneering work of M. Smyth and D. Lehmann.

This Chapter is introductory in character. Only the last section contains material that the reader may be unable to find elsewhere, but even that cannot be counted as novel. On the contrary, we see it an evident and simple application of more general theory developed in [Lehmann 76], [Lehmann&Smyth 81] and [Smyth&Plotkin 82] to the particular area of our interest. The reader is reminded that we deal with a very simple kind of languages: only constructors and recursion. The theory, only small fragments of which we use, is capable of dealing with recursively defined types, polymorphism, etc.

There is one important omission in this short exposition. We do not expand on the treatment of equational theories. We lack motivation for conducting it as, at least at the moment, there is no ready application in sight. Later, we shall give a semantics to a recursive language $\mathcal{L}$. Unfortunately, we have no nice, i.e. finite, equational axiom system to characterise identifications of terms of $\mathcal{L}$ induced by this denotational semantics. Moreover, it seems doubtful that such a system exists in general.

6.1 Syntax

Languages, programming languages included, are defined as collections of *well-formed formulas*, or *terms* built from pre-specified set of *constructors*.

The idea is formalised with the help of the notion of *signature*, called also *type*, [Grätzer 79], or *ranked alphabet*, compare [Abramsky 83].

We restrict attention to the case of so called *one-sorted* signatures. This reflects the fact that all languages considered later have only one syntactic category. The generalisation of most of the results in the first two sections of this Chapter to the many-sorted case can be found e.g., in [ADJ 77].

Definition 6.1.1 A *signature* Σ consists of a set, denoted by Σ as well, together with a function $\mathrm{arity}_\Sigma: \Sigma \to \omega$

Elements of Σ are often referred to as *constructors*, or *function symbols*. We use σ to denote a generic element of Σ. The subscript in arity_Σ will be omitted when the signature is known from the context.

Function symbols with arity 0, i.e., those σ that satisfy $\mathrm{arity}(\sigma) = 0$, are called *constant symbols*. Those with arity 1 are *unary*, those with 2 *binary*, etc. We collect all the symbols of Σ with arity n in the subset $\Sigma_n = \{\sigma \in \Sigma \mid \mathrm{arity}(\sigma) = n\}$ of Σ.

Given a signature Σ we can formalise the notion of a well-formed Σ-formula, or Σ-term.

Definition 6.1.2 The set T_Σ of *ground terms over* Σ is defined inductively as the least one that satisfies:

$$(\Diamond) \qquad \sigma \in \Sigma_n \wedge t_1, \ldots, t_n \in \mathsf{T}_\Sigma \quad \text{implies} \quad \sigma(t_1, \ldots, t_n) \in \mathsf{T}_\Sigma.$$

The condition $(\Diamond)$ asserts that all constant symbols belong to T_Σ. Therefore all symbols which are obtained from the constant symbols by prefixing with a non-constant symbol belong to T_Σ, and so on. If Σ has no constant symbols at all then $\mathsf{T}_\Sigma = \emptyset$.

Instead of writing: $\sigma() \in \mathsf{T}_\Sigma$, for a constant σ, we simply write: $\sigma \in \mathsf{T}_\Sigma$. Also, our general convention of using function symbols in infix notation is extended to the constructors of the signatures considered.

We write $t = t'$ to indicate that the terms t and t' are syntactically identical.

Let x, y, z: Var be a countable set.

Given a signature Σ one often considers a derived signature: $\Sigma(\mathsf{Var}) = \Sigma \cup \mathsf{Var}$, with $\mathrm{arity}_{\Sigma(\mathsf{Var})}(\sigma) = \mathrm{arity}_\Sigma(\sigma)$, for $\sigma \in \Sigma$, and $\mathrm{arity}_{\Sigma(\mathsf{Var})}(x) = 0$, for $x \in \mathsf{Var}$. That is we enreach the signature Σ with the elements of Var as additional constant symbols, and call them *variables*.

Since $\Sigma(\mathsf{Var})$ is a signature it determines the set $\mathsf{T}_{\Sigma(\mathsf{Var})}$ of ground terms over $\Sigma(\mathsf{Var})$. To stress the different role played by constants from Var we will, however, use different notation for it. We shall write $\mathsf{T}_\Sigma(\mathsf{Var})$ instead, and refer to its elements as *(syntactically) finite open terms Σ-terms*.

The next step is to introduce *recursion* to the language. We want to define certain objects as *solutions* of simple *recursive definitions*. The variables introduced previously help to formalise these ideas. By a recursive definition we mean equations like:

$$x \Leftarrow a(x), \tag{6.1}$$

or *simultaneous* equations like:

$$\begin{aligned} x &\Leftarrow a(x) + y, \\ y &\Leftarrow b(x) + c(y). \end{aligned} \tag{6.2}$$

Above $x, y \in \mathsf{Var}$ were variables, a, b, c denoted unary symbols from some signature Σ, while $_ + _$ was a binary infix symbol from Σ.

It is technically convenient to introduce syntactic objects to denote the hypothetical solutions of recursive equations like (6.1) or (6.2). Then we can manipulate them in much the same manner as usual terms. In the case of the above examples we use

$$\mathbf{rec}\, x.\, a(x),$$

to denote the solution of the equation (6.1), and

$$\mathbf{rec}\, x.\, (a(x) + \mathbf{rec}\, y.\, [b(x) + c(y)])$$

$$\mathbf{rec}\, y.\, [b(\mathbf{rec}\, x.\, a(x) + y) + c(y)]$$

to denote the x-th and y-th parts of the solution of the equation (6.2), respectively.

Formally, we extend the syntax as follows.

Definition 6.1.3 The set $\mathsf{Rec}_\Sigma(\mathsf{Var})$ of *recursive terms over* Σ is the set of terms over the signature $\Sigma(\mathsf{Var}) \cup \{\mathbf{rec}\, x.\, _ \mid x \in \mathsf{Var}\}$, where $\mathbf{rec}\, x.\, _$, for $x \in \mathsf{Var}$, are new unary constructors.

Later, the notion of a *solution* of recursive definitions, as in the examples above, is formalised. We will not, however, expand on the formal relationship between the recursive definitions and the terms that denote their solutions. This seems far beyond our needs and besides there are numerous classical texts like [Bekič 69], or [ADJ 77], available on the subject.

We close this section by introducing the notions of free and bound variables formally.

In term $\mathbf{rec}\, x.\, t$ the operator $\mathbf{rec}\, x.\, _$ *binds* occurrences of a variable x in t. Define functions $\mathsf{FV}, \mathsf{BV}\colon \mathsf{Rec}_\Sigma(\mathsf{Var}) \to \mathsf{Var}$ that compute sets of free and bound variables in a term. The definition is by structural induction. Below, $\sigma \in \Sigma_n$, $t, t_1, \ldots, t_n \in \mathsf{Rec}_\Sigma(\mathsf{Var})$, and $x \in \mathsf{Var}$.

$$\begin{aligned} \mathsf{FV}(x) &= \{x\} \\ \mathsf{FV}(\sigma(t_1, \ldots, t_n)) &= \mathsf{FV}(t_1) \cup \ldots \cup \mathsf{FV}(t_n) \\ \mathsf{FV}(\mathbf{rec}\, x.\, t) &= \mathsf{FV}(t) - \{x\} \end{aligned}$$

$$\begin{aligned} \mathsf{BV}(x) &= \emptyset \\ \mathsf{BV}(\sigma(t_1, \ldots, t_n)) &= \mathsf{BV}(t_1) \cup \ldots \cup \mathsf{BV}(t_n) \\ \mathsf{BV}(\mathbf{rec}\, x.\, t) &= \mathsf{BV}(t) \cup \{x\}. \end{aligned}$$

Variable x is called *free*, resp., *bound*, in term t iff $x \in \mathsf{FV}(t)$, resp., $x \in \mathsf{BV}(t)$. Terms without free variables are called *closed*. The set of closed terms is denoted $\mathsf{Rec}_\Sigma(\emptyset)$.

Terms in $\mathsf{Rec}_\Sigma(\mathsf{Var})$ are referred to as *(open) recursive Σ-terms*. Those in $\mathsf{Rec}_\Sigma(\emptyset)$ are obviously *closed recursive Σ-terms*.

The idea of substituting terms for variables is very natural and fruitful. Care must be always taken if one wants to prevent free occurrences of variables being captured as a result of substitution.

There are two methods to achieve it. One, used by many logicians, is to perform substitution naively as long as it is possible. When the danger of capturing arises those bound variables which cause problems are *renamed*. Another one is more uniform since it always renames bound variables. We opt for the second solution. The results quoted follow the general pattern established in [Stoughton ??].

Let us assume that a function $\mathsf{choice}\colon (\mathcal{P}(\mathsf{Var}) - \{\emptyset\}) \to \mathsf{Var}$ is given, such that $\mathsf{choice}(X) \in X$.

The set $\varrho\colon \mathsf{Sub}_\Sigma$ of substitutions is equal $\mathsf{Sub}_\Sigma = \mathsf{Var} \to \mathsf{Rec}_\Sigma(\mathsf{Var})$. The *identity* substitution $\imath$ is defined by $\imath x = x$, for $x \in \mathsf{Var}$. A substitution ϱ is called *closed* iff $\varrho\, x$ is a closed term, for $x \in \mathsf{Var}$. The subscript in Sub_Σ will be omitted when clear from the context.

An auxiliary function $\mathsf{new}\colon \mathsf{Var} \to (\mathsf{Rec}_\Sigma(\mathsf{Var}) \to (\mathsf{Sub} \to (\mathcal{P}\mathsf{Var} - \{\emptyset\})))$ is defined by:

$$\mathsf{new}\, x\, t\, \varrho = \mathsf{Var} - \bigcup\{\mathsf{FV}(\varrho\, z) \mid x \neq z \in \mathsf{FV}(t)\}.$$

Obviously, the set we discharge on the right is finite, so $\mathsf{new}\, x\, t\, \varrho$ is always infinite.

With these tools we can define the simultaneous substitution which always renames bound variables as follows.

Definition 6.1.4 The *simultaneous substitution* $t\,\varrho$ of $\varrho\, x$, for all free occurrences of x in t, is defined by induction on the structure of t.

$$\begin{aligned}
x\,\varrho &= \varrho(x), \qquad \text{for } x \in \mathsf{Var};\\
\sigma(t_1,\ldots,t_n)\,\varrho &= \sigma(t_1\,\varrho,\ldots,t_n\,\varrho),\\
&\qquad \text{for } \sigma \in \Sigma_n, t_1,\ldots,t_n \in \mathsf{Rec}_\Sigma(\mathsf{Var});\\
(\mathsf{rec}\, x.\, t)\varrho &= \mathsf{rec}\, y.\, (t\,\varrho[x \mapsto y]),\\
&\qquad \text{for } x \in \mathsf{Var}, t \in \mathsf{Rec}_\Sigma(\mathsf{Var}),\ \text{where } y = \mathsf{choice}(\mathsf{new}\, x\, t\, \varrho).
\end{aligned}$$

We remind the reader of our convention that for $f\colon X \to Y$, $x \in X$ and $y \in Y$ we denote by $f[x \mapsto y]$ the function defined by:

$$f[x \mapsto y](x') = \begin{cases} y & x = x' \\ f(x') & x \neq x' \end{cases}.$$

Also, for $f, g\colon X \to Y$, $X' \subseteq X$ we use $f =^{X'} g$ to denote: $f(x) = g(x)$, for $x \in X'$.

The *composition* $\varrho_1 \circ \varrho_0$ of substitutions ϱ_0, ϱ_1 is defined by $\quad (\varrho_1 \circ \varrho_0)\, x = (\varrho_0\, x)\, \varrho_1$.

The following results are a simple reformulation of those found in [Stoughton ??].

Lemma 6.1.5 *(i)* $\varrho_0 =^{FV(t)} \varrho_1$ *implies* $t\,\varrho_0 = t\,\varrho_1$;

(ii) $(t\,\varrho_0)\,\varrho_1 = t\,(\varrho_1 \circ \varrho_0)$;

(iii) $(\varrho_2 \circ \varrho_1) \circ \varrho_0 = \varrho_2 \circ (\varrho_1 \circ \varrho_0)$.

We end this section with a remark concerning the so called *α-congruence* of terms. This equivalence relation identifies those terms that differ at most by the names of the bound occurrences of variables.

One of the disadvantages of the first kind of substitutions, that which renames bound variables only when necessary, is that one must work with classes of terms equivalent up to α-congruence. The reason being that the composition of substitutions is not associative: it is associative only up to α-congruence.

As we see in the above Lemma this is not the case for the second kind of simultaneous substitution.

For more on that subject, and the proof of the properties from the Lemma, consult [Stoughton ??].

6.2 A Crash Course in Denotational Semantics

We briefly discuss the standard way in which recursive languages are given denotational semantics. More detailed introduction can be found e.g., in [Stoy 77] and [Gordon 79], while [ADJ 77] and [Guessarian 81] are the right place to look for a more algebraic approach.

Let Σ be a signature. It will stay fixed throughout this and the following sections.

The adjective 'denotational' in the phrase 'denotational semantics' indicates the fact that this is a kind of semantics in which every term of the language is given a *denotation* that explains its meaning. Here, we concentrate attention on the *algebraic semantics*. This is a branch of denotational semantics in which algebraic structures are used to provide the denotations for Σ-terms, and algebraic methods are used to study them.

The section is organised as follows.

First, notion of a Σ-algebra is introduced. This enables us to give the meaning to (syntactically) finite open Σ-terms.

Next, we consider the notion of a Σ- algebra in the category of ω-complete pointed posets.

Finally, we show how the extra structure can be used to give the meaning to recursive terms.

6.2.1 Σ-algebras

We face the following problem.

What kind of structures do we need to interpret (closed) recursive terms over Σ ?

First, the problem contains as a subproblem the question of interpretation of ground terms. There is a well established method to do just that. One uses *Σ-algebras* as domains of interpretation.

Definition 6.2.1 A Σ-*algebra* $\mathcal{A}$ is a pair $\mathcal{A} = \langle A, \Sigma_{\mathcal{A}} \rangle$, where

A is a set called *universe* of $\mathcal{A}$, and

$\Sigma_{\mathcal{A}} = \{\sigma_{\mathcal{A}} : A^n \to A \mid \sigma \in \Sigma \wedge \text{arity}(\sigma) = n\}$ is a set of functions.

It is natural to take as morphisms of Σ-algebras those functions between carriers which preserve their structure.

Definition 6.2.2 Let $\mathcal{A} = \langle A, \Sigma_{\mathcal{A}} \rangle$, $\mathcal{B} = \langle B, \Sigma_{\mathcal{B}} \rangle$ be Σ-algebras. A Σ-*homomorphism* from $\mathcal{A}$ to $\mathcal{B}$, denoted $h : \mathcal{A} \to \mathcal{B}$, is a function $h : A \to B$ such that

$$h(\sigma_{\mathcal{A}}(a_1, \ldots, a_n)) = \sigma_{\mathcal{B}}(h(a_1), \ldots, h(a_n)), \quad \text{for } \sigma \in \Sigma_n,\ a_1, \ldots, a_n \in A.$$

Σ-algebras constitute a category when equipped with Σ-*homomorphisms* as morphisms. It is completely standard to check that the following holds.

Fact 6.2.3 Σ-*algebras with* Σ-*homomorphisms constitute a category, denoted* $\Sigma-\mathbf{alg}(\mathbf{Set})$.

Proof. It is easy to verify that the composition and identities inherited from **Set** make $\Sigma-\mathbf{alg}(\mathbf{Set})$ into a category. □

Remark. The reason for having **Set** mentioned in $\Sigma-\mathbf{alg}(\mathbf{Set})$ is to stress that we are dealing with Σ-algebras in **Set**. Thus, the carriers are sets = objects of **Set**, while constructors of Σ are interpreted as functions = morphisms of **Set**. This observation should highlight the pattern in the subsequent generalisations from **Set**, first to $\omega-\mathbf{CPPO}_{\perp}$, and subsequently to $\omega-\mathbf{Cat}_{\perp}$.

Below we give as an example a construction of a most general, i.e., initial Σ-algebra. Significantly, it is constructed from the syntax of the language under consideration.

Example 6.2.4 There is a universal construction of a Σ-algebra from the syntax. We shall denote it by $\mathcal{T}_\Sigma$. It has:

T_Σ – the set of ground Σ-terms as a universe, and

$\sigma_{\mathcal{T}_\Sigma}(t_1, \ldots, t_n) = \sigma(t_1, \ldots, t_n), \quad \text{for } \sigma \in \Sigma_n,\ t_1, \ldots, t_n, \in T_\Sigma.$

Obviously, $\mathcal{T}_\Sigma$ is a Σ-algebra. ◇

The construction above is indeed universal.

Theorem 6.2.5 $\mathcal{T}_\Sigma$ *is initial in* $\Sigma-\mathbf{alg}(\mathbf{Set})$.

Before the proof of this Theorem let us show how the ground Σ-terms can be interpreted in a Σ-algebra $\mathcal{A} = \langle A, \Sigma_{\mathcal{A}} \rangle$. The interpretation function is a map $\mathcal{A}[\![_]\!] : T_\Sigma \to A$. The definition is inductive, as expected.

$$\mathcal{A}[\![\sigma(t_1, \ldots, t_n)]\!] = \sigma_{\mathcal{A}}(\mathcal{A}[\![t_1]\!], \ldots, \mathcal{A}[\![t_n]\!]), \quad \text{for } \sigma \in \Sigma_n, t_1, \ldots, t_n \in T_\Sigma. \tag{6.3}$$

Its importance is best appreciated in the context of the proof of Theorem 6.2.5.

Proof.[of Theorem 6.2.5] Condition (6.3) says precisely that $\mathcal{A}[\![_]\!]$ is a Σ-homomorphism. Hence, every Σ-homomorphism satisfies it. The uniqueness follows now by structural induction. □

The terms with variables cannot be interpreted, simply, as values in $\mathcal{A}$, since the variables do not have any natural interpretation. It is more fruitful to pursue the idea that they are meaningful only when an *environment* is given in which every variable denotes certain element of $\mathcal{A}$.

Let us introduce a notion of a *$\mathcal{A}$-assignment*, or *$\mathcal{A}$-environment* formally.

Definition 6.2.6 Let $\mathcal{A} = \langle A, \Sigma_{\mathcal{A}} \rangle$ be a Σ-algebra. A *$\mathcal{A}$-environment* ρ is a function: $\rho\colon \mathsf{Var} \to A$.

The set of all $\mathcal{A}$-environments is denoted by $\mathbf{Env}_{\mathcal{A}}$.

Now, the interpreting function for each term should take an $\mathcal{A}$-assignment before coming up with a result, i.e., it should have a type: $\mathcal{A}[\![_]\!]\colon \mathsf{T}_\Sigma(\mathsf{Var}) \to (\mathbf{Env}_{\mathcal{A}} \to \mathcal{A})$. The clauses below can be read as defining an extension of an environment ρ from Var onto $\mathsf{T}_\Sigma(\mathsf{Var})$. The second clause is virtually the same as (6.3). By induction it follows easily that for closed terms t the function $\mathcal{A}[\![t]\!]$ is constant, with the constant determined by (6.3). This should justify the use of the same notation.

$$\begin{aligned} \mathcal{A}[\![x]\!]\rho &= \rho(x), \qquad \text{for } x \in \mathsf{Var}, \rho \in \mathbf{Env}_{\mathcal{A}}; \\ \mathcal{A}[\![\sigma(t_1, \ldots, t_n)]\!]\,\rho &= \sigma_{\mathcal{A}}(\mathcal{A}[\![t_1]\!]\rho, \ldots, \mathcal{A}[\![t_n]\!]\,\rho), \\ &\qquad \text{for } \sigma \in \Sigma_n, t_1, \ldots, t_n \in \mathsf{T}_\Sigma(\mathsf{Var}), \rho \in \mathbf{Env}_{\mathcal{A}}. \end{aligned} \tag{6.4}$$

As previously we find that $\mathcal{A}[\![_]\!]$ is categorically important.

Example 6.2.7 Let $\mathcal{T}_\Sigma(\mathsf{Var})$ be a Σ-algebra with

$\mathsf{T}_\Sigma(\mathsf{Var})$ as universe, and

$\sigma_{\mathcal{T}_\Sigma(\mathsf{Var})}(t_1, \ldots, t_n) = \sigma(t_1, \ldots, t_n)$, for $\sigma \in \Sigma_n$, $t_1, \ldots, t_n, \in \mathsf{T}_\Sigma(\mathsf{Var})$.

Then $\mathcal{T}_\Sigma(\mathsf{Var})$ is a Σ-algebra. ◇

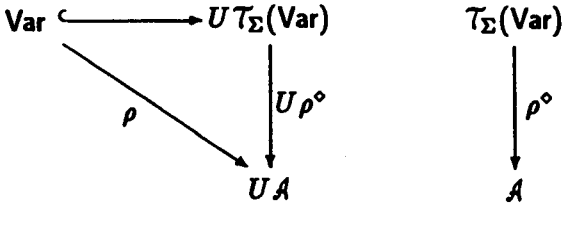
There is an evident forgetful functor $U\colon \Sigma\text{–}\mathbf{alg}(\mathbf{Set}) \to \mathbf{Set}$ given by:

$U(\mathcal{A}) = A$ on objects, and

$U(f\colon \mathcal{A} \to \mathcal{B}) = f$ on morphisms.

With its help we can formulate the universal property of $\mathcal{T}_\Sigma(\mathsf{Var})$.

Theorem 6.2.8 *For all $\mathcal{A}, \rho\colon \mathsf{Var} \to U(\mathcal{A})$ the diagram*



commutes, for a unique Σ-homomorphism $\rho^\diamond$.

This means that $\mathcal{T}_\Sigma$ itself is functorial: $\mathcal{T}_\Sigma$: **Set** $\to \Sigma$**−alg(Set)**, and that $\mathcal{T}_\Sigma \dashv U$, i.e., $\mathcal{T}_\Sigma$ is left-adjoint to U. $\mathcal{T}_\Sigma$(Var) is called sometimes *the free Σ-algebra on* Var. The interpretation function appears again in the proof.

Proof. We show that $\rho^\diamond$ must be defined as follows:

$$\rho^\diamond(t) = \mathcal{A}[\![t]\!]\rho, \qquad \text{for } t \in \mathsf{T}_\Sigma(\mathsf{Var}).$$

Obviously, $\rho^\diamond$ defined above is a Σ-homomorphism, by the second clause in (6.4), and the commutativity of the diagram above is ensured by the first clause in (6.4).

As for the uniqueness, note that any Σ-homomorphism h must satisfy both clauses in (6.4) whenever it makes the diagram commute. Thus, h coincides with $\rho^\diamond$, follows by induction. □

6.2.2 ω-complete Posets, ω-continuous Functions

We extended T_Σ(Var) to Rec_Σ(Var) in order to introduce syntactic names for hypothetical solutions of simple recursive equations. Now we are in the position to formalise these concepts.

A Σ-equation is, in general, a pair of Σ-terms. Here, it is irrelevant that the equations in the recursive definitions we are mostly interested in are of a particularly simple form. We shall use the notation $t \Leftarrow t' \in E$ to indicate that the pair $\langle t, t' \rangle$ is in the set E of equations.

Let $\mathcal{A}$ be a Σ-algebra, and ρ a $\mathcal{A}$ environment.

The equation $t \Leftarrow t'$ is *satisfied* by ρ iff $\mathcal{A}[\![t]\!]\,\rho = \mathcal{A}[\![t']\!]\,\rho$.

Environment ρ *satisfies* set E of equations iff it satisfies all equations in E. Such a ρ is called a *solution* of E in $\mathcal{A}$.

Our immediate problem is to make sure that every recursive definition has a solution. Another one is to find a uniform way of picking up one from the (possibly) many. The following examples demonstrate the problems.

Example 6.2.9 Let $\Sigma = \{a, b, c\}$ with $\mathsf{arity}(a) = \mathsf{arity}(b) = \mathsf{arity}(c) = 1$.

The set of equations $\{x \Leftarrow a(x)\}$ has *no* solutions in $\mathcal{T}_\Sigma$ the initial Σ-algebra of Example 6.2.4. ◇

Example 6.2.10 For any Σ-algebra $\mathcal{A}$ the set of equations $\{x \Leftarrow x\}$ has entire $\mathbf{Env}_\mathcal{A}$ as the set of solutions, i.e., every $\mathcal{A}$-environment ρ satisfies it. ◇

Now, let $\mathcal{A}$ be a Σ-algebra, and let $E = \{x_1 \Leftarrow t_1, \ldots, x_n \Leftarrow t_n\}$ be a set of recursive definitions, where $x_i = x_j$ iff $i = j$, for $i, j \in \{1, \ldots, n\}$. Then E determines a functional $\mathcal{F}_E$: $\mathbf{Env}_\mathcal{A} \to \mathbf{Env}_\mathcal{A}$ defined by:

$$\mathcal{F}_E \rho\, x = \begin{cases} \mathcal{A}[\![t_i]\!]\,\rho & x = x_i \\ \rho(x) & x \notin \{x_1, \ldots, x_n\} \end{cases} \qquad \text{for } x \in \mathsf{Var} \tag{6.5}$$

We can reformulate the problem with the help of the following observation.

Fact 6.2.11 *ρ is a solution of E iff $\rho = \mathcal{F}_E(\rho)$.*

Proof. Immediate from the definitions. □

Therefore we have to ensure that the functionals like (6.5) do have fixed points.

In the late sixties/early seventies D. Scott developed the now standard approach to the problem. We need to add an extra structure to our Σ-algebras, to equip them with partial order relations. The crucial observation is that if a poset $\mathbf{P}$ is sufficiently *complete* then any *continuous* map $f: \mathbf{P} \to \mathbf{P}$ has a *least fixed point* by a version of the Tarski Fixpoint Theorem.

For this idea to work we need a special kind of posets.

Let $\mathbf{P} = \langle P, \sqsubseteq_{\mathbf{P}} \rangle$ be a poset. It is natural to consider posets together with functions between them that preserve the ordering. These, obviously, constitute a category.

We will be interested in *ω-chains*, i.e., sets like $X = \{x_0, x_1, \ldots\} \subseteq P$ that satisfy: $x_n \sqsubseteq_{\mathbf{P}} x_{n+1}$, for $n \in \omega$.

Evidently, every order preserving function maps ω-chains onto ω-chains. We call such a function *continuous* iff it preserves existing least upper bounds of ω-chains.

Notation $f: \mathcal{A} \to_\omega \mathcal{B}$ is used to indicate that f is continuous map between posets $\mathcal{A}$ and $\mathcal{B}$. The set of all continuous maps between $\mathcal{A}$ and $\mathcal{B}$ is denoted $\mathcal{A} \to_\omega \mathcal{B}$.

A *ω-complete pointed poset* $\mathcal{A}$, abbr.: *ω-cppo*, is a triple $\mathcal{A} = \langle A, \sqsubseteq_{\mathcal{A}}, \bot_{\mathcal{A}} \rangle$ where $\langle A, \sqsubseteq_{\mathcal{A}} \rangle$ is a poset which has a least upper bound: $\bigsqcup_{\mathcal{A}} X$, for every ω-chain $X \subseteq A$, and a least element: $\bot_{\mathcal{A}}$.

An order preserving function $f: \mathcal{A} \to \mathcal{B}$ between posets with bottoms: $\bot_{\mathcal{A}}$ and $\bot_{\mathcal{B}}$ is called *strict* if it preserves bottom: $f(\bot_{\mathcal{A}}) = \bot_{\mathcal{B}}$.

Remark. The reader should be aware that there are many other proposals available in the literature. Like *continuous lattices* [Scott 76], *algebraic* ω-cppo-s [Smyth&Plotkin 82] *directed* cppo-s [Guessarian 81], etc. Our choice is motivated by later generalisation. This seems to be the most general category of posets considered in the literature so far.

Fact 6.2.12 *ω-cppo-s with continuous functions as morphisms constitute a category, denoted* $\omega-\mathbf{CPPO}$.

Similarly, ω-cppo-s with strict continuous functions as morphisms constitute a category, denoted $\omega-\mathbf{CPPO}_\bot$.

Proof. Standard. □

We are now ready to state the most important result in the area of denotational semantics. It is a simple reformulation of the well known Tarski Fixpoint Theorem [Tarski 55] for complete lattices.

Let us first introduce the notion of a fixed point formally.

Definition 6.2.13 Let $\mathcal{A} = \langle A, \sqsubseteq_{\mathcal{A}} \rangle$ be a poset and $f: \mathcal{A} \to \mathcal{A}$ a map.

A *prefixed point* of f is any $a \in A$ s.t., $f(a) \sqsubseteq_{\mathcal{A}} a$.

A *fixed point* of f is any $a \in A$ s.t., $f(a) = a$.

And now the already announced Theorem.

Theorem 6.2.14 (Fixpoint Theorem) *Let $\mathcal{A}$ be a ω-cppo. Then every continuous function $f: \mathcal{A} \to_{\omega} \mathcal{A}$ has a least fixed point: $\Upsilon(f) = \bigsqcup_{\mathcal{A}} \{\bot_{\mathcal{A}}, f(\bot_{\mathcal{A}}), f(f(\bot_{\mathcal{A}})), \ldots\}$.*

Proof. Let $X = \{\bot_{\mathcal{A}}, f(\bot_{\mathcal{A}}), f^2(\bot_{\mathcal{A}}), \ldots\}$. We show that $\bigsqcup_{\mathcal{A}} X$ is the fixed point, and the least prefixed point of f. Then the result follows.

Since f is order preserving and $\bot_{\mathcal{A}} \sqsubseteq_{\mathcal{A}} f(\bot_{\mathcal{A}})$ then we have by induction on n:

$$f^n(\bot_{\mathcal{A}}) \sqsubseteq_{\mathcal{A}} f^{n+1}(\bot_{\mathcal{A}}) \qquad \text{for } n \in \omega.$$

This means that X is a ω-chain, and has a least upper bound $\bigsqcup X$. By continuity of f we have

$$\begin{aligned} f(\bigsqcup X) &= \bigsqcup f(X) \\ &= \bigsqcup (X - \{\bot_{\mathcal{A}}\}) \\ &= \bigsqcup X, \end{aligned}$$

so that $\bigsqcup X$ is a fixed point of f.

Now, let us see that $\bigsqcup X$ is the least prefixed point of f. Let a be any prefixed point of f, i.e., $f(a) \sqsubseteq_{\mathcal{A}} a$. Then $f^n(a) \sqsubseteq_{\mathcal{A}} a$, for $n \in \omega$, follows by induction since f is order preserving. From $\bot_{\mathcal{A}} \sqsubseteq_{\mathcal{A}} a$ by the same argument it follows that $f^n(\bot_{\mathcal{A}}) \sqsubseteq_{\mathcal{A}} f^n(a)$. Hence, $f^n(\bot_{\mathcal{A}}) \sqsubseteq_{\mathcal{A}} a$, for $n \in \omega$, i.e., a is an upper bound of X. Therefore, $\bigsqcup X \sqsubseteq_{\mathcal{A}} a$, as required. □

We must answer some general questions concerning the above categories.

What constructions create ω-cppo-s from the given ones?

Which (strict) continuous morphisms arise naturally?

Here we investigate only those of our immediate interest. The reader is advised to consult [Smyth&Plotkin 82] for more. We let $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$ denote ω-cppo-s in the remaining part of this section.

First examples come from the observation that the functor $U: \omega\text{-}\mathbf{CPPO} \to \mathbf{Set}$ that 'forgets' about the bottom and the ordering in ω-cppo-s, creates finite products in both: $\omega\text{-}\mathbf{CPPO}$ and $\omega\text{-}\mathbf{CPPO}_{\perp}$.

1 the *terminal* object in both: $\omega\text{-}\mathbf{CPPO}$ and $\omega\text{-}\mathbf{CPPO}_{\perp}$. There is only one way to make $\mathbf{1} = \{\star\}$, i.e., the terminal object in **Set**, into a poset. This poset has of course bottom, and it is a ω-cppo for equally obvious reasons. We note in passing that **1** is in $\omega\text{-}\mathbf{CPPO}_{\perp}$ not only terminal, but initial as well, so a *zero* object.

$\mathcal{A} \times \mathcal{B}$ the *product* of $\mathcal{A}$ and $\mathcal{B}$ is a ω-cppo with

$A \times B$ as the universe;

$\sqsubseteq_{\mathcal{A}\times\mathcal{B}}$ defined coordinatewise:

$$\langle a, b\rangle \sqsubseteq_{\mathcal{A}\times\mathcal{B}} \langle a', b'\rangle \quad \text{iff} \quad a \sqsubseteq_{\mathcal{A}} a' \wedge b \sqsubseteq_{\mathcal{B}} b';$$

$\bot_{\mathcal{A}\times\mathcal{B}} = \langle \bot_{\mathcal{A}}, \bot_{\mathcal{B}}\rangle$.

$\pi_{\mathcal{A}}: \mathcal{A} \times \mathcal{B} \to_\omega \mathcal{A}, \pi_{\mathcal{B}}: \mathcal{A} \times \mathcal{B} \to_\omega \mathcal{B}$ the *projection* functions are strict and continuous.

The above gives us the following well known statement.

Fact 6.2.15 *The product of ω-cppo-s defined above is a categorical product in both:* $\omega-\mathbf{CPPO}$ *and* $\omega-\mathbf{CPPO}_\bot$.

$f \times g: \mathcal{A} \times C \to_\omega \mathcal{B} \times \mathcal{D}$ is the continuous *product* of $f: \mathcal{A} \to_\omega \mathcal{B}$ and $g: C \to_\omega \mathcal{D}$. It is defined using categorical properties of the product.

$\mathcal{A}^n$ the *n-th power* of ω-cppo-s is defined as usual:

$\mathcal{A}^0 = \mathbf{1}$, i.e., the terminal object in $\omega-\mathbf{CPPO}$.

$\mathcal{A}^{n+1} = \mathcal{A}^n \times \mathcal{A}$.

$id_{\mathcal{A}}: \mathcal{A} \to_\omega \mathcal{A}$ the *identity* function is strict and continuous.

$K_b: \mathcal{A} \to_\omega \mathcal{B}$ the *constant* function given by: $K_b = \lambda a{:}A.\, b$ is continuous. It is strict iff $b = \bot_{\mathcal{B}}$.

$\Delta^n_{\mathcal{B}}: \mathcal{B} \to_\omega \mathcal{B}^n$ is a strict and continuous *diagonal* map given by

$$\Delta^n_{\mathcal{B}}(b) = (b, \ldots, b).$$

$\mathcal{B} \to_\omega C$ the set of continuous maps from $\mathcal{B}$ to C. It becomes a ω-cppo if we put:

$f \sqsubseteq_{\mathcal{B}\to_\omega C} g$ iff $f(b) \sqsubseteq_C g(b)$, for $b \in B$;

$\bot_{\mathcal{B}\to_\omega C} = \lambda b{:}B.\, \bot_C$.

$X \to \mathcal{A}$ More generally, we can make the set of *all* functions from a set X to the carrier A of ω-cppo $\mathcal{A}$ into a ω-cppo. Let

$f \sqsubseteq_{X\to\mathcal{A}} g$ iff $f(x) \sqsubseteq_{\mathcal{A}} g(x)$, for $x \in X$;

$\bot_{X\to\mathcal{A}} = \lambda x{:}X.\, \bot_{\mathcal{A}}$.

Let us note at this point that an alternative way to define the n-th power of a ω-cppo $\mathcal{A}$ would be to take $n \to \mathcal{A}$ instead where, according to our convention, $n = \{i \in \omega \mid i < n\}$.

$\Delta^X_{\mathcal{A}}: \mathcal{A} \to_\omega (X \to \mathcal{A})$ We have a more general strict and continuous *diagonal* map. It is given by:

$$\Delta^X_{\mathcal{A}}(a) = \lambda x{:}X.\, a.$$

$\Upsilon_{\mathcal{B}}: (\mathcal{B} \to_\omega \mathcal{B}) \to_\omega \mathcal{B}$ the *least fixed point operator* is defined by:

$\Upsilon_{\mathcal{B}}(f) =$ the least fixed point of f

It is the essence of the approach that every continuous map $f: \mathcal{B} \to_\omega \mathcal{B}$ has, by Theorem 6.2.14, a least fixed point. Moreover, the operation of taking least fixed points is itself continuous, and strict as well.

$curry: (\mathcal{A} \times \mathcal{B} \to_\omega \mathcal{C}) \to (\mathcal{A} \to_\omega (\mathcal{B} \to_\omega \mathcal{C}))$, is the well-known isomorphism of ω-cppo-s. It is given by:

$$curry(f) = \lambda a{:}A.\, \lambda b{:}B.\, f(a,b).$$

It is itself strict and continuous, but this is of no real interest for us.

We leave all the claims made above without proofs. These can be found in the literature, e.g., [Stoy 77], [Guessarian 81], [Smyth&Plotkin 82], etc. Moreover, they follow from the general results we shall quote in the subsequent section.

6.2.3 Σ-domains

We need to put these two kinds of structures, i.e., Σ-algebras and ω-cppo-s, together to interpret recursive languages. What we obtain will be called Σ-*domains*.

Definition 6.2.16 A Σ-*domain* $\mathcal{A}$ is a four-tuple:

$$\mathcal{A} = \langle A, \sqsubseteq_{\mathcal{A}}, \bot_{\mathcal{A}}, \Sigma_{\mathcal{A}} \rangle$$

where

$\langle A, \Sigma_{\mathcal{A}} \rangle$ is a Σ-algebra,
$\langle A, \sqsubseteq_{\mathcal{A}}, \bot_{\mathcal{A}} \rangle$ is a ω-cppo,
$\sigma_{\mathcal{A}}: A^n \to_\omega A$ are continuous, for $\sigma \in \Sigma_n$.

A Σ-*homomorphism* f from $\mathcal{A}$ to $\mathcal{B}$ of Σ-domains is a strict continuous map between the underlying ω-cppo-s which is a Σ-homomorphism between the underlying Σ-algebras.

As expected, in this way we obtain a category of Σ-domains.

Fact 6.2.17 *Σ-domains with Σ-homomorphisms as their morphisms, constitute a category, denoted* $\Sigma\text{-}\mathbf{alg}(\omega\text{-}\mathbf{CPPO}_\bot)$.

Proof. Standard. One needs to verify that the composition and identities inherited from $\omega\text{-}\mathbf{CPPO}_\bot$ make $\Sigma\text{-}\mathbf{alg}(\omega\text{-}\mathbf{CPPO}_\bot)$ into a category. □

Finally, let the interpreting function $\mathcal{A}[\![_]\!]: \mathbf{Rec}_\Sigma(\mathbf{Var}) \to (\mathbf{Env}_{\mathcal{A}} \to_\omega \mathcal{A})$ be defined inductively as follows:

$$\begin{aligned}
\mathcal{A}[\![x]\!] &= \lambda\rho{:}\mathbf{Env}_{\mathcal{A}}.\, \rho(x), \qquad \text{for } x \in \mathbf{Var};\\
\mathcal{A}[\![\sigma(t_1,\ldots,t_n)]\!] &= \sigma_{\mathcal{A}} \circ (\mathcal{A}[\![t_1]\!] \times \ldots \times \mathcal{A}[\![t_n]\!]) \circ \Delta^n_{\mathbf{Env}_{\mathcal{A}}},\\
&\qquad \text{for } \sigma \in \Sigma_n, t_1,\ldots,t_n \in \mathbf{Rec}_\Sigma(\mathbf{Var}); \qquad (6.6)\\
\mathcal{A}[\![\mathbf{rec}\, x.\, t]\!] &= \Upsilon_{\mathcal{A}} \circ curry(\mathcal{A}[\![t]\!] \circ \lambda\langle\rho,a\rangle{:}\mathbf{Env}_{\mathcal{A}} \times \mathcal{A}.\, \rho[x \mapsto a]),\\
&\qquad \text{for } x \in \mathbf{Var}, t \in \mathbf{Rec}_\Sigma(\mathbf{Var}).
\end{aligned}$$

As a consequence of our unproved claims we obtain the following result.

Lemma 6.2.18 *$\mathcal{A}[\![_]\!]$ is well-defined.*

Proof. It is easy to show that $\lambda\rho{:}\mathbf{Env}_{\mathcal{A}}.\ \rho(x)$, and $\lambda\langle\rho,a\rangle{:}\mathbf{Env}_{\mathcal{A}}\times A.\ \rho[x\mapsto a]$ are continuous, for $x\in\mathsf{Var}$. All the other functions used in the definition (6.6) were claimed continuous. Hence, the result follows by induction on the structure of the terms involved. □

Let us unpack what is hidden above behind the notation

The first two clauses are identical to those in the last definition, (6.4), modulo reformulation. This means that the above function is an extension of (6.4) from $\mathsf{T}_\Sigma(\mathsf{Var})$ onto $\mathsf{Rec}_\Sigma(\mathsf{Var})$.

For a recursive term $\mathbf{rec}\ x.\ t$ we obtain the following.
First, $\mathcal{A}[\![t]\!]\circ\Phi_x^{\mathcal{A}}{:}\mathbf{Env}_{\mathcal{A}}\times A\to_\omega A$ is given by:

$$(\mathcal{A}[\![t]\!]\circ\Phi_x)(\rho,a)=\mathcal{A}[\![t]\!](\rho[x\mapsto a]),\qquad \text{for } a\in A,\rho\in\mathbf{Env}_{\mathcal{A}}.$$

Thus, $curry(\mathcal{A}[\![t]\!]\circ\Phi_x^{\mathcal{A}}){:}\mathbf{Env}_{\mathcal{A}}\to_\omega(A\to_\omega A)$ is given by:

$$curry(\mathcal{A}[\![t]\!]\circ\Phi_x^{\mathcal{A}})\,\rho=\lambda a{:}A.\ \mathcal{A}[\![t]\!](\rho[x\mapsto a]),\qquad \text{for } \rho\in\mathbf{Env}_{\mathcal{A}}.$$

Finally, $\mathcal{A}[\![\mathbf{rec}\ x.\ t]\!]{:}\mathbf{Env}_{\mathcal{A}}\to_\omega A$ is given by:

$$\begin{aligned}\mathcal{A}[\![\mathbf{rec}\ x.\ t]\!]\,\rho&=\Upsilon_{\mathcal{A}}\circ curry(\mathcal{A}[\![t]\!]\circ\Phi_x^{\mathcal{A}})\,\rho\\&=\bigsqcup\{\bot_{\mathcal{A}},\ \mathcal{A}[\![t]\!](\rho[x\mapsto\bot_{\mathcal{A}}]),\ \mathcal{A}[\![t]\!](\rho[x\mapsto\mathcal{A}[\![t]\!](\rho[x\mapsto\bot_{\mathcal{A}}])]),\\&\qquad\mathcal{A}[\![t]\!](\rho[x\mapsto\mathcal{A}[\![t]\!](\rho[x\mapsto\mathcal{A}[\![t]\!](\rho[x\mapsto\bot_{\mathcal{A}}])])]),\ldots\},\qquad\text{for }\rho\in\mathbf{Env}_{\mathcal{A}}.\end{aligned}$$

As we see the meaning of a recursive term in an environment is obtained as a least upper bound of a certain chain. One could show that the meaning of the term $\mathbf{rec}\ x.\ t$ is a lub of a sequence of meanings of Σ-terms that *syntactically approximate* the term. This, however, is beyond our interest, and therefore omitted.

For completeness sake we prove the Substitution Lemma of denotational semantics. First, however, the *composition* $\rho\circ\varrho$ of substitution ϱ with $\mathcal{A}$-environment ρ is defined to be a $\mathcal{A}$-environment given by: $\rho\circ\varrho(x)=\mathcal{A}[\![\varrho\,x]\!]\rho$.

Lemma 6.2.19 *(i) If $\rho_0=^{\mathsf{FV}(t)}\rho_1$ then $\mathcal{A}[\![t]\!]\rho_0=\mathcal{A}[\![t]\!]\rho_1$;*
(ii) $\mathcal{A}[\![t\,\varrho]\!]\rho=\mathcal{A}[\![t]\!](\rho\circ\varrho)$.

Proof. (i) Follows easily by induction on the structure of t. See also proof of more general Lemma 6.3.13.

(ii) By induction on the structure of t. Cases where t is a variable or a function symbol applied to some terms are left to the reader. For $t = \mathbf{rec}\, x.\, u$ we have:

$$\begin{aligned}
\mathcal{A}[\![(\mathbf{rec}\, x.\, u)\, \varrho]\!]\rho &= \mathcal{A}[\![\mathbf{rec}\, y.\, u\, \varrho[x \mapsto y]]\!]\rho \\
&= \Upsilon_{\mathcal{A}} \circ curry(\mathcal{A}[\![u\, \varrho[x \mapsto y]]\!] \circ \Phi_y^{\mathcal{A}})\rho \\
&= \Upsilon_{\mathcal{A}}(\lambda a{:}A.\ \mathcal{A}[\![u\, \varrho[x \mapsto y]]\!]\rho[y \mapsto a]) \\
&= \Upsilon_{\mathcal{A}}(\lambda a{:}A.\ \mathcal{A}[\![u]\!](\rho[y \mapsto a] \circ \varrho[x \mapsto y])) \qquad \text{induction,}
\end{aligned}$$

where $y = \mathsf{choice}(\mathsf{new}\, x\, t\, \varrho)$, while

$$\mathcal{A}[\![\mathbf{rec}\, x.\, u]\!](\rho \circ \varrho) = \Upsilon_{\mathcal{A}}(\lambda a{:}A.\ \mathcal{A}[\![u]\!](\rho \circ \varrho[x \mapsto a]).$$

So, the result follows by (i) if one knows that

$$\rho \circ \varrho[x \mapsto a]) =^{\mathsf{FV}(u)} \rho[y \mapsto a] \circ \varrho[x \mapsto y].$$

The reader can fill in the remaining details himself, or find them in [Stoughton ??]. □

As a corollary we give a result that shows that $\mathbf{rec}\, x.\, t$ and its expansion $t[x \mapsto \mathbf{rec}\, x.\, t]$ have the same meaning.

Corollary 6.2.20 $\mathcal{A}[\![\mathbf{rec}\, x.\, t]\!] = \mathcal{A}[\![t[x \mapsto \mathbf{rec}\, x.\, t]]\!]$.

Proof. The reader may wish to check that the following calculations are correct.

$$\begin{aligned}
&\mathcal{A}[\![\mathbf{rec}\, x.\, t]\!]\, \rho = \Upsilon_{\mathcal{A}} \circ curry(\mathcal{A}[\![t]\!] \circ \Phi_x)\, \rho \\
&\quad = \bigsqcup\{\bot_{\mathcal{A}},\ \mathcal{A}[\![t]\!](\rho[x \mapsto \bot_{\mathcal{A}}]),\ \mathcal{A}[\![t]\!](\rho[x \mapsto \mathcal{A}[\![t]\!](\rho[x \mapsto \bot_{\mathcal{A}}])]), \\
&\qquad\quad \mathcal{A}[\![t]\!](\rho[x \mapsto \mathcal{A}[\![t]\!](\rho[x \mapsto \mathcal{A}[\![t]\!](\rho[x \mapsto \bot_{\mathcal{A}}])])]), \ldots\} \\
&\quad = \bigsqcup\{\mathcal{A}[\![t]\!](\rho[x \mapsto \bot_{\mathcal{A}}]),\ \mathcal{A}[\![t]\!](\rho[x \mapsto \mathcal{A}[\![t]\!](\rho[x \mapsto \bot_{\mathcal{A}}])]), \ldots\} \\
&\quad = \mathcal{A}[\![t]\!](\bigsqcup\{\rho[x \mapsto \bot_{\mathcal{A}}],\ \rho[x \mapsto \mathcal{A}[\![t]\!](\rho[x \mapsto \bot_{\mathcal{A}}])], \ldots\}) \\
&\quad = \mathcal{A}[\![t]\!]\, \rho[x \mapsto \bigsqcup\{\bot_{\mathcal{A}},\ \mathcal{A}[\![t]\!](\rho[x \mapsto \bot_{\mathcal{A}}]), \ldots\}] \\
&\quad = \mathcal{A}[\![t]\!]\, \rho[x \mapsto \mathcal{A}[\![\mathbf{rec}\, x.\, t]\!]\, \rho \\
&\quad = \mathcal{A}[\![t[x \mapsto \mathbf{rec}\, x.\, t]]\!]\, \rho.
\end{aligned}$$

□

Remark. One thing remains missing: an account of categorical importance of the interpreting function.

The recursive terms have obviously the structure of a Σ-algebra. Also, they can be equipped with appropriate substitutive preordering, or even a partial order. Unfortunately, not all ω-chains in such a poset have lub-s, and so it is not a ω-cppo.

The usually given characterisation involves the procedure of completion of posets to ω-cppo-s, with *continuity constraints*, i.e., a requirement that certain existing lub-s are to be preserved in the completion. This is extended to completion of partially ordered Σ-algebras to Σ-domains. One can show that such a completion of partially ordered Σ-algebra of closed

recursive Σ-terms is initial in the category of Σ-domains, which corresponds to Theorem 6.2.5. Appropriate generalisation of Theorem 6.2.8 also holds, see e.g., [ADJ 77], [Stoughton 86] and [Hennessy 85/86].

We do not expand on the matter. For one thing it would take too much space to review the results already available. Another point is that in our opinion they do not give a completely satisfactory answer to the problem. Simply, the completion procedure adds to the carrier certain new elements.

6.3 Categorical Interpretation of Recursive Languages

There is a natural generalisation from ordered sets to categories. Most of the results we need were obtained by D. Lehmann in collaboration with M. Smyth. The pattern of generalisation is obvious. We obtain the following correspondences:

poset	category
order preserving function	functor
bottom	initial object
ω-chain	ω-diagram
least upper bound	colimit
lub of ω-chain	colimit of ω-diagram
ω-continuous function	ω-colimits preserving functor

6.3.1 ω-categories and ω-functors.

In the remaining part of the section $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ will denote categories.

Our first task is to generalise the notion of ω-cppo.

Definition 6.3.1 ([Smyth&Plotkin 82]) Category $\mathbf{A}$ is called a *ω-complete pointed category*, abbr.: *ω-category*, iff it has colimits of all ω-diagrams and an initial object.

A functor $F: \mathbf{A} \to \mathbf{B}$ between ω-categories is called *continuous*, or *ω-functor* iff it preserves colimits of ω-diagrams. It is called *strict* provided it preserves initial objects.

The reader should note that in category theory the terminology ω-category has a different meaning.

The definition of a ω-category indeed generalises that of ω-cppo. Similarly, (strict) ω-functors generalise (strict) ω-continuous maps. To see this it is enough to consider posets as categories, and order preserving maps as functors.

We use $F: \mathbf{A} \to_\omega \mathbf{B}$ to denote ω-functors. This is consistent with our notation for posets. The convention to use the term *continuous functor* as a synonym to *ω-functor* extends the previous one, and the reader should be aware that numerous categorists use this terminology to talk about limits preserving functors.

We obtain the following results.

Lemma 6.3.2 *Composition of (strict) ω-functors gives (strict) ω-functors.*
Small ω-categories with strict ω-functors constitute a category, denoted $\omega-\mathbf{Cat}_{\perp}$.
Small ω-categories with ω-functors constitute a category, denoted $\omega-\mathbf{Cat}$.

Proof. Obvious. □

Next, we generalise the notion of a (pre-)fixed point. Note that this time we need to know a morphism as well as an object. In case of posets regarded as categories this simplifies to the requirement that such a morphism exists, i.e., $f(x) \sqsubseteq_{\mathbf{P}} x$, for a poset $\mathbf{P}$ and $f: \mathbf{P} \to \mathbf{P}$.

Definition 6.3.3 [[Smyth&Plotkin 82]] Let $\mathbf{A}$ be a category and $F: \mathbf{A} \to \mathbf{A}$ be an endofunctor. Then a *prefixed point* of F is a pair: $\langle A, \alpha \rangle$, where A is an object of $\mathbf{A}$ and $\alpha: FA \to A$ is a morphism.

A *fixed point* is such a prefixed point $\langle A, \alpha \rangle$ that $\alpha: FA \to A$ is an isomorphism in $\mathbf{A}$.

Often, the prefixed points of F are referred to as *F-algebras*, e.g., [Smyth&Plotkin 82], while [Arbib&Manes 75] call them *F-dynamic*.

Definition 6.3.4 Let $F: \mathbf{A} \to \mathbf{A}$ be a functor, $\langle A, \alpha \rangle, \langle A', \alpha' \rangle$ be F-algebras. Then their *morphism* $h: \langle A, \alpha \rangle \to \langle A', \alpha' \rangle$, also referred to as *$F$-homomorphism*, is just a morphism $h: A \to A'$ in $\mathbf{A}$ such that the following diagram commutes:

$$\begin{array}{ccc} F(A) & \xrightarrow{\alpha} & A \\ {\scriptstyle F(h)}\downarrow & & \downarrow{\scriptstyle h} \\ F(A') & \xrightarrow{\alpha'} & A' \end{array}$$

The above definitions give a category. The proof of the following Fact is standard, and therefore omitted.

Fact 6.3.5 *F-algebras with their morphisms form a category of* F-algebras in $\mathbf{A}$, *denoted* $F - \mathbf{alg}(\mathbf{A})$, *for an endofunctor* $F: \mathbf{A} \to \mathbf{A}$.

Finally, we can turn to the intended generalisation of the Fixpoint Theorem, Theorem 6.2.14. More general results can be found in various places in the literature, e.g., [Lehmann 76], or [Lehmann&Smyth 81]. The version presented is a special case of The Basic Lemma of [Smyth&Plotkin 82].

Theorem 6.3.6 *Let $\mathbf{A}$ be a ω-category and let $F: \mathbf{A} \to \mathbf{A}$ be a ω-functor. Then there exist an initial F-algebra, and it is an initial fixed point of F as well.*

Proof. Let $\perp_{\mathbf{A}}$ be an initial object in $\mathbf{A}$, and let us denote by $\perp$ the unique morphism $\perp: \perp_{\mathbf{A}} \to F\perp_{\mathbf{A}}$. An ω-diagram Γ is defined by: $\Gamma = \perp_{\mathbf{A}} \xrightarrow{\perp} F\perp_{\mathbf{A}} \xrightarrow{F\perp} F^2\perp_{\mathbf{A}} \xrightarrow{F^2\perp} \ldots$.

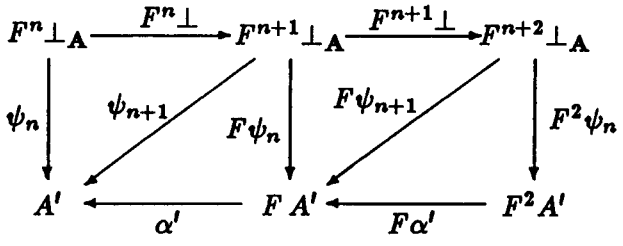
Since $\mathbf{A}$ has ω-colimits we can find a colimiting cocone ϕ of Γ, $\phi: \Gamma \to A$. Let $\phi^-: F \circ \Gamma \to A$ be a cocone defined by: $\phi^-_n = \phi_{n+1}: F^{n+1}\perp_{\mathbf{A}} \to A$, for $n \in \omega$.

From continuity of F it follows that there exists a morphism $\alpha: FA \to A$ mediating between the colimiting cocone $F\phi$, and ϕ^-. Thus, $\langle A, \alpha \rangle$ is a F-algebra.

It is easy to show that α is an isomorphism, and we leave it as an exercise. It remains to show that $\langle A, \alpha \rangle$ is an initial F-algebra. To this end assume that $\langle A', \alpha' \rangle$ is a F-algebra. Let $\psi: \Gamma \to A'$ be a cocone defined by:

$$\psi_0 = \bot': \bot_{\mathbf{A}} \to A' \quad \text{– the unique morphism from initial } \bot_{\mathbf{A}}, \text{ and}$$
$$\psi_{n+1} = \alpha' \circ F\psi_n.$$

To see that we get a cocone consider the following diagram:



For $n = 0$ the left square commutes by initiality of $\bot_{\mathbf{A}}$ and definition of ψ_1. In general, the commutativity of a left square is an inductive assumption. Thus, the right square commutes, as an image of a commuting diagram via a functor. But then

$$\psi_{n+1} = \alpha' \circ F\psi_n = \alpha' \circ (F\psi_{n+1} \circ F^{n+1}\bot) = \psi_{n+2} \circ F^{n+1}\bot,$$

as required.

Since ψ is a cocone we obtain a morphism $f: A \to A'$ mediating between the colimiting cocone ϕ, and ψ. But then Ff mediates between $F\phi$ and $F\psi$. This implies that both: $f \circ \alpha$ and $\alpha' \circ Ff$ are mediating between $F\phi$ and ψ^-. Therefore, $f \circ \alpha = \alpha' \circ Ff$, i.e., f is a homomorphism of F-algebras.

Conversely, any F-homomorphism $f: \langle A, \alpha \rangle \to \langle A', \alpha' \rangle$ must mediate between ϕ and ψ, which implies uniqueness. The proof goes by induction. For $n = 0$ the equality $f \circ \phi_0 = \psi_0$ follows from initiality of $\bot_{\mathbf{A}}$. For $n + 1$ we have:

$$\begin{aligned}
f \circ \phi_{n+1} &= f \circ (\alpha \circ F\phi_n) \\
&= (\alpha' \circ Ff) \circ F\phi_n && \text{since } f \text{ is a } F\text{-homomorphism} \\
&= \alpha' \circ (F(f \circ \phi_n)) \\
&= \alpha' \circ F\psi_n && \text{by induction} \\
&= \psi_{n+1},
\end{aligned}$$

as required. □

The above is a slight reformulation of Theorem 2 in [Lehmann 76], which in turn is a corollary to the Theorem 1, ibid., credited there to M. Smyth.

Once more we are interested in some general constructions that give ω-categories and ω-functors. We follow the presentation of the previous section rather closely. The reader is advised to consult [Lehmann 76] and [Lehmann&Smyth 81] for more. We let $\mathbf{A}, \mathbf{B}, \mathbf{C}$, etc., denote ω-categories in the remaining part of this section.

First examples come from the observation that the forgetful functor $U:\omega-\mathbf{Cat} \to \mathbf{Cat}$ creates finite products in $\omega-\mathbf{Cat}$ and in $\omega-\mathbf{Cat}_\perp$.

$\mathbf{1_{Cat}}$ a *terminal* category with one object, and one morphism. It is a ω-category for obvious reasons. It is easy to see that it is a terminal object in $\omega-\mathbf{Cat}$ and in $\omega-\mathbf{Cat}_\perp$. In $\omega-\mathbf{Cat}_\perp$ it is initial as well.

$\mathbf{A \times B}$ the usual *product* of categories. It has initial objects: $\langle \perp_{\mathbf{A}}, \perp_{\mathbf{B}} \rangle$ where $\perp_{\mathbf{A}}, \perp_{\mathbf{B}}$ are initial in $\mathbf{A}$ and $\mathbf{B}$, respectively. Recall that all limits and colimits in the product are computed componentwise, so in particular the product has ω-(co)limits iff both component categories have them.

$\pi_{\mathbf{A}}: \mathbf{A \times B} \to_\omega \mathbf{A}, \pi_{\mathbf{B}}: \mathbf{A \times B} \to_\omega \mathbf{B}$ the usual *projection* functors are strict and continuous.

This gives us the following well known result quoted here without proof.

Fact 6.3.7 *The product of ω-cppo-s defined above is a categorical product in both: $\omega-\mathbf{Cat}$ and $\omega-\mathbf{Cat}_\perp$.*

$F \times G: \mathbf{A \times C} \to_\omega \mathbf{B \times D}$ is the *product* of ω-functors: $F: \mathbf{A} \to_\omega \mathbf{B}$, and $G: \mathbf{C} \to_\omega \mathbf{D}$. As a consequence of the above Fact product becomes a bifunctor $_ \times _ : \omega-\mathbf{Cat}^2 \to \omega-\mathbf{Cat}$. In addition to being continuous it cuts down to $\omega-\mathbf{Cat}_\perp$: $F \times G$ is strict for strict F and G.

$id_{\mathbf{A}}: \mathbf{A} \to_\omega \mathbf{A}$ the *identity* functor is obviously strict and continuous.

$K_B: \mathbf{A} \to_\omega \mathbf{B}$ the *constant* functor, where $B \in \mathbf{B}$, given by:

$$\begin{aligned} K_B(A) &= B, \qquad \text{for } A \in \mathbf{A}; \\ K_B(f) &= id_B, \qquad \text{for } f \in \mathbf{Hom_A}. \end{aligned}$$

It is always continuous. It is strict iff $B = \perp_{\mathbf{B}}$.

$\mathbf{A}^X$ or $X \to \mathbf{A}$ The category of all functors from a category X to $\mathbf{A}$. It has functors as objects and natural transformations as morphisms. It has $K_{\perp_{\mathbf{A}}}$ as initial objects, for $\perp_{\mathbf{A}}$ initial in $\mathbf{A}$, see [Lehmann 76]. It has colimits of ω-diagrams, since these are computed pointwise, see [Mac Lane 71, pp.111–112] for discussion of the general, but dual, case.

$\Delta^X_{\mathbf{A}}: \mathbf{A} \to_\omega \mathbf{A}^X$ the strict and continuous *diagonal* functor is given by:

$$\begin{aligned} \Delta^X_{\mathbf{A}}(A) &= K_A \qquad \text{the constant functor, for } A \in \mathbf{A}; \\ (\Delta^X_{\mathbf{A}}(f))_x &= f \qquad \text{for } f \in \mathbf{Hom_A}, x \in \mathbf{X}. \end{aligned}$$

One readily sees that for a ω-diagram $\Gamma: \omega \to \mathbf{A}$ with colimiting cocone $\phi: \Gamma \to A$, the cocone $\Delta^X_{\mathbf{A}}(\phi): \Delta^X_{\mathbf{A}} \circ \Gamma \to K_A$ is colimiting.

$\mathbf{A} \to_\omega \mathbf{B}$ the full subcategory of $\mathbf{A} \to \mathbf{B}$ of continuous functors. It inherits initial objects from $\mathbf{A}^{\mathbf{B}}$. Lehmann, [Lehmann 76,Lehmann 76a], gives a detailed proof that it is indeed a ω-category. See also [Lehmann&Smyth 81].

$\Upsilon_{\mathbf{A}}: (\mathbf{A} \to_\omega \mathbf{A}) \to_\omega \mathbf{A}$ the strict and continuous *least fixed point operator*. It is defined as a composition of two strict and continuous functors.

Let $\perp_{\mathbf{A}}$ be an initial object in $\mathbf{A}$. Define $\Xi_{\mathbf{A}}: (\mathbf{A} \to_\omega \mathbf{A}) \to_\omega \mathbf{A}^\omega$ on objects, i.e., functors:

$$\Xi_{\mathbf{A}}F = \bot_{\mathbf{A}} \xrightarrow{\bot} F\bot_{\mathbf{A}} \xrightarrow{F\bot} F^2\bot_{\mathbf{A}} \xrightarrow{F^2\bot} \ldots, \qquad \text{for } F: \mathbf{A} \to_\omega \mathbf{A};$$

and on morphisms, i.e., natural transformations:

$$(\Xi_{\mathbf{A}}\tau)_n = ((\tau)^{\textcircled{n}})_{\bot_{\mathbf{A}}} = (\underbrace{\tau \circ \ldots \circ \tau}_{n \text{ times}})_{\bot_{\mathbf{A}}}: F^n\bot_{\mathbf{A}} \to G^n\bot_{\mathbf{A}}, \qquad \text{for } \tau: F \to G,\ n \in \omega.$$

Recall that $\tau \circ \tau'$ denotes the horizontal composition of natural transformations.

Lemma 6.3.8 $\Xi_{\mathbf{A}}: (\mathbf{A} \to_\omega \mathbf{A}) \to \mathbf{A}^\omega$ *is strict and continuous.*

Proof. Strictness is, hopefully, evident. Consult [Lehmann&Smyth 81] for the proof of continuity of $\Xi_{\mathbf{A}}$. □

Therefore, we can put: $\Upsilon_{\mathbf{A}} = \text{colim} \circ \Xi_{\mathbf{A}}$, where colim: $\mathbf{A}^\omega \to \mathbf{A}$ is the functor that sends ω-diagrams to vertices of their colimiting cocones. As a left adjoint to $\Delta^\omega_{\mathbf{A}}$, see [Mac Lane 71], it is automatically strict and ω-continuous by Proposition 2.2.2.

More specificly, for $F: \mathbf{A} \to \mathbf{A}$ continuous $\Upsilon_{\mathbf{A}}F$ is, simply, the vertex of an initial F-algebra $\langle A, \alpha \rangle$, while $\Upsilon_{\mathbf{A}}\tau$, for $\tau: F \to G$, is the unique F-morphism from the F-algebra $\langle A, \alpha \rangle$ to the F-algebra $\langle B, \beta' \rangle$, where $\langle B, \beta \rangle$ is initial G-algebra, and $\beta' = \beta \circ \tau_B$.

For the detailed proofs and discussion consult [Lehmann&Smyth 81].

It is well known that a bifunctor $F: \mathbf{A} \times \mathbf{B} \to \mathbf{C}$ is a ω-functor iff the functors: $F(A, _): \mathbf{B} \to \mathbf{C}$ and $F(_, B): \mathbf{A} \to \mathbf{C}$ are ω-functors, for all objects A, B of $\mathbf{A}, \mathbf{B}$, respectively, where the latter are defined by

$$F(A, _)(B) = F(A, B), \qquad \text{for } B \in \mathbf{B};$$

$$F(A, _)(f) = F(id_A, f), \qquad \text{for } f \in \mathbf{Hom}_{\mathbf{B}},$$

and $F(_, B)$ similarly.

curry: $(\mathbf{A} \times \mathbf{B} \to_\omega \mathbf{C}) \to (\mathbf{A} \to_\omega (\mathbf{B} \to_\omega \mathbf{C}))$ is an isomorphism of categories given by:

$$curry(F)\,A = F(A, _), \qquad \text{for } A \in \mathbf{A};$$

$$curry(F)\,f: F(A, _) \to F(A', _), \qquad \text{for } f: A \to A' \text{ in } \mathbf{A} \text{ where}$$

$$(curry(F)\,f)_B = F(f, id_B);$$

$$((curry(\tau: F \to G))_A)_B = \tau_{(A,B)}, \qquad \text{for } A \in \mathbf{A}, B \in \mathbf{B}.$$

Its inverse *uncurry*: $(\mathbf{A} \to_\omega (\mathbf{B} \to_\omega \mathbf{C})) \to (\mathbf{A} \times \mathbf{B} \to_\omega \mathbf{C})$ is given by

$$\begin{aligned} uncurry(F)\langle A, B \rangle &= F\,A\,B \qquad \text{for } A \in \mathbf{A}, B \in \mathbf{B}; \\ uncurry(F)\langle f, g \rangle &= F\,A'(g) \circ (F\,f)_B \\ &= (F\,f)_{B'} \circ F\,A(g), \qquad \text{for } f: \to A', g: B \to B'; \\ (uncurry(\tau))_{\langle A,B \rangle} &= (\tau_A)_B, \qquad \text{for } A \in \mathbf{A}, B \in \mathbf{B}, \tau: F \to G. \end{aligned}$$

The reader is urged to consult [Lehmann&Smyth 81] for the proofs of all the above claims and more discussion.

6.3.2 Categorical Semantics of Recursive Languages.

One could motivate categorically our simple definition of a Σ-algebra in the category of all small ω-categories with strict ω-functors as morphisms. This would help to understand why prefixed point of a functor F are called F-algebras, or rather that our Σ-algebras can be seen as prefixed points of appropriate functors. We shall not attempt to do that as it can be found in the literature, see e.g., [Lehmann&Smyth 81], [Mac Lane 71].

Instead, we present more elementary description of Σ-algebras in $\omega-\mathbf{Cat}_\perp$. We refer to them, in agreement with previous convention, as Σ-*domains*. Indeed, an algebraic Σ-domain is a categorical Σ-domain of a simple form, according to our view of posets as categories.

Definition 6.3.9 A (categorical) Σ-*domain* $\mathbf{A}$ is a ω-category $\mathbf{A}$ together with a set $\{\sigma_{\mathbf{A}}: \mathbf{A}^n \to_\omega \mathbf{A} \mid \sigma \in \Sigma \wedge \mathrm{arity}(\sigma) = n\}$ of ω-functors.

A *homomorphism* H of Σ-domains $\mathbf{A}, \mathbf{B}$ is a strict ω-functor $H: \mathbf{A} \to_\omega \mathbf{B}$ such that the following diagram commutes:

$$\begin{array}{ccc} \mathbf{A}^n & \xrightarrow{H \times \ldots \times H} & \mathbf{B}^n \\ {\scriptstyle \sigma_{\mathbf{A}}}\downarrow & & \downarrow{\scriptstyle \sigma_{\mathbf{B}}} \\ \mathbf{A} & \xrightarrow{H} & \mathbf{B} \end{array}$$

for all $\sigma \in \Sigma_n$.

We obtain an exact copy of the Fact 6.2.17 which states that the (categorical) domains and their morphisms form a category.

Fact 6.3.10 Σ-*domains with* Σ-*homomorphisms as their morphisms, constitute a category, denoted* $\Sigma-\mathbf{alg}(\omega-\mathbf{Cat}_\perp)$.

> **Proof.** Standard. One needs to verify that the composition and identities inherited from $\omega-\mathbf{CPPO}_\perp$ makes $\Sigma-\mathbf{alg}(\omega-\mathbf{CPPO}_\perp)$ into a category. □

With the above machinery we are almost ready to define (categorical) denotational semantics of recursive languages. We assume that a signature Σ is given, and let $\mathbf{A}$ be a (categorical) Σ-domain. The reader will notice that the clauses are virtually identical to (6.6) in the previous section.

First, let $\mathbf{Env}_{\mathbf{A}} = \mathrm{Var} \to \mathbf{A}$ be a category of $\mathbf{A}$-environments, i.e., the category of all functors from Var considered as a discrete category to $\mathbf{A}$. We argued that categories like $\mathbf{Env}_{\mathbf{A}}$ are ω-categories whenever $\mathbf{A}$ is a ω-category, which it is as a Σ-domain.

Next, let us define two auxiliary ω-functors.

Let $\Psi_x^{\mathbf{A}}: \mathbf{Env}_{\mathbf{A}} \to_\omega \mathbf{A}$, for $x \in \mathrm{Var}$ be defined by:

$$\Psi_x^{\mathbf{A}}(\rho) = \rho(x) \quad \text{and} \quad \Psi_x^{\mathbf{A}}(\tau) = \tau_x,$$

for $\rho \in \mathbf{Env_A}$, $\tau \in \mathbf{Hom_{Env_A}}$.

Also, let $\Phi_x^{\mathbf{A}}: \mathbf{Env_A} \times \mathbf{A} \to_\omega \mathbf{Env_A}$, for $x \in \mathsf{Var}$, be defined by:

$$\Phi_x^{\mathbf{A}}(\rho, A) = \rho[x \mapsto A] \quad \text{and} \quad \Phi_x^{\mathbf{A}}(\tau, f) = \tau[x \mapsto f],$$

for $\rho \in \mathbf{Env_A}$, $\tau \in \mathbf{Hom_{Env_A}}$, $A \in \mathbf{A}$ and $f \in \mathbf{Hom_A}$.

Lemma 6.3.11 *$\Psi_x^{\mathbf{A}}$ and $\Phi_x^{\mathbf{A}}$ are ω-functors.*

Proof. Colimits (and also limits) in functor categories are defined 'pointwise'. With this observation in mind the result follows easily. □

The interpreting function $\mathbf{A}[\![_]\!]: \mathsf{Rec}_\Sigma(\mathsf{Var}) \to (\mathbf{Env_A} \to_\omega \mathbf{A})$ is defined by induction on the structure of terms in $\mathsf{Rec}_\Sigma(\mathsf{Var})$ as follows:

$$\begin{aligned}
\mathbf{A}[\![x]\!] &= \Psi_x^{\mathbf{A}}, \qquad \text{for } x \in \mathsf{Var};\\
\mathbf{A}[\![\sigma]\!](t_1, \ldots, t_n) &= \sigma_{\mathbf{A}} \circ (\mathbf{A}[\![t_1]\!] \times \ldots \times \mathbf{A}[\![t_n]\!]) \circ \Delta^n_{\mathbf{Env_A}},\\
&\qquad \text{for } \sigma \in \Sigma_n, t_1, \ldots, t_n \in \mathsf{Rec}_\Sigma(\mathsf{Var});\\
\mathbf{A}[\![\mathsf{rec}\, x.\, t]\!] &= \Upsilon_{\mathbf{A}} \circ \mathit{curry}(\mathbf{A}[\![t]\!] \circ \Phi_x),\\
&\qquad \text{for } x \in \mathsf{Var}, t \in \mathsf{Rec}_\Sigma(\mathsf{Var}).
\end{aligned} \tag{6.7}$$

From the results established before we draw an immediate conclusion.

Theorem 6.3.12 *$\mathbf{A}[\![_]\!]$ is well-defined*

Proof. It is the consequence of all the results reviewed in this section that all the functors used in the definition are continuous. Thus, the result follows by structural induction. □

In the future the sub-/superscript $\mathbf{A}$ in $\Upsilon_{\mathbf{A}}$, $\Psi_x^{\mathbf{A}}$ and $\Phi_x^{\mathbf{A}}$ will be omitted whenever it is evident from the context what category $\mathbf{A}$ is being considered.

In the sequel we will need a standard result saying that the denotation of a term, which is a functor, depends essentially only on the value of the environments and their morphisms at those variables that occur free in the term. This, obviously, is a generalisation of the first part of Lemma 6.2.19.

Lemma 6.3.13 *Let $\rho, \rho' \in \mathbf{Env_A}$, $\tau, \tau' \in \mathbf{Hom_{Env_A}}$ and $t \in \mathsf{Rec}_\Sigma(\mathsf{Var})$. Then*

$$\rho =^{\mathsf{FV}(t)} \rho' \;\Rightarrow\; \mathbf{A}[\![t]\!]\rho = \mathbf{A}[\![t]\!]\rho' \quad \textit{and} \quad \tau =^{\mathsf{FV}(t)} \tau' \;\Rightarrow\; \mathbf{A}[\![t]\!]\tau = \mathbf{A}[\![t]\!]\tau'.$$

Proof. By induction on the structure of t.

$\boxed{t = x}$ Then $\mathsf{FV}(t) = \{x\}$ and the result follows by definition of $\mathbf{A}[\![t]\!]$.

$\boxed{t = \sigma(t_1, \ldots, t_n)}$ Then $\mathsf{FV}(t) = \mathsf{FV}(t_1) \cup \ldots \cup \mathsf{FV}(t_n) \supseteq \mathsf{FV}(t_i)$, for $i = 1, \ldots, n$. The result follows by straightforward induction.

$\boxed{t = \mathsf{rec}\, x.\, t'}$ This time we have $\mathsf{FV}(t) = \mathsf{FV}(t') - \{x\}$.

We show $\mathbf{A}[\![t]\!]\rho = \mathbf{A}[\![t]\!]\rho'$. For $A \in \mathbf{A}$ we obtain

$$\begin{aligned}
\mathbf{A}[\![t]\!]\rho\, A &= (\Upsilon_{\mathbf{A}} \circ curry(\mathbf{A}[\![u]\!] \circ \Phi_x))\,\rho\, A \\
&= \Upsilon_{\mathbf{A}}(\mathbf{A}[\![t']\!]\,\rho[x \mapsto A]) \\
&= \Upsilon_{\mathbf{A}}(\mathbf{A}[\![t']\!]\,\rho'[x \mapsto A]) \qquad \text{induction} \\
&= \mathbf{A}[\![t]\!]\,\rho'\, A
\end{aligned}$$

The inductive step above is justified as $\rho[x \mapsto A]\,(y) = \rho'[x \mapsto A]\,(y)$, for $y \in \mathsf{FV}(t')$.

Similarly, for $f \in \mathbf{Hom}_{\mathbf{A}}$ we have:

$$\begin{aligned}
\mathbf{A}[\![t]\!]\,\rho\, f &= \Upsilon_{\mathbf{A}} \circ curry(\mathbf{A}[\![t']\!] \circ \Phi_x)\,\rho\, f \\
&= \Upsilon_{\mathbf{A}}(\mathbf{A}[\![t']\!]\, id_\rho[x \mapsto f]) \\
&= \Upsilon_{\mathbf{A}}(\mathbf{A}[\![t']\!]\, id_{\rho'}[x \mapsto f]) \qquad \text{induction} \\
&= \mathbf{A}[\![t]\!]\,\rho'\, f
\end{aligned}$$

The inductive step depends upon $(id_\rho[x \mapsto f])_y = (id_{\rho'}[x \mapsto f])_y$, for $y \in \mathsf{FV}(t')$.

And finally, $\mathbf{A}[\![t]\!]\,\tau = \mathbf{A}[\![t]\!]\,\tau'$, since for $A \in \mathbf{A}$ we obtain

$$\begin{aligned}
(\mathbf{A}[\![t]\!]\,\tau)_A &= (\Upsilon_{\mathbf{A}} \circ curry(\mathbf{A}[\![t']\!] \circ \Phi_x)\,\tau)_A \\
&= \Upsilon_{\mathbf{A}}(\mathbf{A}[\![t']\!]\,\tau[x \mapsto id_A]) \\
&= \Upsilon_{\mathbf{A}}(\mathbf{A}[\![t']\!]\,\tau'[x \mapsto id_A]) \qquad \text{induction} \\
&= (\mathbf{A}[\![t]\!]\,\tau')_A
\end{aligned}$$

Again, the inductive hypothesis can be applied, since for $y \in \mathsf{FV}(t')$ we have $(\tau[x \mapsto id_A])_y = (\tau'[x \mapsto id_A])_y$. □

From the general perspective it may be a bit surprising to obtain an identity, rather than an isomorphism, in the above result.

Also the second part of Lemma 6.2.19, known usually as The Substitution Lemma, can be generalised without any substantial changes in the proof. We do not have any application for it and therefore the discussion is omitted.

Let us recapitulate on the contents of this section.

In order to give categorical denotational semantics for a recursive language over a signature Σ one has to chose:

(i) a ω-category $\mathbf{A}$;

(ii) for each function symbol $\sigma \in \Sigma_n$ a continuous functor $\sigma_{\mathbf{A}}\colon \mathbf{A}^n \to_\omega \mathbf{A}$.

And that is all.

Chapter 7

$\mathcal{L}$ — a Universal Language for Synchronisation

In this Chapter we introduce a language $\mathcal{L}$. We call it 'universal' for it is possible to model in it various synchronisation mechanisms from languages like: *CCS* [Milner 80], *SCCS* [Milner 83], *TCSP* [Hoare et. al. 84], etc. In fact, it is powerful enough to model any synchronisation mechanism definable by means of Winskel's *synchronisation algebras*, [Winskel 84b].

It is pleasing to find a simple language in which such a variety of synchronisation operators can be derived. Even more pleasing is that it is possible to consider many of the above languages as sublanguages of $\mathcal{L}$ with their operational semantics derived from the operational semantics of $\mathcal{L}$. Also, one can study for instance Milner's *SCCS* enriched with synchronisation operators from, say, *TCSP*, as a sublanguage of $\mathcal{L}$.

Therefore, $\mathcal{L}$ provides a uniform framework within which *CCS*-like languages can be studied. This, in turn makes studying $\mathcal{L}$ worthwhile. When, for instance, we give $\mathcal{L}$ a denotational semantics in Chapter 8 then, automatically, we obtain denotational semantics for all its sublanguages.

We cannot claim, however, that all the syntactic constructs from all *CCS*-like languages find their place as derived operators of $\mathcal{L}$ equally well. Specifically, the many *sum* operators cannot be satisfactorily accommodated.

Winskel's synchronisation algebras were devised with the aim of providing such a uniform framework as well. Hence, we should compare the two attempts more closely.

First, it is only fair to say that from this perspective our approach is just a variation on Winskel's work.

On one hand it is impossible to find a synchronisation algebra that would correspond to the synchronisation operator of $\mathcal{L}$. One can, however, adapt Winskel's definition, so that the task becomes feasible under some restrictions. Thus, the two approaches are in a broad sense equally powerful.

Historically, the language $\mathcal{L}$ was invented to serve a different purpose. It was devised as

a remedy to counter the difficulties with the construction of a non-interleaving semantics for *CCS* from its syntax and the operational semantics as given by Milner in [Milner 80].

There were many attempts to give *CCS* a non-interleaving semantics denotationally. For example Petri nets, [Goltz&Mycroft 84], or event structures, [Winskel 82], were used.

On the other hand there are natural criterions that a construction of, say, an event structure corresponding to a closed *CCS*-term from the syntax and the operational semantics should satisfy. It appears that *CCS* with its operational semantics given by Milner does not admit such a construction that would give the same result as denotational semantics proposed in the literature so far. But we have to postpone the discussion until Chapter 9.

The Chapter is organised as follows.

The first short section gives the formal definition of the syntax of $\mathcal{L}$, that is the signature of $\mathcal{L}$ is defined.

The second section defines its operational semantics in an axiomatic fashion à la Plotkin, see [Plotkin 81]. This method of presentation of operational semantics will be referred to as *axiomatic operational semantics.*

In the third section we make precise what it means that $\mathcal{L}$ models other languages. Milner's *CCS* is chosen as a particular example. It is shown that there exists a nice correspondence between the original *CCS*, with its axiomatic operational semantics, and the sublanguage $\mathcal{CCS}$ of $\mathcal{L}$ with which we intend to model it. As promised, the sublanguage $\mathcal{CCS}$ has its axiomatic operational semantics derived from that of $\mathcal{L}$.

Finally, in the last section we briefly discuss what should be done to accommodate operators from other languages. This is much easier to accomplish, in general, than in the case of *CCS*. Special attention is also given to simulation in $\mathcal{L}$ operators definable in terms of synchronisation algebras.

7.1 $\mathcal{L}$ – the Definition

To define a language all we need is to give its signature. Our definition contains a parameter, so that in fact we define a class of languages. Each of them depends on a given in advance set M of *elementary action symbols.* A particular choice of M will depend on its future intended applications. For the purpose of this section we assume that this set has been fixed, and we use m to denote its typical element.

Our first goal is to define $\mathcal{M} = \mathcal{M}(M)$ – the set of *action symbols over* M. Intuitively, $\mathcal{M}$ is a set which contains all binary products and coproducts in **Pfn** of $\mathcal{M}$ with itself, and also contains M.

Let $\Sigma^{\mathcal{M}}$ be a signature $\Sigma^{\mathcal{M}} = \Sigma_1^{\mathcal{M}} \cup \Sigma_2^{\mathcal{M}}$, i.e., $\Sigma_n^{\mathcal{M}} = \emptyset$, for $n \notin \{1,2\}$, where

$\Sigma_1^{\mathcal{M}} = \{\mathtt{inl}_, \mathtt{inr}_, \langle _, \star\rangle, \langle \star, _\rangle\}$;

$\Sigma_2^{\mathcal{M}} = \{\langle _, _\rangle\}$.

Now, we take $\mathcal{M}$ to be (the carrier of) the free $\Sigma^{\mathcal{M}}$-algebra on M, compare Example 6.2.7

and Theorem 6.2.8. That is to say, we take as M the set of all well-formed terms built from operators of Σ^M on elements of M as constants. We use μ, ν as generic elements of M.

Now, we are ready to present the signature $\Sigma^{\mathcal{L}}$ of the language $\mathcal{L}$.

We put $\Sigma^{\mathcal{L}} = \Sigma_0^{\mathcal{L}} \cup \Sigma_1^{\mathcal{L}} \cup \Sigma_2^{\mathcal{L}}$, where

$\Sigma_0^{\mathcal{L}} = \{\Omega\}$ – *inaction* operator;

$\Sigma_1^{\mathcal{L}} = \{m\cdot _ \mid m \in M\}$ – *action* or *prefix* operators;

$\cup\{_ \setminus L \mid L \subseteq M\}$ – *restriction* operators;

$\cup\{_[R] \mid R\colon M \rightharpoonup M\}$ – *renaming* operators;

$\Sigma_2^{\mathcal{L}} = \{_ \# _, _ \,\&\, _\}$ – *coproduct* and *product* operators.

In the future we will use a slightly simplified notation as compared to the notation introduced in Chapter 6. Thus, we write $\mathsf{T}_{\mathcal{L}}$ rather than $\mathsf{T}_{\Sigma^{\mathcal{L}}}$, and $\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$ for $\mathsf{Rec}_{\Sigma^{\mathcal{L}}}(\mathsf{Var})$, etc. Moreover, we use E, F to range over $\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$.

7.2 $\mathcal{L}$ – Axiomatic Operational Semantics

The operational semantics of $\mathcal{L}$ is presented in the now standard way developed by Plotkin in his Århus notes, [Plotkin 81]. We think of closed terms in $\mathcal{L}$ as *processes*. The idea is to formalise what it means for a process E to *evolve*. This is done by introduction of a concept of *atomic*, or *one-step computation*. We will write $E \xrightarrow{\mu} F$ to indicate that process E can in one step change into F, while the change is observable as μ.

Always when used in the context of axiomatic operational semantics we let μ and ν range over $M \cup \{\imath\}$ where $\imath$ is a special *silent action* not in M. Intuitively, $E \xrightarrow{\imath} F$ means that process E can evolve into F in a way that no external observer can ever notice.

To cope with this new action we extend partial unary functions $R\colon M \rightharpoonup M$ to total maps with $M \cup \{\imath\}$ as domain and codomain as follows:

$$R\mu = \begin{cases} \imath & \mu = \imath \vee R\mu\uparrow \\ R\mu & \textit{otherwise} \end{cases}.$$

This applies to all unary constructors in Σ^M.

The extension of the binary constructor is defined as follows:

$$\langle \mu, \nu \rangle = \begin{cases} \imath & \mu = \imath = \nu \\ \langle \mu, \star \rangle & \mu \neq \imath = \nu \\ \langle \star, \nu \rangle & \mu = \imath \neq \nu \\ \langle \mu, \nu \rangle & \mu \neq \imath \neq \nu \end{cases}.$$

The essence of Plotkin's idea is to introduce a formal *inference system*, within which statements of the form $E \xrightarrow{\mu} F$ can be derived. The system is based on the *natural deduction system* developed by Gentzen. Gentzen's main goal was to capture the *natural* way in which one usually reasons about validity of complex logical assertions in terms of the validity of their sub-assertions, see [Gentzen 34], and also [Prawitz 65]. The rules for $\mathcal{L}$ are collected

action	$\dfrac{}{m\cdot E \xrightarrow{m} E}$(act-$m$) for $m \in M$
restriction	$\dfrac{E \xrightarrow{\mu} F}{E \setminus L \xrightarrow{\mu} F \setminus L}$(res-$L$) for $\mu \notin L \subseteq \mathcal{M}$
renaming -hiding	$\dfrac{E \xrightarrow{\mu} F}{E[R] \xrightarrow{R\mu} F[R]}$(ren-$R$) for $R\colon \mathcal{M} \to \mathcal{M}$;
coproduct	$\dfrac{E \xrightarrow{\mu} F}{E \# E' \xrightarrow{\mathtt{inl}\mu} F[\mathtt{inl}]}$(cop-l) $\dfrac{E \xrightarrow{\mu} F}{E' \# E \xrightarrow{\mathtt{inr}\mu} F[\mathtt{inr}]}$(cop-r)
product	$\dfrac{E \xrightarrow{\mu} F}{E \,\&\, E' \xrightarrow{\langle \mu, \star \rangle} F \,\&\, E'}$(prod-l) $\dfrac{E \xrightarrow{\mu} F}{E' \,\&\, E \xrightarrow{\langle \star, \mu \rangle} E' \,\&\, F}$(prod-r)
synchronisation	$\dfrac{E \xrightarrow{\mu} F \qquad E' \xrightarrow{\nu} F'}{E \,\&\, E' \xrightarrow{\langle \mu, \nu \rangle} F \,\&\, F'}$(prod)
recursion	$\dfrac{E[x \mapsto \mathbf{rec}\, x.\, E] \xrightarrow{\mu} F}{\mathbf{rec}\, x.\, E \xrightarrow{\mu} F}$(rec)

Table 7.1: Axiomatic operational semantics for $\mathcal{L}$

in Table 7.1. The formulas over the bar in a rule are called *assumptions*, the one under the bar is the rule's *conclusion*. The system presented in Table 7.1 is referred to as $\mathcal{L}$ as well.

One of the main features of the Gentzen's system is clearly preserved in $\mathcal{L}$. In every rule from some assumptions made on atomic computations available from processes $E_1,\ldots,E_n$, for $n = \mathsf{arity}(\sigma)$, we are allowed to infer that an atomic computation from a more complex process $\sigma(E_1,\ldots,E_n)$ can take place, for some $\sigma \in \Sigma_n^{\mathcal{L}}$. In Gentzen's terminology our rules are *introduction rules*. Each of them corresponds to, *introduces*, a particular construct in $\mathcal{L}$, namely the outer one in the source of the change reported in the conclusion of the rule under consideration. For instance the rule (act-m) introduces the action operator, while the three rules (prod-l), (prod-r) and (prod) take care of the product operator.

As usual we consider *derivation trees* or *proofs* in $\mathcal{L}$. We write p,q: $\mathsf{Proof}_{\mathcal{L}}$ to denote the set of all proofs in $\mathcal{L}$. It is defined inductively as follows. For any proof rule in $\mathcal{L}$ with, say, n assumptions, and for any n proofs already in $\mathsf{Proof}_{\mathcal{L}}$, if the rule is *applicable* to the conclusions of the proofs then we can add to $\mathsf{Proof}_{\mathcal{L}}$ a new proof that has the result of an *application* as the conclusion, and the proofs as *branches*. We assume that each proof contains detailed information on the way it was built. In particular it is known which inference rules were used to create it. In the future we shall usually suppress these details for simplicity.

To formalise the idea we would need concepts like *unification*, definable by means of

known already substitution. But we hope that in our case it is obvious what happens, and refrain from making things formal this time. Instead, we offer several examples of proofs and non-proofs in $\mathcal{L}$.

Example 7.2.1 The following are examples of derivation trees:

Every instance of the action rule is a proof in $\mathcal{L}$. These are the building bricks of the proofs in $\mathcal{L}$. Only the trees with the 'empty leaves' form the proper proofs.

$$\frac{}{m\cdot E \xrightarrow{m} E}(\text{act-}m)$$

To act recursion must be unwound:

$$\dfrac{\dfrac{\dfrac{}{m\cdot \mathtt{rec}\, y.\, m\cdot y \xrightarrow{m} \mathtt{rec}\, y.\, m\cdot y}(\text{act}-m)}{\mathtt{rec}\, y.\, m\cdot y \xrightarrow{m} \mathtt{rec}\, y.\, m\cdot y}(\text{rec})}{\mathtt{rec}\, x.\, \mathtt{rec}\, z.\, m\cdot z \xrightarrow{m} \mathtt{rec}\, y.\, m\cdot y}(\text{rec})$$

Note, that the bound variable in the first term of the conclusion can be renamed as a result of substitution. Above $y = \mathrm{choiceVar}$ follows from the definition of substitution.

And finally a somewhat more elaborate example of a derivation tree. Observe that there are many simple actions contributing to the final activity.

$$\dfrac{\dfrac{\dfrac{}{m\cdot E \xrightarrow{m} E}}{m\cdot E \,\#\, F \xrightarrow{\mathtt{inl}\, m} E[\mathtt{inl}]} \quad \dfrac{\dfrac{}{m'\cdot E' \xrightarrow{m'} E'} \quad \dfrac{}{m''\cdot E'' \xrightarrow{m''} E''}}{m'\cdot E' \,\&\, m''\cdot E'' \xrightarrow{\langle m', m''\rangle} E' \,\&\, E''}}{(m\cdot E \,\#\, F) \,\&\, (m'\cdot E' \,\&\, m''\cdot E'') \xrightarrow{\langle \mathtt{inl}\, m, \langle m', m''\rangle\rangle} E[\mathtt{inl}] \,\&\, (E' \,\&\, E'')}$$

The following are not derivation trees in $\mathcal{L}$:

$$\dfrac{\dfrac{\dfrac{\vdots}{}}{\mathtt{rec}\, x.\, x \xrightarrow{\mu} \mathtt{rec}\, x.\, x}}{\mathtt{rec}\, x.\, x \xrightarrow{\mu} \mathtt{rec}\, x.\, x}$$

– The tree is infinite.

The tree with a non-empty leaf:

$$\dfrac{\mathtt{rec}\, x.\, x \xrightarrow{\mu} \mathtt{rec}\, x.\, x}{\mathtt{rec}\, x.\, x \xrightarrow{\mu} \mathtt{rec}\, x.\, x}$$

◇

We shall write $p \vdash E \xrightarrow{\mu} F$ to indicate that $E \xrightarrow{\mu} F$ is the conclusion of a proof p. We say in such situation that the proof p leads *from E to F via μ*. Notation $p \vdash E \rightarrow$ stands for $\exists F, \mu\,.\, p \vdash E \xrightarrow{\mu} F$. Also, $\mathcal{L} \vdash E \xrightarrow{\mu} F$ is used to indicate that $E \xrightarrow{\mu} F$ is the conclusion of some proof in $\mathcal{L}$.

Remark. In the standard presentation of inference systems one often introduces in place of rules with no assumptions a special class of formulas, called *axioms*. Only axioms are allowed later to become leaves of the proofs. We have decided against tradition for the sake of uniformity. Clearly, both approaches are equivalent.

Let us comment on the rules presented.

First, let us try to explain the role played by the silent action symbol in the rules.
In the rule **(res-L)** it is never the case, by definition, that $\imath \in L$. Hence,

$$\frac{E \xrightarrow{i} F}{E \setminus L \xrightarrow{i} F \setminus L}$$

is always an instance of (**res-**L).

The next rule, renaming, is crucial for it introduces the silent actions. In particular, if $R\,\mu\!\uparrow$, for some $\mu \in \mathcal{M}$, then the rule (**ren-**R) specialises to:

$$\frac{E \xrightarrow{\mu} F}{E[R] \xrightarrow{i} F[R]}.$$

It should help justify our decision to call the renaming operator also *hiding*. An action labelled with i is hidden from the influence of the outside world. The reason being that a silent action can be neither forced to, nor prevented from happening simultaneously with other actions. Let us try to explain what we mean by this.

Let us consider the following instance of the rule (**prod**) where $\nu \in \mathcal{M}$

$$\frac{E \xrightarrow{i} F \qquad E' \xrightarrow{\nu} F'}{E \,\&\, E' \xrightarrow{\langle \star, \nu \rangle} F \,\&\, F'}.$$

Perhaps the right way to look at it is to think that the rule (**prod**) enables two actions to happen *simultaneously*, while the rules (**prod-l**) and (**prod-r**) enable changes in one part of the product of two processes to take place without any interference from the other component. By placing the product in a context one can try to enforce simultaneous progress in both subprocesses, or ban it. This can be done only by using the restriction operator. A look at Table 7 will tell that only restriction can prohibit actions from subprocesses to be part of the activity of a process. Thus, if we restrict a product $E \,\&\, F$ with the help of $L = \{\langle \mu, \nu \rangle \mid \mu, \nu \in \mathcal{M}\}$, i.e., consider the process $(E \,\&\, F) \setminus L$ then the cooperation between E and F will be effectively forbidden. No action from E can take place simultaneously with any action from F.

With assumptions $\mathcal{L} \vdash E \xrightarrow{i} F$ and $\mathcal{L} \vdash E' \xrightarrow{\nu} F'$ the above rule shows that:

$$E \,\&\, E' \xrightarrow{\langle \star, \nu \rangle} F \,\&\, F',$$

is a possible evolution from $E \,\&\, E'$.

On the other hand we also can see the following evolution from the same process:

$$E \,\&\, E' \xrightarrow{i} F \,\&\, E' \xrightarrow{\langle \star, \nu \rangle} F \,\&\, F',$$

that is $E \,\&\, E'$ can first evolve silently to an intermediate process, and subsequently evolve to the same $F \,\&\, F'$ with the overall activity also observable as $\langle \star, \nu \rangle$. These changes can take place also in the opposite order.

In no way one can allow only one of those changes, and forbid the other by means of putting the process $E \,\&\, F$ in an appropriate context. Clearly, all operators give the same results when applied to both evolutions.

This exemplifies what was meant by freedom of a silent action to happen simultaneously with other actions, and freedom to take place by its own.

Second, there is a significant absence of a rule for the operator Ω. This means that the process Ω can exhibit no evolution at all.

Third, we see that the action rule, or more precisely action rule schema, is the only one that has no premises. The rules of this sort are the cornerstone of the construction of proofs in $\mathcal{L}$, as seen in the Example. This is because all the proofs in $\text{Proof}_{\mathcal{L}}$ are finite, and all of them have to have empty leaves.

Note, that only the elementary actions are allowed to mark the arrow of the conclusion in this rule.

Finally, as we have seen in the Example, the recursive process exhibits precisely the behaviour of its unfolding.

7.3 *CCS* as a Sublanguage of $\mathcal{L}$

The aim of this section is to show how a typical *CCS*-like language can be modelled in $\mathcal{L}$. The obvious choice is to take *CCS* itself. For the sake of simplicity we consider here *CCS* without value-passing. Later we shall see that this language brings all the technical problems, and weak points of the proposed solution into focus. We can therefore spend less time in the next section while considering other synchronisation techniques.

7.3.1 *CCS* – Syntax and Axiomatic Operational Semantics

In this subsection the language of Milner's *Calculus of Communicating Systems* is presented, as defined in [Milner 80], see also [Milner 83]. We shall refer to it as *CCS*. The reader is urged to consult the above sources for motivation of the choices made in developing the particular formalism. Otherwise, the terminology used might be difficult to understand.

In order to avoid clashes we have to introduce some minor changes in Milner's notation. We assume that a set Δ of ***channel names*** is given, with α, β ranging over it. Next, we take two copies of Δ to obtain $\lambda: \Lambda = \Delta + \Delta$ the set of ***channel names and conames.*** Elements of the first copy of Δ, again referred to as α, β, are called ***output ports***, while those from the second are ***input ports***, and carry a tag, e.g., $\overline{\alpha}$. Channels α and $\overline{\alpha}$ are called ***complementary.*** The set $m: M_{CCS}$ of ***elementary action symbols*** of *CCS* is defined as $M_{CCS} = \Lambda \cup \{\tau\}$, where $\tau \notin \Lambda$ is to denote ***unobservable action.***

The operation of taking a co-channel is extended onto entire M_{CCS}, and we put:

$$\overline{m} = \begin{cases} \alpha & m = \overline{\alpha} \\ \overline{\alpha} & m = \alpha \\ \tau & m = \tau \end{cases} .$$

To give the syntax of *CCS* we need to introduce just one more set. We define the set REN of ***renamings*** in *CCS* $R: \text{REN} \subseteq M_{CCS} \to M_{CCS}$, to be the set of all bijections of M_{CCS} that satisfy: $R(\overline{m}) = \overline{R(m)}$, and so $R(\tau) = \tau$ in particular.

action	$\dfrac{}{m\cdot E \xrightarrow{m} E}$(act-$m$) for $m \in M_{CCS}$
restriction	$\dfrac{E \xrightarrow{m} F}{E \setminus \alpha \xrightarrow{m} F \setminus \alpha}$(res-$\alpha$) for $\alpha \neq m \neq \overline{\alpha}$, $\alpha \in \Delta$
renaming	$\dfrac{E \xrightarrow{m} F}{E[R] \xrightarrow{Rm} F[R]}$(ren-$R$) for $R \in$ REN
sum	$\dfrac{E \xrightarrow{m} F}{E + E' \xrightarrow{m} F}$(+-l) $\dfrac{E \xrightarrow{m} F}{E' + E \xrightarrow{m} F}$(+-r)
parallel	$\dfrac{E \xrightarrow{m} F}{E \mid E' \xrightarrow{m} F \mid E'}$(|-l) $\dfrac{E \xrightarrow{m} F}{E' \mid E \xrightarrow{m} E' \mid F}$(|-r)
communication	$\dfrac{E \xrightarrow{\lambda} F \quad E' \xrightarrow{\overline{\lambda}} F'}{E \mid E' \xrightarrow{\tau} E' \mid F'}$(comm) for $\lambda \in \Lambda$
recursion	$\dfrac{E[x \mapsto \mathbf{rec}\, x.\, E] \xrightarrow{m} F}{\mathbf{rec}\, x.\, E \xrightarrow{m} F}$(rec)

Table 7.2: Axiomatic operational semantics for *CCS*

CCS – the Syntax

Now, we are ready to present the signature Σ^{CCS} of the language *CCS*.

We put $\Sigma^{CCS} = \Sigma_0^{CCS} \cup \Sigma_1^{CCS} \cup \Sigma_2^{CCS}$, where

$\Sigma_0^{CCS} = \{\mathtt{NIL}\}$ – ***inaction*** operator;

$\Sigma_1^{CCS} = \{m\cdot _ \mid m \in M_{CCS}\}$ – ***action*** operators;

$\cup \{_ \setminus \alpha \mid \alpha \in \Delta\}$ – ***restriction*** operators;

$\cup \{_[R] \mid R \in \mathrm{REN}\}$ – ***renaming*** operators;

$\Sigma_2^{CCS} = \{_ + _, _ \mid _\}$ – ***sum*** and ***communication*** operators.

Again, we write T_{CCS} rather than $\mathsf{T}_{\Sigma^{CCS}}$, and $\mathsf{Rec}_{CCS}(\mathsf{Var})$ for $\mathsf{Rec}_{\Sigma^{CCS}}(\mathsf{Var})$, etc. We use E, F to range over $\mathsf{Rec}_{CCS}(\mathsf{Var})$.

CCS – the Axiomatic Operational Semantics

The operational semantics of *CCS* is given in Table 7.2.

The notion of a derivation tree is the same as before. And as before by Proof_{CCS} we denote the set of all proofs derived using inference rules from the Table 7.2.

Ω			
$m\cdot_$,			for $m \in M_{CCS}$
$_ \setminus \{\alpha, \overline{\alpha}\}$,			for $\alpha \in \Delta$
$_[R^\dagger]$,			for $R \in \mathrm{REN}$
$_ +_{CCS} _$	$=$	$(_ \# _)[\mathrm{R}_+]$	
$_ \|_{CCS} _$	$=$	$((_ \,\&\, _) \setminus \mathrm{L}_\|)[\mathrm{R}_\|]$	
$_\mathtt{idl}$	$=$	$(_[\mathtt{inl}])[\mathrm{R}_+]$	
$_\mathtt{idr}$	$=$	$(_[\mathtt{inr}])[\mathrm{R}_+]$	

Table 7.3: Operators of CCS

7.3.2 CCS – a Sublanguage of $\mathcal{L}$

Our intention is to model CCS as a sublanguage of $\mathcal{L}$. To this end we need to choose a variety of derived operators of $\mathcal{L}$ with which to model CCS operators. But first of all we need to specialise the set of elementary action symbols of $\mathcal{L}$ to M_{CCS}. This gives us in turn the set $\mathcal{M}_{CCS} = \mathcal{M}(M_{CCS})$ of action symbols.

The operators of CCS are all collected in Table 7.3. There is an evident correspondence between operators of CCS and CCS, except for two new operators in the latter. This similarity is intentional, as we shall see later. The parameters that appear in the definitions of the CCS operators are defined below.

$\mathrm{R}_+ : M_{CCS} \to M_{CCS}$ is given by:

$$\mathrm{R}_+(\mu) = \begin{cases} \nu & \mu = \mathtt{inl}\,\nu \\ \nu & \mu = \mathtt{inr}\,\nu \end{cases}, \qquad \text{for } \mu \in M_{CCS}.$$

$\mathrm{L}_\| \subseteq M_{CCS}$ is the set given by:

$$\mathrm{L}_\| = \{\mu \in M_{CCS} \mid \forall m \in M_{CCS}.(\langle m, \star\rangle \neq \mu \neq \langle \star, m\rangle) \wedge \forall \lambda, \lambda' \in \Lambda.(\mu = \langle \lambda, \lambda'\rangle \Rightarrow \lambda \neq \overline{\lambda'})\}$$

$\mathrm{R}_\| : M_{CCS} \to M_{CCS}$ is given by:

$$\mathrm{R}_\|(\mu) = \begin{cases} \nu & \mu = \langle \nu, \star\rangle \\ \nu & \mu = \langle \star, \nu\rangle \\ \tau & \mu = \langle \lambda, \overline{\lambda}\rangle \end{cases}, \qquad \text{for } \mu \in M_{CCS},\ \lambda \in \Lambda.$$

For $R \in \mathrm{REN}$ we denote by $R^\dagger$ a formal extension of the total function $R : M_{CCS} \to M_{CCS}$ onto M_{CCS}, i.e., we take $R^\dagger$ to be the same set as R but this time we take M_{CCS} as both: domain and codomain.

An important observation is that we have both:

$$\mathrm{R}_+ \circ \mathtt{inl} = id_{M_{CCS}} \quad \text{and} \quad \mathrm{R}_+ \circ \mathtt{inr} = id_{M_{CCS}}. \tag{7.1}$$

The remaining part of this subsection is devoted to showing that the CCS has an operational semantics governed by inference rules derived from these of $\mathcal{L}$. For certain operators

like prefixing, restriction, renaming and recursion, this is obvious. For the other four we establish the derived inference rules in a series of lemmas.

Let us start from the sum operator in CCS.

Lemma 7.3.1 *(i) The following are derived rules of inference in $\mathcal{L}$:*

$$\frac{E \xrightarrow{\mu} F}{E +_{CCS} E' \xrightarrow{\mu} F\mathtt{idl}}(+_{CCS}\text{-l}) \quad \textit{and} \quad \frac{E \xrightarrow{\mu} F}{E' +_{CCS} E \xrightarrow{\mu} F\mathtt{idr}}(+_{CCS}\text{-r}).$$

(ii) If $\mathcal{L} \vdash E +_{CCS} E' \xrightarrow{\mu} F$ then $F = F'\mathtt{idl}$ or $F = F'\mathtt{idr}$ and any proof of $E +_{CCS} E' \xrightarrow{\mu} F$ in $\mathcal{L}$ can be seen as using $(+_{CCS}\text{-l})$ or $(+_{CCS}\text{-r})$ as the last rule, respectively.

Proof. (i) We give a derivation of rule $(+_{CCS}\text{-l})$ only. The derivation of the other one is symmetric.

$$\frac{\dfrac{E \xrightarrow{\mu} F}{E \# E' \xrightarrow{\mathtt{inl}\mu} F[\mathtt{inl}]}}{(E \# E')[\mathrm{R}_+] \xrightarrow{(\mathrm{R}_+ \circ \mathtt{inl})\mu} (F[\mathtt{inl}])[\mathrm{R}_+]}$$

but $(\mathrm{R}_+ \circ \mathtt{inl})\mu = id_{\mathcal{M}_{CCS}}\mu = \mu$, by (7.1), as required. Evidently, the order of rules applied to obtain the derivation was, in order of application, (cop-l) and (ren-R_+).

(ii) Let p be a proof of $E +_{CCS} E' \xrightarrow{\mu} F$ in $\mathcal{L}$. Since $E +_{CCS} E' = (E \# E')[\mathrm{R}_+]$ it follows from the inductive definition of the set $\mathrm{Proof}_{\mathcal{L}}$ that the last rule used to get p had to be (ren-R_+). Thus,

$$p = \frac{p'}{(E \# E')[\mathrm{R}_+] \xrightarrow{\mathrm{R}_+\mu'} F'[\mathrm{R}_+]}$$

for some $F' \in \mathrm{Rec}_{CCS}(\emptyset)$, and $\mu' \in \mathcal{M}_{CCS}$ or $\mu' = \imath$, s.t., $F'[\mathrm{R}_+] = F$, $\mu = \mathrm{R}_+(\mu')$ and $p' \vdash E \# E' \xrightarrow{\mu'} F'$.

Similarly, for p' we have that it had to be a result of an application of exactly one of the rules: (cop-l) and (cop-r). We can assume w.l.o.g., that it was the rule (cop-l) which was applied as the last one. Then

$$p' = \frac{p''}{E \# E' \xrightarrow{\mathtt{inl}\mu''} F''[\mathtt{inl}]}$$

for some $F'' \in \mathrm{Rec}_{CCS}(\emptyset)$, and $\mu'' \in \mathcal{M}_{CCS}$ or $\mu'' = \imath$, s.t., $\mu' = \mathtt{inl}\,\mu''$, $F' = F''[\mathtt{inl}]$ and $p'' \vdash E \xrightarrow{\mu''} F'$. But then $\mu = (\mathrm{R}_+ \circ \mathtt{inl})\mu'' = id''_\mu = \mu''$, by (7.1) as required.

Finally, we see that

$$p = \frac{p'' \vdash E \xrightarrow{\mu} F''}{E +_{CCS} E' \xrightarrow{\mu} F''\mathtt{idl}}$$

i.e., that p can be seen as having $(+_{CCS}\text{-l})$ as the last rule applied. □

Next, we come to the synchronisation operator in CCS. The situation now is slightly more complicated.

Lemma 7.3.2 *(i) The following are derived rules of inference in $\mathcal{L}$:*

$$\frac{E \xrightarrow{\lambda} F \qquad E' \xrightarrow{\overline{\lambda}} F'}{E|_{CCS}E' \xrightarrow{\tau} F|_{CCS}F'}(\text{comm-}CCS), \qquad \text{for } \lambda \in \Lambda$$

$$\frac{E \xrightarrow{\mu} F}{E|_{CCS}E' \xrightarrow{\mu} F|_{CCS}E'}(|_{CCS}\text{-l}) \qquad \frac{E \xrightarrow{\mu} F}{E'|_{CCS}E \xrightarrow{\mu} E'|_{CCS}F}(|_{CCS}\text{-r})$$

$$\frac{E \xrightarrow{\imath} F \qquad E' \xrightarrow{\mu} F'}{E|_{CCS}E' \xrightarrow{\mu} F|_{CCS}F'}(\imath\text{-l}) \qquad \frac{E \xrightarrow{\mu} F \qquad E' \xrightarrow{\imath} F'}{E|_{CCS}E' \xrightarrow{\mu} F|_{CCS}F'}(\imath\text{-r})$$

for $\mu \in M_{CCS}$ or $\mu = \imath$;

(ii) If $\mathcal{L} \vdash E|_{CCS}E' \xrightarrow{\mu} F''$ then $F'' = F|_{CCS}F'$, for some F, F'. If $p \vdash E|_{CCS}E' \xrightarrow{\mu} F|_{CCS}F'$ in $\mathcal{L}$ then p can be seen as using one of the above derived rules as the last one.

Before we start proving the last Lemma a remark that might be useful. We note that for $\mu = \imath$ the last two rules: ($\imath$-l) and ($\imath$-r), specialise to the same rule:

$$\frac{E \xrightarrow{\imath} F \qquad E' \xrightarrow{\imath} F'}{E|_{CCS}E' \xrightarrow{\imath} E'|_{CCS}F'}(\imath\text{-}\imath).$$

Later we shall see that those rules which cope with the silent action do not contribute to the operational semantics of CCS.

Proof.[of Lemma 7.3.2] (i) We present derivation of only one of the five inference rules in question. The remaining are left to the reader as an exercise. Let us take rule (comm-CCS) as an example. The following is its derivation in $\mathcal{L}$:

$$\frac{\dfrac{\dfrac{E \xrightarrow{\lambda} F \qquad E' \xrightarrow{\overline{\lambda}} F'}{E \,\&\, E' \xrightarrow{\langle \lambda, \overline{\lambda} \rangle} F \,\&\, F'}}{(E \,\&\, E') \setminus \mathrm{L}_| \xrightarrow{\langle \lambda, \overline{\lambda} \rangle} (F \,\&\, F') \setminus \mathrm{L}_|}}{((E \,\&\, E') \setminus \mathrm{L}_|)[\mathrm{R}_|] \xrightarrow{\tau} ((F \,\&\, F') \setminus \mathrm{L}_|)[\mathrm{R}_|]}$$

The first is an application of the rule (**prod**). Then we can apply (**res-**$\mathrm{L}_|$), since $\langle \lambda, \overline{\lambda} \rangle \notin \mathrm{L}_|$. The final is an application of the rule (**ren-**$\mathrm{R}_|$), and by definition of $\mathrm{R}_|$ we have $\mathrm{R}_|\langle \lambda, \overline{\lambda} \rangle = \tau$.

(ii) The argument follows the path established in the proof of 7.3.1(ii). Assume $p \vdash E|_{CCS}E' \xrightarrow{\mu} F$. Then $E|_{CCS}E' = ((E \,\&\, E') \setminus \mathrm{L}_|)[\mathrm{R}_|]$ and so (**ren-**$\mathrm{R}_|$) had to be the last rule applied. Therefore for some $F' \in \mathrm{Rec}_{\mathcal{L}}(\emptyset)$, $\mu' \in M_{CCS}$ or $\mu' = \imath$, s.t., $\mu = \mathrm{R}_|\mu'$ and $F = F'[\mathrm{R}_|]$ we have:

$$p = \frac{p'}{((E \,\&\, E') \setminus \mathrm{L}_|)[\mathrm{R}_|] \xrightarrow{\mu} F'[\mathrm{R}_|]}$$

for some $p' \in \mathrm{Proof}_{\mathcal{L}}$ s.t., $p' \vdash (E \,\&\, E') \setminus \mathrm{L}_| \xrightarrow{\mu'} F'$. As for the proof p' we deduce in turn that it ends with an application of (**res-**$\mathrm{L}_|$) as the last rule, and therefore for some $F'' \in \mathrm{Rec}_{\mathcal{L}}(\emptyset)$ s.t., $F' = F'' \setminus \mathrm{L}_|$, and $p'' \in \mathrm{Proof}_{\mathcal{L}}$ we have:

$$p' = \frac{p''}{(E \,\&\, E') \setminus \mathrm{L}_| \xrightarrow{\mu'} F'' \setminus \mathrm{L}_|}$$

where $p'' \vdash E \,\&\, E' \xrightarrow{\mu'} F''$ and $\mu' \notin \mathrm{L}_|$. At this stage we have to consider three cases. If the last rule applied to obtain p'' was **(prod)** then for some $F''', F'''' \in \mathrm{Rec}_{\mathcal{L}}(\emptyset)$ and $\nu, \nu' \in M_{CCS}$, $\nu = \lambda$ or $\nu' = \lambda$, s.t., $F'' = F''' \,\&\, F''''$, and $\mu' = \langle \nu, \nu' \rangle$ we have:

$$p'' = \frac{q \qquad q'}{E \,\&\, E' \xrightarrow{\mu'} F''' \,\&\, F''''}$$

for some proofs q, q' in $\mathrm{Proof}_{\mathcal{L}}$ s.t., $q \vdash E \xrightarrow{\nu} F'''$ and $q' \vdash E' \xrightarrow{\nu'} F''''$. If neither $\nu = \lambda$ nor $\nu' = \lambda$ then, from $\langle \nu, \nu' \rangle \notin \mathrm{L}_|$, it follows that for some $\lambda \in \Lambda$ we have: $\lambda = \nu$ and $\overline{\lambda} = \nu'$. But then $\mu = \mathrm{R}_| \mu' = \mathrm{R}_| \langle \lambda, \overline{\lambda} \rangle = \tau$. Moreover, $F = ((F''' \,\&\, F'''') \setminus \mathrm{L}_|)[\mathrm{R}_|] = F'''|_{CCS} F''''$, as required. We see that the last part of the proof p can be shortened with the help of the derived rule **(comm-*CCS*)**.

If $\nu = \lambda \neq \nu'$ then $\langle \nu, \nu' \rangle = \langle \star, \nu' \rangle \notin \mathrm{L}_|$, and $\mu = \mathrm{R}_| \langle \star, \nu' \rangle = \nu'$, so that $\nu' \in M_{CCS}$, by the definition of $\mathrm{L}_|$. Thus, the last part of the proof p can be abbreviated with the help of the rule **(λ-l)**. Similarly, in case $\nu' = \lambda$ it can be shown that the end of the proof p can be abbreviated with **(λ-r)**.

There remain two cases to consider: p'' ends with an application of **(prod-l)** or **(prod-r)**. One shows then that the last part of p is an expansion of the rule **($|_{CCS}$-l)**, resp., **($|_{CCS}$-r)**. The proofs are similar and left to the reader. □

And finally, we come to the derived inference rules for the two new, w.r.t., *CCS*, operators of *CCS*. Proofs of the next two Lemmas follow the pattern established already in Lemma 7.3.1 and Lemma 7.3.2, and are therefore left to the reader.

Lemma 7.3.3 *(i) The following is a derived inference rule in $\mathcal{L}$:*

$$\frac{E \xrightarrow{\mu} F}{E\mathtt{idl} \xrightarrow{\mu} F\mathtt{idl}}(\mathtt{idl})$$

(ii) If $\mathcal{L} \vdash E\mathtt{idl} \xrightarrow{\mu} F$ then $F = F'\mathtt{idl}$ for some $F' \in \mathrm{Rec}_{\mathcal{L}}(\emptyset)$, and any proof of $E\mathtt{idl} \xrightarrow{\mu} F$ in $\mathcal{L}$ can be seen as using **(idl)** *as the last rule.*

Proof. Omitted. □

Lemma 7.3.4 *(i) The following is a derived inference rule in $\mathcal{L}$:*

$$\frac{E \xrightarrow{\mu} F}{E\mathtt{idr} \xrightarrow{\mu} F\mathtt{idr}}(\mathtt{idr})$$

(ii) If $\mathcal{L} \vdash E\mathtt{idr} \xrightarrow{\mu} F$ then $F = F'\mathtt{idr}$ for some $F' \in \mathrm{Rec}_{\mathcal{L}}(\emptyset)$, and any proof of $E\mathtt{idr} \xrightarrow{\mu} F$ in $\mathcal{L}$ can be seen as using **(idr)** *as the last rule.*

Proof. Omitted. □

action $\dfrac{}{m.\,E \xrightarrow{m} E}(\text{act-}m)$ for $m \in M_{CCS}$

restriction $\dfrac{E \xrightarrow{m} F}{E \setminus \{\alpha,\overline{\alpha}\} \xrightarrow{m} F \setminus \{\alpha,\alpha\}}(\text{res-}\{\alpha,\overline{\alpha}\})$ for $\alpha \neq m \neq \overline{\alpha}$, $\alpha \in \Delta$

renaming $\dfrac{E \xrightarrow{m} F}{E[R^\dagger] \xrightarrow{R^\dagger m} F[R^\dagger]}(\text{ren-}R^\dagger)$ for $R \in \text{REN}$

sum $\dfrac{E \xrightarrow{m} F}{E +_{CCS} E' \xrightarrow{m} F\mathtt{idl}}(+_{CCS}\text{-l})$ $\dfrac{E \xrightarrow{m} F}{E' +_{CCS} E \xrightarrow{m} F\mathtt{idr}}(+_{CCS}\text{-r})$

parallel $\dfrac{E \xrightarrow{m} F}{E|_{CCS}E' \xrightarrow{m} F|_{CCS}E'}(|_{CCS}\text{-l})$ $\dfrac{E \xrightarrow{m} F}{E'|_{CCS}E \xrightarrow{m} E'|_{CCS}F}(|_{CCS}\text{-r})$

communication $\dfrac{E \xrightarrow{\lambda} F \quad E' \xrightarrow{\overline{\lambda}} F'}{E|_{CCS}E' \xrightarrow{\tau} E'|_{CCS}F'}(\text{comm-}CCS)$ for $\lambda \in \Lambda$

idle operators $\dfrac{E \xrightarrow{m} F}{E\mathtt{idl} \xrightarrow{m} F\mathtt{idl}}(\text{idl})$ $\dfrac{E \xrightarrow{m} F}{E\mathtt{idr} \xrightarrow{m} F\mathtt{idr}}(\text{idr})$

recursion $\dfrac{E[x \mapsto \mathbf{rec}\,x.\,E] \xrightarrow{m} F}{\mathbf{rec}\,x.\,E \xrightarrow{m} F}(\text{rec})$

Table 7.4: Axiomatic operational semantics for CCS

Operational Semantics of CCS

Throughout the last paragraph we have been using CCS-terms as a notational device, or a mask that hides the *real* $\mathcal{L}$-terms that lay beneath them. Thus, we have been thinking in terms of the result of a translation of CCS-terms into $\mathcal{L}$-terms by substituting the definitions for operators of CCS.

It will be helpful to introduce a formal notation for this translation. For $E \in \text{Rec}_{CCS}(\text{Var})$ we write $\overline{\overline{E}}$ to denote the result of expanding the definitions of the derived operators. For instance, $\overline{\overline{x}} = x$, for $x \in \text{Var}$, $\overline{\overline{E +_{CCS} F}} = (\overline{\overline{E}} \,\#\, \overline{\overline{F}})[\text{R}_+]$, and so on.

The axiomatic operational semantics of CCS is presented in Table 7.4. In the following Theorem we present two results that give a real meaning to our claim that "CCS has its operational semantics derived from that of $\mathcal{L}$".

The first part of the Theorem shows that the above translation map extends onto derivation trees of CCS in the unique sensible way. The second part says that any derivation in $\mathcal{L}$ of an atomic action starting from a translation of a CCS process is an image of a proof in CCS via this translation. In this sense the operational semantics of $\mathcal{L}$ is a *conservative* extension if the operational semantics of CCS. It does not introduce any new one-step computations

to the (images of) processes in CCS.

Theorem 7.3.5 *There exists a unique function* $\overline{_}: \mathsf{Proof}_{CCS} \to \mathsf{Proof}_{\mathcal{L}}$ *s.t.:*

(i) For $E, F \in \mathsf{Rec}_{CCS}(\emptyset)$, $m \in M_{CCS}$ *and* $p \in \mathsf{Proof}_{CCS}$*:*

$$p \vdash E \xrightarrow{m} F \text{ in } CCS \quad \textit{implies} \quad \overline{\overline{p}} \vdash \overline{\overline{E}} \xrightarrow{m} \overline{\overline{F}} \text{ in } \mathcal{L};$$

(ii) For $E \in \mathsf{Rec}_{CCS}(\emptyset)$, $F \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)$, $\mu = \lambda$ *or* $\mu \in M_{CCS}$ *and* $q \in \mathsf{Proof}_{\mathcal{L}}$*:*

$$q \vdash \overline{\overline{E}} \xrightarrow{\mu} F \quad \textit{implies} \quad \lambda \neq \mu \in M_{CCS}, \textit{ and } \exists p \in \mathsf{Proof}_{CCS} . \overline{\overline{p}} = q.$$

Proof. First, we define a function that satisfies condition (i).

The definition is inductive. Suppose $p \vdash E \xrightarrow{m} F$ in CCS. In cases other than: $E = E' +_{CCS} E''$, $E = E' |_{CCS} E''$, $E = E'\mathtt{idl}$ and $E = E'\mathtt{idr}$ we see that the last rule applied is one of the rules of $\mathcal{L}$. We have reduced the size of the problem, and so the result follows by induction.

In one of the four case mentioned above we replace the last inference rule applied to obtain p by appropriate derivation obtained in one of the Lemma 7.3.1(i), Lemma 7.3.2(i), Lemma 7.3.3(i) and Lemma 7.3.4(i). The remaining derivation tree(s) are of smaller size, and so they have translations that satisfy (i). The result follows by induction.

Now, we show that $\overline{_}$ is the unique function that satisfies both conditions. The proof goes by structural induction again. The only difference is that this time we use the second parts of the Lemmas. □

7.3.3 CCS versus CCS

Finally, we relate the original language of CCS together with its operational semantics to the sublanguage CCS of $\mathcal{L}$. To this end two maps are defined that translate terms of one language to the terms of the other. Only one of these constructions on terms extends to a translation of derivation trees of the operational semantics of the two languages.

We start with embedding CCS into CCS. To cope with recursion we have to translate open terms, not only the closed ones.

Definition 7.3.6 $\lceil _ \rceil: \mathsf{Rec}_{CCS}(\mathsf{Var}) \to \mathsf{Rec}_{CCS}(\mathsf{Var})$ is defined inductively as follows. For $x \in \mathsf{Var}$, $m \in M_{CCS}$, $\alpha \in \Delta$, $R \in \mathrm{REN}$ and $E, E' \in \mathsf{Rec}_{CCS}(\mathsf{Var})$ we put:

$\lceil x \rceil = x$;
$\lceil \mathtt{NIL} \rceil = \Omega$;
$\lceil m \cdot E \rceil = m \cdot \lceil E \rceil$;
$\lceil E \setminus \alpha \rceil = \lceil E \rceil \setminus \{\alpha, \overline{\alpha}\}$;
$\lceil E[R] \rceil = \lceil E \rceil [R^{\dagger}]$;
$\lceil E + E' \rceil = \lceil E \rceil +_{CCS} \lceil E' \rceil$;
$\lceil E \mid E' \rceil = \lceil E \rceil |_{CCS} \lceil E' \rceil$;
$\lceil \mathbf{rec}\, x.\, E \rceil = \mathbf{rec}\, \lceil x \rceil . \lceil E \rceil$.

Recall that $R^\dagger$ in the clause for renaming denotes a partial function $R^\dagger: M_{CCS} \rightharpoonup M_{CCS}$ defined on M_{CCS} only, and there equal to R.

Example 7.3.7 It is not the case that the above embedding extends onto proofs. Consider the unique atomic computation of process $a\cdot b\cdot \mathtt{NIL} + \mathtt{NIL}$ in CCS:

$$\frac{\dfrac{}{a\cdot b\cdot \mathtt{NIL} \xrightarrow{a} b\cdot \mathtt{NIL}}}{a\cdot b\cdot \mathtt{NIL} + \mathtt{NIL} \xrightarrow{a} b\cdot \mathtt{NIL}}.$$

And now compare it to the unique atomic computation that originates in its translation:

$$\frac{\dfrac{}{a\cdot b\cdot \Omega \xrightarrow{a} b\cdot \Omega}}{a\cdot b\cdot \Omega +_{CCS} \Omega \xrightarrow{a} (b\cdot \Omega)\mathtt{idl}}.$$

We see that in general $CCS \vdash E \xrightarrow{m} F$ does not imply $\mathcal{CCS} \vdash \lceil E \rceil \xrightarrow{m} \lceil F \rceil$. ◇

And here is the translation going in the other direction.

Definition 7.3.8 $\lfloor _ \rfloor: \mathsf{Rec}_{CCS}(\mathsf{Var}) \to \mathsf{Rec}_{\mathcal{CCS}}(\mathsf{Var})$ is defined inductively as follows. For $x \in \mathsf{Var}$, $m \in M_{CCS}$, $\alpha \in \Delta$, $R \in \mathrm{REN}$ and $E, E' \in \mathsf{Rec}_{CCS}(\mathsf{Var})$ we put:

$\lfloor x \rfloor = x$;
$\lfloor \Omega \rfloor = \mathtt{NIL}$;
$\lfloor m\cdot E \rfloor = m\cdot \lfloor E \rfloor$;
$\lfloor E \setminus \{\alpha, \overline{\alpha}\} \rfloor = \lfloor E \rfloor \setminus \alpha$;
$\lfloor E[R^\dagger] \rfloor = \lfloor E \rfloor [R]$;
$\lfloor E +_{CCS} E' \rfloor = \lfloor E \rfloor + \lfloor E' \rfloor$;
$\lfloor E |_{CCS} E' \rfloor = \lfloor E \rfloor \,|\, \lfloor E' \rfloor$;
$\lfloor E\mathtt{idl} \rfloor = \lfloor E \rfloor$;
$\lfloor E\mathtt{idr} \rfloor = \lfloor E \rfloor$;
$\lfloor \mathtt{rec}\, x.\, E \rfloor = \mathtt{rec}\, \lfloor x \rfloor .\, \lfloor E \rfloor$.

In order to relate operational semantics of CCS and $\mathcal{CCS}$, and more specifically, to cope with recursion, we need the following lemma that relates the translations to the substitution. First, we note that the translations extend onto substitutions. We consider the case of $\lfloor _ \rfloor$ only, the other one is symmetric.

Given a substitution $\varrho: \mathsf{Var} \to \mathsf{Rec}_{CCS}(\mathsf{Var}) = \mathsf{Sub}_{CCS}$ we define $\lfloor \varrho \rfloor: \mathsf{Var} \to \mathsf{Rec}_{\mathcal{CCS}}(\mathsf{Var})$ by: $\lfloor \varrho \rfloor = \lfloor _ \rfloor \circ \varrho$, i.e., $(\lfloor \varrho \rfloor)x = \lfloor \varrho x \rfloor$, for $x \in \mathsf{Var}$.

The definitions of functions FV and new can be found in Chapter 6.

Lemma 7.3.9 *For $E \in \mathsf{Rec}_{CCS}(\mathsf{Var})$, $F \in \mathsf{Rec}_{\mathcal{CCS}}(\mathsf{Var})$, $x \in \mathsf{Var}$, $\varrho \in \mathsf{Sub}_{CCS}$ and $\varrho' \in \mathsf{Sub}_{\mathcal{CCS}}$ the following holds:*

(i) $\mathsf{FV}(E) = \mathsf{FV}(\lfloor E \rfloor)$ *and* $\mathsf{FV}(F) = \mathsf{FV}(\lceil F \rceil)$;

(ii) $\mathsf{new}\, x\, E\, \varrho = \mathsf{new} \lfloor x \rfloor \lfloor E \rfloor \lfloor \varrho \rfloor$ *and* $\mathsf{new}\, x\, F\, \varrho' = \mathsf{new} \lceil x \rceil \lceil F \rceil \lceil \varrho' \rceil$;

(iii) $\lfloor E\varrho \rfloor = \lfloor E \rfloor \lfloor \varrho \rfloor$ *and* $\lceil F\varrho' \rceil = \lceil F \rceil \lceil \varrho' \rceil$.

Proof. (i) By structural induction on E.

(ii) Follows from (i) and the definition of the function new in the definition of substitution, see Definition 6.1.4.

(iii) By structural induction on E, resp., F. Only one translation is considered. The proof for the other is virtually identical. We concentrate on $\lfloor _ \rfloor$, and consider only one of the many trivial cases as an example, and then turn attention to the one not entirely trivial.

Assume $E = E' +_{CCS} E''$. Then

$$\begin{array}{lcll} \lfloor (E' +_{CCS} E'')\varrho \rfloor & = & \lfloor E'\varrho +_{CCS} E''\varrho \rfloor & \text{by def., of substitution} \\ & = & \lfloor E'\varrho \rfloor + \lfloor E''\varrho \rfloor & \text{by def., of } \lfloor _ \rfloor \\ & = & \lfloor E' \rfloor \lfloor \varrho \rfloor + \lfloor E'' \rfloor \lfloor \varrho \rfloor & \text{twice by induction} \\ & = & (\lfloor E' \rfloor + \lfloor E'' \rfloor)\lfloor \varrho \rfloor & \text{by def., of substitution} \\ & = & \lfloor E' + E'' \rfloor \lfloor \varrho \rfloor & \text{by def., of } \lfloor _ \rfloor. \end{array}$$

Now, assume $E = \mathbf{rec}\, x.\, F$. Then we can calculate as follows:

$$\begin{array}{lcll} \lfloor (\mathbf{rec}\, x.\, F)\varrho \rfloor & = & \lfloor \mathbf{rec}\, y.\, F\varrho[x \mapsto y] \rfloor & \text{by def., of substitution} \\ & = & \mathbf{rec}\, y.\, \lfloor F\varrho[x \mapsto y] \rfloor & \text{by def., of } \lfloor _ \rfloor \\ & = & \mathbf{rec}\, y.\, \lfloor F \rfloor \lfloor \varrho[x \mapsto y] \rfloor & \text{by induction} \\ & = & \mathbf{rec}\, y.\, \lfloor F \rfloor \lfloor \varrho \rfloor [\lfloor x \rfloor \mapsto y] & \text{by def., of } \lfloor _ \rfloor \\ & = & (\mathbf{rec}\, \lfloor x \rfloor.\, \lfloor F \rfloor)\lfloor \varrho \rfloor & \text{by def., of substitution} \\ & = & \lfloor \mathbf{rec}\, x.\, F \rfloor \lfloor \varrho \rfloor & \text{by def., of } \lfloor _ \rfloor. \end{array}$$

The property (ii) is used at second last of the above stages. □

The next result shows how atomic changes, i.e., their proofs, corresponding to a term E in one language are related to the changes corresponding to its translation.

Theorem 7.3.10 *(i) Both* $\lceil _ \rceil$ *and* $\lfloor _ \rfloor$, *send closed terms to closed terms.*

(ii) For $E \in \mathsf{Rec}_{CCS}(\mathsf{Var})$ *we have:* $\lfloor \lceil E \rceil \rfloor = E$.

(iii) For $E, F \in \mathsf{Rec}_{CCS}(\emptyset)$, $m \in M_{CCS}$, $p \in \mathsf{Proof}_{CCS}$:
$p \vdash E \xrightarrow{m} F$ *implies* $\lfloor p \rfloor \vdash \lfloor E \rfloor \xrightarrow{m} \lfloor F \rfloor$, *for some* $\lfloor p \rfloor \in \mathsf{Proof}_{CCS}$.

(iv) For $E \in \mathsf{Rec}_{CCS}(\emptyset)$, $F \in \mathsf{Rec}_{CCS}(\emptyset)$, $m \in M_{CCS}$, $q \in \mathsf{Proof}_{CCS}$:
$q \vdash \lfloor E \rfloor \xrightarrow{m} F$ *implies* $\exists p \in \mathsf{Proof}_{CCS} \exists F' \in \mathsf{Rec}_{CCS}(\emptyset)\,.\, \lfloor F' \rfloor = F \wedge p \vdash E \xrightarrow{m} F'$.

Proof. (i) Follows immediately from Lemma 7.3.9(i).

(ii) By structural induction on E.

(iii) We show by structural induction that the translation map $\lfloor _ \rfloor$ can be extended to a map $\lfloor _ \rfloor : \mathsf{Proof}_{CCS} \to \mathsf{Proof}_{CCS}$ in such a way that if $p \vdash E \xrightarrow{m} F$ in CCS then $\lfloor p \rfloor \vdash \lfloor E \rfloor \xrightarrow{m} \lfloor F \rfloor$ in CCS.

Let $p \vdash E \xrightarrow{m} F$ in CCS. There are eight main cases to consider, according to the last rule applied in the construction of p. Most of them follow the same pattern. Among those the case when (rec) was the last rule is perhaps the most difficult.

So, assume $E = \mathbf{rec}\, x.\, E'$. Then for some $p' \in \mathsf{Proof}_{CCS}$ we have:

$$p = \frac{p' \vdash E'[x \mapsto E] \xrightarrow{m} F}{E \xrightarrow{m} F}.$$

Then $\lfloor p' \rfloor \vdash \lfloor E'[x \mapsto E] \xrightarrow{m} F \rfloor$, by induction hypothesis, and $\lfloor E'[x \mapsto E] \rfloor = \lfloor E' \rfloor \{\lfloor x \rfloor \mapsto \lfloor E \rfloor\}$, by Lemma 7.3.9(iii). Thus, we can apply (rec) to $\lfloor p' \rfloor$ to end up with a proof $\lfloor p \rfloor$ s.t., $\lfloor p \rfloor \vdash \mathbf{rec}\, \lfloor x \rfloor.\, \lfloor E' \rfloor \xrightarrow{m} \lfloor F \rfloor$. But then $\lfloor E \rfloor = \mathbf{rec}\, \lfloor x \rfloor.\, \lfloor E' \rfloor$, as required.

There are two cases that do not quite fall into the pattern discussed above. They correspond to the sum, and the idle operators.

Assume, w.l.o.g., that ($+_{CCS}$-l) was the last rule of p. Then for some $p' \in \mathsf{Proof}_{CCS}$, $E', E'', F' \in \mathsf{Rec}_{CCS}(\emptyset)$ s.t., $E = E' +_{CCS} E''$, $F = F'\mathtt{idl}$ and we have:

$$p = \frac{p' \vdash E' \xrightarrow{m} F'}{E \xrightarrow{m} F}.$$

By induction we get $\lfloor p' \rfloor \vdash \lfloor E' \rfloor \xrightarrow{m} \lfloor F' \rfloor$. Moreover, $\lfloor E \rfloor = \lfloor E' \rfloor + \lfloor E'' \rfloor$ and $\lfloor F \rfloor = \lfloor F' \rfloor$. So, a proof $\lfloor p \rfloor$ with required properties can be built from $\lfloor p' \rfloor$ by an application of the rule (+-l) of CCS.

As for the idle operators the solution is obvious. Assume, w.l.o.g., that $E = E'\mathtt{idl}$. Then for some $F' \in \mathsf{Rec}_{CCS}(\emptyset)$, and $p' \in \mathsf{Proof}_{CCS}$ s.t., $F = F'\mathtt{idl}$, we have:

$$p = \frac{p' \vdash E' \xrightarrow{m} F'}{E \xrightarrow{m} F}.$$

But $\lfloor E \rfloor = \lfloor E' \rfloor$ and $\lfloor F \rfloor = \lfloor F' \rfloor$, so we can put $\lfloor p \rfloor = \lfloor p' \rfloor$.

(iv) Left to the reader. This time from a proof $q \vdash \lfloor E \rfloor \xrightarrow{m} F$ in CCS one should recover F' and a proof $p \vdash E \xrightarrow{m} F'$ in CCS. This, obviously, is best done inductively. □

Let us summarise briefly the results of this section. We have started with Milner's CCS. Next, we have chosen sublanguage CCS of $\mathcal{L}$. This sublanguage was shown to have an operational semantics derivable from the semantics of $\mathcal{L}$. It is related with CCS via a pair of maps. One of them embeds CCS into CCS. The other translates not only terms, but derivation trees from CCS to CCS.

7.3.4 CCS without τ

It is tempting to give a different reinterpretation of CCS as a sublanguage of $\mathcal{L}$. After all we have introduced the silent action symbol for the same purpose that τ was introduced to CCS in the first place.

The changes required seem evident.

First, we redefine $\mathrm{R}_|\colon M_{CCS} \to M_{CCS}$. Let

$$\mathrm{R}_| \mu = \nu \quad \text{if } \mu = \langle \nu, \star \rangle \text{ or } \mu = \langle \star, \nu \rangle, \quad \text{for } \mu \in M_{CCS}.$$

That is the change amounts to leaving $R_{|}\langle\lambda,\overline{\lambda}\rangle$ undefined, for $\lambda \in \Lambda$. With minor modifications to the proof we obtain the following variant of Lemma 7.3.2.

Lemma 7.3.11 *(i) The following are derived rules of inference in $\mathcal{L}$:*

$$\frac{E \xrightarrow{\lambda} F \qquad E' \xrightarrow{\overline{\lambda}} F'}{E|_{CCS}E' \xrightarrow{\imath} F|_{CCS}F'}(\text{comm-}\imath), \qquad \text{for } \lambda \in \Lambda$$

$$\frac{E \xrightarrow{\mu} F}{E|_{CCS}E' \xrightarrow{\mu} F|_{CCS}E'}(|_{CCS}\text{-l}) \qquad \frac{E \xrightarrow{\mu} F}{E'|_{CCS}E \xrightarrow{\mu} E'|_{CCS}F}(|_{CCS}\text{-r})$$

$$\frac{E \xrightarrow{\imath} F \qquad E' \xrightarrow{\mu} F'}{E|_{CCS}E' \xrightarrow{\mu} F|_{CCS}F'}(\imath\text{-l}) \qquad \frac{E \xrightarrow{\mu} F \qquad E' \xrightarrow{\imath} F'}{E|_{CCS}E' \xrightarrow{\mu} F|_{CCS}F'}(\imath\text{-r})$$

for $\mu \in M_{CCS}$ or $\mu = \imath$;

(ii) If $\mathcal{L} \vdash E|_{CCS}E' \xrightarrow{\mu} F''$ then $F'' = F|_{CCS}F'$, for some F, F'. If $p \vdash E|_{CCS}E' \xrightarrow{\mu} F|_{CCS}F'$ in $\mathcal{L}$ then p can be seen as using one of the above derived rules as the last one.

Proof. Omitted. □

Let us take the remaining operators as before, with minor modifications described later. The derived axiomatic operational semantics of the resulting sublanguage has an expected property — only chanel names, their conames, and $\imath$ appear as labels in the conclusions of the inference rules. This axiomatic system is very similar to the one presented in Table 7.4. The main difference is that instead of three inference rules to cope with parallel composition operator in the latter, we have to admit all five derived rules from Lemma 7.3.11. Unlike proviously, this time the silent action symbol is a legitimate part of CCS.

For this reason we cannot claim that this sublanguage models correctly the original language. Unlike τ in CCS the silent action in this version of CCS inherits from $\mathcal{L}$ its freedom to refuse to synchronise, and more importantly to synchronise with other actions.

Perhaps it is better to return to our earlier explanation, and insist that the freedom of silent actions mentioned above refers to simultaneous occurrence with other actions. Then the synchronisation can be underestood to be a method of enforcing and/or prohibiting simultaneous occurrences of actions. From such perspective it seems reasonable, and also natural to exempt silent actions from from the influence of any synchronisation mechanism.

We have mentioned before that there is another minor change required in the definition of this modified CCS. The point is that in $\mathcal{L}$ there is no construction like $\imath\cdot_$ to correspond to $\tau\cdot_$ in CCS. Tacitly, we have taken the same alphabet M_{CCS} of elementary action symbols as before. This, obviously includes τ, but in fact any auxilary symbol would do. To ensure that τ never appears as a label in the conclusion we allow only prefixing with elementary actions other than τ. To model operator $\tau\cdot_$ from CCS a derived operator $\imath\cdot_ = (\tau\cdot_)[\imath]$ is introduced. By abuse of notation $\imath: M_{CCS} \to M_{CCS}$ is given by:

$$\imath\,\mu = \mu \quad \text{if } \mu \in \Lambda, \quad \text{for } \mu \in M_{CCS}.$$

It is immediate to see that the following is the only derived inference rule for this derived prefixing operator:

$$\frac{}{¿\cdot E \xrightarrow{¿} E[¿]}.$$

Also, any proof in $\mathcal{L}$ with $¿\cdot E$ as the source of the change reported in the conclusion can be seen as resulting from an application of the above rule.

7.4 Modelling in $\mathcal{L}$ Operators from Other Languages

We end this Chapter with rather sketchy account on the modelling in $\mathcal{L}$ of other, mainly synchronisation primitives from languages other than CCS.

We consider in turn synchronisation operators from Milner's $SCCS$, Hoare's $TCSP$, and finally those definable by means of Winskel's synchronisation algebras. We end this section and Chapter with some remarks on the shortcomings of the presented approach.

7.4.1 $SCCS$ – a Synchronous Communication Principle

Our account of $SCCS$, i.e., *Synchronous CCS* is based on [Milner 83]. This paper contains an example of a simulation of one language within another. In fact it is asynchronous CCS that is being modeled in $SCCS$ there.

To cope with $SCCS$ in a way similar to what we have done for CCS we need different assumptions about the nature of the set of elementary action symbols of $\mathcal{L}$. After [Milner 83] we take M_{SCCS} to be an *abelian monoid*, i.e., $M_{SCCS} = \langle M_{SCCS}, \times, 1\rangle$. There $_ \times _ : M^2_{SCCS} \to M_{SCCS}$ is a binary operation called *product of actions* that is *associative*, i.e., $(m \times m') \times m'' = m \times (m' \times m'')$ and *commutative*, i.e., $m \times m' = m' \times m$. Finally, $1 \in M_{SCCS}$ is a unit of the product. In the future we omit '$\times$', and write mm' rather than $m \times m'$.

In place of '$_ \mid _$' in CCS we have a new binary operator $_ \times _$ in $SCCS$. There is only one inference rule in place of the previous three:

$$\frac{E \xrightarrow{m} E' \qquad F \xrightarrow{m'} F'}{E \times F \xrightarrow{mm'} E' \times F'}(\times)$$

The other operators of $SCCS$ and their rules are identical with those in CCS.

The set REN of renamings comprises now *automorphisms* of M_{SCCS}, i.e., $R \in$ REN iff $R\colon M_{SCCS} \to M_{SCCS}$ is a bijection s.t., $R1 = 1$ and $R(mm') = (Rm)(Rm')$.

We cope with modeling the 'tight' synchronisation mechanism from $SCCS$ in $\mathcal{L}$ in the way that is only a minor modification of the way we have tackled the 'loose' synchronisation from CCS. We define a derived operator $_ \times_{SCCS} _$ in $\mathcal{L}$ and show it has derived operational meaning that coincides with the meaning of $_ \times _$.

We put

$$_ \times_{SCCS} _ = ((_ \,\&\, _) \setminus \mathrm{L}_\times)[\mathrm{R}_\times]$$

where the parameters of the above definition are defined as follows:

$\mathrm{L}_\times \subseteq M_{CCS}$ is the set given by:

$$L_\times = \{\mu \in M_{SCCS} \mid \mu = \langle \nu, \nu' \rangle \Rightarrow (\nu \notin M_{SCCS} \vee \nu' \notin M_{SCCS})\}$$

$R_\times : M_{CCS} \rightharpoonup M_{CCS}$ is given by:

$$R_\times(\langle m, m' \rangle) = mm', \qquad \text{for } m, m' \in M_{CCS}.$$

As before for $R \in \mathrm{REN}$ we denote by $R^\dagger$ a formal extension of the total function $R: M_{SCCS} \to M_{SCCS}$ onto M_{SCCS}.

Lemma 7.4.1 *(i) The following are derived rules of inference in $\mathcal{L}$:*

$$\frac{E \xrightarrow{m} E' \qquad F \xrightarrow{m'} F'}{E \times_{SCCS} F \xrightarrow{mm'} E' \times_{SCCS} F'}(\times_{SCCS}) \qquad \frac{E \xrightarrow{i} E' \qquad F \xrightarrow{i} F'}{E \times_{SCCS} F \xrightarrow{i} E' \times_{SCCS} F'}(i\text{-}i)$$

(ii) If $\mathcal{L} \vdash E \times_{SCCS} E' \xrightarrow{\mu} F''$ then $\mu \in M_{SCCS}$ or $\mu = i$ and $F'' = F \times_{SCCS} F'$, for some F, F', and any proof of $E \times_{SCCS} E' \xrightarrow{\mu} F \times_{SCCS} F'$ in $\mathcal{L}$ can be seen as using one of the above derived rules as the last one.

Proof. Omitted. □

Now, one can select a sublanguage $SCCS$ of $\mathcal{L}$, just as we have done for CCS, and show that the relationship between $SCCS$ and $SCCS$ is the same as the one between CCS and CCS. The details are left to the reader.

7.4.2 *TCSP*

In [Hoare et. al. 84], see also [Hoare 85], a variety of operators for building process is defined. For now we concentrate on those among them which are related to cooperation between processes. There are two such operators. Our presentation is somewhat different from the one given by Hoare et. al. The reason is that for Hoare and his collegues it is the denotational semantics that carries the explanation of the meaning of a syntactic construct.

This is in the line with the mathematical tradition in Oxford that stems from the work of Scott and Strachey on the semantics of programming languages. In this Chapter the operational description of an operator is taken as primitive, following the school of thought developed in Edinburgh.

Our account is based almost entirely on the first of the above mentioned sources. From our perspective the so called 'laws' ibid., are almost what we want for the operational description of the operators.

Hoare's book, [Hoare 85], is written with a particular denotational model in mind. It is therefore rather difficult to guess what kind of operational intuitions, if any, the author would accept as the right ones.

Parallel composition by intersection: $_ \parallel _$. This operator has its meaning given by one rule only:

$$\frac{E \xrightarrow{m} F \qquad E' \xrightarrow{m} F'}{E \parallel F \xrightarrow{m} E' \parallel F'}(\parallel)$$

Thus, as [Hoare et. al. 84] put it, "an event can occur only when E and F can participate in it simultaneously".

Parallel composition by interleaving: $_ \| _$. In some sense it is an opposite to the composition by intersection operator. Here, $E \| E'$ evolves by a change in one of E and E', and only one of them. Hence the rules:

$$\frac{E \xrightarrow{m} F}{E \| E' \xrightarrow{m} F \| E'}(\|\text{-l}) \qquad \frac{E' \xrightarrow{m} F'}{E \| E' \xrightarrow{m} E \| F'}(\|\text{-r}).$$

We model Hoare's operators in $\mathcal{L}$ in what should be now a standard manner. First, note that there are no assumptions on the structure of the set of elementary action symbols M. Define

$$_ \|_{TCSP} _ = ((_ \,\&\, _) \setminus \mathrm{L}_{\|})[\mathrm{R}_{\|}], \quad \text{and} \quad _ \|_{TCSP} _ = ((_ \,\&\, _) \setminus \mathrm{L}_{\|})[\mathrm{R}_{\|}]$$

where

$\mathrm{L}_{\|} \subseteq \mathcal{M}$ is the set given by:

$$\mathrm{L}_{\|} = \{\mu \in \mathcal{M} \mid \forall m, m' \in M. \langle m, m' \rangle = \mu \Rightarrow m \neq m'\}$$

$\mathrm{R}_{\|}: \mathcal{M} \rightharpoonup \mathcal{M}$ is given by:

$$\mathrm{R}_{\|}(\mu) = m, \quad \text{if } \mu = \langle m, m \rangle, \quad \text{for } \mu \in \mathcal{M},\ m \in M.$$

$\mathrm{L}_{\|} \subseteq \mathcal{M}$ is the set given by:

$$\mathrm{L}_{\|} = \mathcal{M} - \{\langle m, \star \rangle \mid m \in M\} - \{\langle \star, m \rangle \mid m \in M\}$$

$\mathrm{R}_{\|}: \mathcal{M} \rightharpoonup \mathcal{M}$ is given by:

$$\mathrm{R}_{\|}(\mu) = m, \quad \text{if } \mu = \langle m, \star \rangle \text{ or } \mu = \langle \star, m \rangle, \quad \text{for } \mu \in \mathcal{M},\ m \in M.$$

Again, we have a result analogous to Lemma 7.3.2. It says that the derived operators $_ \|_{TCSP} _$ and $_ \|_{TCSP} _$, inherit from $\mathcal{L}$ derived inference rules that coincide with those for $_ \| _$ and $_ \| _$, respectively.

Lemma 7.4.2 *(i) The following are derived rules of inference in $\mathcal{L}$:*

$$\frac{E \xrightarrow{m} E' \qquad F \xrightarrow{m} F'}{E \|_{TCSP} F \xrightarrow{m} E' \|_{TCSP} F'}(\|_{TCSP}) \qquad \frac{E \xrightarrow{i} E' \qquad F \xrightarrow{i} F'}{E \|_{TCSP} F \xrightarrow{i} E' \|_{TCSP} F'}(i\text{-}i)$$

(ii) If $\mathcal{L} \vdash E \|_{TCSP} E' \xrightarrow{\mu} F''$ then $\mu \in M_{SCCS}$ or $\mu = i$ and $F'' = F \|_{TCSP} F'$, for some F, F', and any proof of $E \|_{TCSP} E' \xrightarrow{\mu} F \|_{TCSP} F'$ in $\mathcal{L}$ can be seen as using one of the above derived rules as the last one.

Proof. Omitted. □

Lemma 7.4.3 *(i) The following are derived rules of inference in $\mathcal{L}$:*

$$\frac{E \xrightarrow{\mu} F}{E \,\|_{TCSP}\, E' \xrightarrow{\mu} F \,\|_{TCSP}\, E'}(\|_{TCSP}\text{-l}) \qquad \frac{E' \xrightarrow{\mu} F'}{E \,\|_{TCSP}\, E' \xrightarrow{\mu} E \,\|_{TCSP}\, F'}(\|_{TCSP}\text{-r})$$

$$\frac{E \xrightarrow{i} F \qquad E' \xrightarrow{\mu} F'}{E \,\|_{TCSP}\, F \xrightarrow{\mu} E' \,\|_{TCSP}\, F'}(i\text{-l}) \qquad \frac{E \xrightarrow{\mu} F \qquad E' \xrightarrow{i} F'}{E \,\|_{TCSP}\, F \xrightarrow{\mu} E' \,\|_{TCSP}\, F'}(i\text{-r})$$

for $\mu \in M$ *or* $\mu = i$;

(ii) If $\mathcal{L} \vdash E \,\|_{TCSP}\, E' \xrightarrow{\mu} F''$ *then* $\mu \in M_{SCCS}$ *or* $\mu = i$ *and* $F'' = F \,\|_{TCSP}\, F'$, *for some* F, F', *and any proof of* $E \,\|_{TCSP}\, E' \xrightarrow{\mu} F \,\|_{TCSP}\, F'$ *in* $\mathcal{L}$ *can be seen as using one of the above derived rules as the last one.*

Proof. Omitted. □

One final remark here seems important. Both operators do not make any restrictions as to the structure of the set M of elementary action symbols. It should be therefore evident that we can add them to both: CCS and $SCCS$. In this way we obtain a new sublanguage of $\mathcal{L}$ with inherited from $\mathcal{L}$ inference rules.

7.4.3 Synchronisation Algebras

Winskel's attempt to bring the variety of synchronisation techniques under one roof culminated in introduction of *synchronisation algebras*, see [Winskel 84b], and also [Winskel 82] for the first presentation and the intuitions behind their introduction. A synchronisation algebra is used as a parameter in the definition of a synchronisation operator in a CCS-like language. The choice of different synchronisation algebras yields different CCS-like languages. Thus, we have a synchronisation algebra for CCS itself, another for $SCCS$, and those corresponding to the synchronisation operators in $TCSP$.

In [Winskel 84b] the author shows that a general (interleaving) theory for a CCS-like language with a synchronisation operator parameterised with a synchronisation algebra can be developed. This theory specialises to the theory of a particular language when an appropriate synchronisation algebra is taken. The disadvantage of this proposal is that we cannot apply it to talk about, say, $TCSP$, because this language contains *two* synchronisation operators. The obvious remedy would be to extend the language, and consider more than one operator.

On the other hand we have seen already that both $TCSP$ operators can be modelled as derived operators of $\mathcal{L}$. We have argued that they can be added to CCS to form a sublanguage of $\mathcal{L}$ which has *three* different synchronisation operators. From this perspective our proposal seems to be more flexible. Nonetheless, at least at the moment we fall short of having full $TCSP$ as a sublanguage of $\mathcal{L}$. We shall discuss the problem in the last paragraph. In the remaining part of the present section we show that every operator definable by means of a synchronisation algebra can be modelled in $\mathcal{L}$ as a derived operator, with its inference rules derivable from those of $\mathcal{L}$.

Let us start by quoting the definition of a synchronisation algebra from [Winskel 84b].

Definition 7.4.4 A *synchronisation algebra*, abbr.: *sa*, is an algebra $L = \langle L, \bullet, *, 0\rangle$ where L is a set of *labels* with $*, 0 \in L$, s.t., $L - \{*, 0\} \neq \emptyset$, and $_ \bullet _$ is a binary commutative and associative operation on L which satisfies:

(i) $\forall \lambda \in L . \lambda \bullet 0 = 0$, and

(ii) $* \bullet * = *$ and $\forall \lambda, \lambda' \in L . \lambda \bullet \lambda' = * \Rightarrow \lambda = * = \lambda'$.

The two distinguished elements of L are to play a special role. We use $m{:}\, M = L - \{*, 0\}$ to denote the set of *proper* labels.

Every sa L determines a binary operator in a *CCS*-like language: $_ \textcircled{L} _$. Its operational meaning is given by the following rules:

$$\frac{E \xrightarrow{m} F}{E \textcircled{L} E' \xrightarrow{m \bullet *} F \textcircled{L} E'}(*\text{-l}) \qquad \frac{E' \xrightarrow{m} F'}{E \textcircled{L} E' \xrightarrow{* \bullet m} E \textcircled{L} F'}(*\text{-r}),$$

for $m \in M$ s.t., $* \bullet m \neq 0$, and

$$\frac{E \xrightarrow{m} F \qquad E' \xrightarrow{m'} F'}{E \textcircled{L} E' \xrightarrow{m \bullet m'} F \textcircled{L} F'}(\bullet)$$

for $m, m' \in M$ s.t., $m \bullet m' \neq 0$.

An example should help develop the ideas.

Example 7.4.5 Let us consider the following sa L for *CCS*, see [Winskel 84b]:

$\bullet$	$*$	α	$\overline{\alpha}$	β	$\overline{\beta}$	$\cdots$	τ	0
$*$	$*$	α	$\overline{\alpha}$	β	$\overline{\beta}$	$\cdots$	τ	0
α	α	0	τ	0	0	$\cdots$	0	0
$\overline{\alpha}$	$\overline{\alpha}$	τ	0	0	0	$\cdots$	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
τ	τ	0	0	0	0	$\cdots$	0	0
0	0	0	0	0	0	$\cdots$	0	0

It is easy to verify that the inference rules obtained from L for $_ \textcircled{L} _$ coincide with the usual rules for $_ \mid _$. ◇

The same can be said about the other operators and their synchronisation algebras. Thus, indirectly, existence of a sa for *CCS* with value passing shows that this version of Milner's language can be modelled as a sublanguage of $\mathcal{L}$ as well.

Our method of modelling $_ \textcircled{L} _$ has the same form as previously. Define

$$_ \textcircled{L} _ = ((_ \,\&\, _) \setminus \mathrm{L}_{\textcircled{L}})[\mathrm{R}_{\textcircled{L}}]$$

where

$\mathrm{L}_{\textcircled{L}} \subseteq \mathcal{M}$ is the set given by:

$$\mathrm{L}_{\textcircled{L}} = \mathcal{M} - \{\langle m, m'\rangle \mid m, m' \in M \wedge m \bullet m' \neq 0\}$$
$$- \{\langle m, *\rangle \mid m \in M \wedge m \bullet * \neq 0\} - \{\langle *, m\rangle \mid m \in M \wedge * \bullet m \neq 0\}.$$

$R_{\mathbb{L}}: M \to M$ is given by:

$$R_{\mathbb{L}}(\mu) = \begin{cases} m \bullet * & \mu = \langle m, * \rangle \\ * \bullet m' & \mu = \langle *, m' \rangle \\ m \bullet m' & \mu = \langle m, m' \rangle \end{cases}, \qquad \text{for } \mu \in M,\ m, m' \in M.$$

We note that from the definition of synchronisation algebras it follows that $R_{\mathbb{L}}$ is well defined for $\mu = \langle m, m' \rangle \notin L_{\mathbb{L}}$.

We end with the usual kind of result that says that the derived operator has the same (derived) inference rules as those for $_ \mathbb{L} _$.

Lemma 7.4.6 *(i) The following are derived rules of inference in $\mathcal{L}$:*

$$\frac{E \xrightarrow{m} F}{E \mathbb{L} E' \xrightarrow{m \bullet *} F \mathbb{L} E'}(\mathbb{L}\text{-l}) \qquad \frac{E' \xrightarrow{m} F'}{E \mathbb{L} E' \xrightarrow{* \bullet m} E \mathbb{L} F'}(\mathbb{L}\text{-r}), \quad \textit{for } m \bullet * \neq 0;$$

$$\frac{E \xrightarrow{m} E' \qquad F \xrightarrow{m} F'}{E \mathbb{L} F \xrightarrow{m \bullet m'} E' \mathbb{L} F'}(\mathbb{L}) \qquad \textit{for } m \bullet m' \neq 0;$$

$$\frac{E \xrightarrow{\imath} F \qquad E' \xrightarrow{\mu} F'}{E \mathbb{L} F \xrightarrow{\mu} E' \mathbb{L} F'}(\imath\text{-l}) \qquad \frac{E \xrightarrow{\mu} F \qquad E' \xrightarrow{\imath} F'}{E \mathbb{L} F \xrightarrow{\mu} E' \mathbb{L} F'}(\imath\text{-r}) \quad \textit{where } \mu \in M \textit{ or } \mu = \imath;$$

(ii) If $\mathcal{L} \vdash E \mathbb{L} E' \xrightarrow{\mu} F''$ then $\mu \in M$ or $\mu = \imath$ and $F'' = F \mathbb{L} F'$, for some F, F', and any proof of $E \mathbb{L} E' \xrightarrow{\mu} F \mathbb{L} F'$ in $\mathcal{L}$ can be seen as using one of the above derived rules as the last one.

Proof. Omitted. □

7.4.4 Loose Ends

There remains a question as to what extent the synchronisation operator of $\mathcal{L}$ itself can, or rather cannot, be given a synchronisation algebra.

We have seen that every synchronisation algebra determines a binary operator equipped with inference rules of a special form. Let us consider now a class of binary *synchronisation operators* defined as follows.

Assume M is a set of action symbols. For $_ \otimes _$ to be a synchronisation operator we require that for some functions *left*, *right*: $M \to M$, and $_ \otimes _ : M^2 \to M$ the only inference rules for $_ \otimes _$ are the following:

$$\frac{E \xrightarrow{m} F}{E \otimes E' \xrightarrow{\mathit{left}\, m} F \otimes E'}(\text{l}) \quad \text{and} \quad \frac{E' \xrightarrow{m'} F'}{E \otimes E' \xrightarrow{\mathit{right}\, m'} E \otimes F'}(\text{r}),$$

for $m \in \mathrm{dom}(\mathit{left})$, $m' \in \mathrm{dom}(\mathit{right})$, and

$$\frac{E \xrightarrow{m} F \qquad E' \xrightarrow{m'} F'}{E \otimes E' \xrightarrow{m \otimes m'} F \otimes F'}(\otimes)$$

for $\langle m, m' \rangle \in \mathrm{dom}(\otimes)$.

Every such synchronisation operator $_ \otimes _$ can be modelled in $\mathcal{L}$. Just take $_ \odot _ = ((_ \,\&\, _) \setminus \mathrm{L}_\odot)[\mathrm{R}_\odot]$, where

$\mathrm{L}_\odot \subseteq M$ is the set given by:

$$\mathrm{L}_\odot = M - \{\langle m, * \rangle \mid m \in \mathrm{dom}(\mathit{left})\} - \{\langle *, m \rangle \mid m \in \mathrm{dom}(\mathit{right})\} - \mathrm{dom}(\otimes)$$

$\mathrm{R}_\odot \colon M \to M$ is given by:

$$\mathrm{R}_\odot \mu = \begin{cases} \mathit{left}\, m & \mu = \langle m, * \rangle \\ \mathit{right}\, m & \mu = \langle *, m \rangle \\ m \otimes m' & \mu = \langle m, m' \rangle \end{cases}, \qquad \text{for } \mu \in M,\ m, m' \in M.$$

The details are left to the reader.

Now, every operator $_ \mathbin{\textcircled{L}} _$ corresponding to a sa L is of this form. Just take

$$\mathit{left} = \lambda m{:}M.\ m \bullet *, \quad \text{for } m \bullet * \neq 0,$$

$$\mathit{right} = \lambda m{:}M.\ * \bullet m, \quad \text{for } * \bullet m \neq 0,$$

$$m \otimes m' = m \bullet m', \quad \text{for } m \bullet m' \neq 0.$$

This determines a subclass of synchronisation operators. Namely those with parameters satisfying: $\mathit{left} = \mathit{right}$ and $_ \otimes _$ is commutative an associative. Conversely, every synchronisation operator $_ \otimes _$ from this subclass determines a synchronisation algebra $L_\otimes$ with M as the set of proper labels.

Just take $L_\otimes = \langle M + \{*, 0\}, \bullet, *, 0 \rangle$ where

$$\lambda \bullet \lambda' = \begin{cases} 0 & \lambda = 0 \vee \lambda' = 0 \\ * & \lambda = * = \lambda' \\ \mathit{left}\, \lambda & \lambda \in M \wedge \lambda' = * \\ \mathit{right}\, \lambda' & \lambda = * \wedge \lambda' \in M \\ \lambda \otimes \lambda' & \lambda, \lambda' \in M \end{cases}$$

This correspondence is an 'equivalence'. If we start from a sa L, take corresponding operator $_ \mathbin{\textcircled{L}} _$, and then come back, we obtain an isomorphic sa. If, on the other hand, we start from an operator $_ \otimes _$, take the determined sa, L say, then $_ \mathbin{\textcircled{L}} _$ has the same inference rules as $_ \otimes _$.

Obviously, the operator $_ \,\&\, _$ from $\mathcal{L}$ is a synchronisation operator in the above sense. It is not the case, however, that it belongs to the special subclass of operators definable by means of sa-s. After application of the procedure described above we obtain:

$\bullet$	$*$	$\lightning$	m	$\mathtt{inl}\, m$	$\cdots$	$\langle m, m' \rangle$	$\cdots$	0
$*$	$*$	$\lightning$	$\langle *, m \rangle$	$\langle *, \mathtt{inl}\, m \rangle$	$\cdots$	$\langle *, \langle m, m' \rangle \rangle$	$\cdots$	0
$\lightning$	$\lightning$	$\lightning$	$\langle *, m \rangle$	$\langle *, \mathtt{inl}\, m \rangle$	$\cdots$	$\langle *, \langle m, m' \rangle \rangle$	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
m''	$\langle m'', * \rangle$	$\langle m'', * \rangle$	$\langle m'', m \rangle$	$\langle m'', \mathtt{inl}\, m \rangle$	$\cdots$	$\langle m'', \langle m, m' \rangle \rangle$	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	

Clearly, this table gives a definition of a non-commutative and non-associative operator. Thus, at least these two restrictions should be dropped from the definition of synchronisation algebras to make $\mathcal{L}$ susceptible to the theory developed by Winskel. Luckily, it seems that the restrictions do not play any major part in Winskel's theory. Hence, one can argue that the two approaches have the same expressive power.

We have pointed already that at the moment we are not in the position to model entire $TCSP$ within $\mathcal{L}$. The reason is that there are, among others, two non-detereministic *choice* operators.

So far we have seen how the sum operator from CCS can be modelled in $\mathcal{L}$. Now, with the help of unobservable actions, we are in a position to model another one. We shall call it ***internal choice operator***, since, unlike Milner's sum operator, it is the process itself who decides which of the subprocesses does it really stand for. This operator was studied by Hennessy, in [Hennessy 83,Hennessy 85] and [Hennessy 85/86]. Also, it can be found in [Hoare et. al. 84], denoted there as $_ \sqcap _$. We adopt notation used by Hennessy, and write $_ \oplus _$. Its operational semantics in the form closest to ours can be found in [Hennessy 85/86]. We propose the following inference rules:

$$\frac{}{E \oplus E' \xrightarrow{\imath} E}(\oplus\text{-l}) \qquad \frac{}{E \oplus E' \xrightarrow{\imath} E'}(\oplus\text{-r})$$

To model it in $\mathcal{L}$ we extend the language slightly. We allow prefixing by unobservable action, so new terms like $\imath \cdot E$ appear in $\mathcal{L}$. The reader may note that this operator itself can be modelled in $\mathcal{L}$ in a manner suggested when considering CCS without τ, with a little extra fuss. It has the same operational meaning as any other action operator:

$$\frac{}{\imath \cdot E \xrightarrow{\imath} E}(\text{act-}\imath)$$

is the only new inference rule.

The internal choice is modelled in $\mathcal{L}$ now almost as Milner's sum was.

Define $_ \uplus _ = ((\imath \cdot _) \# (\imath \cdot _))[\mathrm{R}_+]$, where R_+ was used already to define $_ +_{CCS} _$.

The following are the only derived rules, note that as in case of $_ +_{CCS} _$ we and up introducing idle operators as a byproduct:

$$\frac{}{E \uplus E' \xrightarrow{\imath} E\mathtt{idl}}(\uplus\text{-l}) \qquad \frac{}{E \uplus E' \xrightarrow{\imath} E'\mathtt{idr}}(\uplus\text{-r})\,.$$

There are however consequences of the introduction of $\imath$ as the silent action. The rules like ($\imath$-l) in Lemma 7.3.2 now start playing important role. One can no longer forget about them, like we did previously. And they seem to introduce superfluous synchronisations of actions with those supposedly unobservable.

Let us finally draw some limits to the modelling power of $\mathcal{L}$.

First, it seems impossible to model within $\mathcal{L}$ the other choice operator from $TCSP$.

Second, $\mathcal{L}$ is based essentially on CCS, and so does not support ***sequential composition operator*** so popular with $TCSP$ and ASC, see [Bergstra&Klop 84] for more on the latter.

Chapter 8

$\mathcal{L}$ — a Categorical Semantics

In this Chapter the language $\mathcal{L}$ is given a categorical denotational semantics. From the general theory developed in Chapter 6 it follows that a ω-category together with a bunch of continuous functors of appropriate arity is all that is needed.

We choose to work with **LSS** — a category of *labelled skeletal systems*. It is equivalent to **LAS**, considered in Part I category of labelled asynchronous systems. Instead of actions in asynchronous systems we have here the action's occurrences plus explicit equivalence relation between them. The main reason behind that choice is that skeletal systems are more natural when it comes to formally comparing the operational and the denotational semantics. In Chapter 9 we shall see that the occurrences of actions correspond to the derivation trees of the operational semantics.

Some of the categorical properties of **LSS** and its subcategories are investigated. In particular, **LSS** is shown to be a ω-category. A specific ω-subcategory $\mathcal{A}_M$ of **LSS**, together with appropriate ω-functors on it are chosen as the domain of interpretation of $\mathcal{L}$.

All but one of the functors arise naturally from the categorical structure of **LSS**. It was Winskel who first suggested that the *CCS* sum operator can be modelled as a coproduct in appropriate categories, and that many synchronisation operators are related to product, see [Winskel 82,Winskel 84a]. His idea is pursued here with appropriate modifications. Some adjustments are needed since we work with categories of labelled systems. Thus, we model the coproduct operator of $\mathcal{L}$ as a relabelled coproduct, and similarly product operator as a relabelled product in **LSS**. This, incidentally, explains the choice of terminology used in the definition of $\mathcal{L}$.

The constant operator of $\mathcal{L}$ is interpreted as an initial object of $\mathcal{A}_M$, again following Winskel's idea.

Also, the relabelling-hiding, and the restriction operators are interpreted by functors that arise naturally from the structure of **LSS**. It was only after a talk given at Sussex by Edmund Robinson that the author realised that he was trying to interpret $\mathcal{L}$ in one of the *fibre* categories determined be a forgetful functor $\mathbf{U}_M$: **LSS** $\rightarrow$ **Pfn**. This functor sends a system S to the set of its labels M_S, and a morphism $f: S \rightarrow S'$ to its relabelling part f_M.

Categorically, this functor is an *op-fibration*, and a *fibration* — the latter requires certain

modifications in the categories considered. As a consequence of the op-fibration there exist functors between fibre categories corresponding to every partial function between the sets of labels. These functors are used to interpret the renaming-hiding operator, and also in the interpretation of product and coproduct operators. Similarly, fibration determines functors between fibres corresponding to injective partial functions between the sets of labels. These functors are used to interpret the restriction operators of $\mathcal{L}$.

The Chapter consists of two sections.

In the first one an appropriate domain $\mathcal{A}_{\mathcal{M}}$ of interpretation is chosen. We study category **LSS** and its subcategories, with $\mathcal{A}_{\mathcal{M}}$ as one of them. We prove **LSS** to be equivalent to **LAS** — the category of labelled asynchronous systems studied in Chapter 4.

Next, a forgetful functor $\mathrm{U}: \mathbf{LSS} \to \mathbf{ppGrph} \times \mathbf{Pfn}$ is defined. It is shown that U creates in **LSS** (small) limits, coproducts and ω-colimits. Therefore, **LSS** is a (small) complete ω-category in which binary product commute with ω-colimits.

Subsequently, attention is directed at the full coreflective subcategory of **LSS** with c^*-generated systems. It is shown that the coreflector is a continuous functor.

Next, the subcategories of labelled skeletal systems over a fixed alphabet are considered. They are the fibres determined by U_M. We show that this functor is an op-fibration, and a fibration — after appropriate modification of the categories.

Finally, a class of categories $\mathcal{A}_M$ is defined, for $M \in \mathbf{Pfn}$. Each $\mathcal{A}_M$ is defined as the full subcategory of the fibre $\mathbf{LSS}^M$ with c^*-generated forward stable systems.

In the second section the interpretation of $\mathcal{L}$ in $\mathcal{A}_{\mathcal{M}}$ is given. It utilises the little theory developed in the first section.

8.1 An Appropriate Category: $\mathcal{A}_{\mathcal{M}}$

8.1.1 Category of Labelled Skeletal Systems: LSS

Let us define an important category **LSS**, of labelled skeletal systems. It is equivalent to **LAS** — the category of labelled asynchronous systems that was studied earlier. For the sake of future applications we prefer to work with skeletal systems. They are more natural when it comes to recovering *asynchronous-systems-like* objects from the syntax and operational semantics of the language.

There is an important consequence of the equivalence. Suppose we have established equivalence between two ω-categories **C** and **D** via $F: \mathbf{C} \to \mathbf{D}$ and $G: \mathbf{D} \to \mathbf{C}$. Then, for any recursive language $\mathsf{Rec}_\Sigma(\mathsf{Var})$, a semantics of $\mathsf{Rec}_\Sigma(\mathsf{Var})$ in **C** determines automatically an equivalent semantics $\mathbf{C}^*[\![_]\!]: \mathsf{Rec}_\Sigma(\mathsf{Var}) \to [\mathbf{Env}_\mathbf{D} \to_\omega \mathbf{D}]$ in **D**. This is achieved by letting: $\mathbf{C}^*[\![t]\!]\rho = F \circ \mathbf{C}[\![t]\!](G \circ \rho)$.

The 'equivalence' of semantics means that $\mathbf{C}^{**}[\![t]\!] \cong \mathbf{C}[\![t]\!]$, i.e., the functors $\mathbf{C}^{**}[\![t]\!]$ and $\mathbf{C}[\![t]\!]$ are naturally isomorphic, for $t \in \mathsf{Rec}_\Sigma(\mathsf{Var})$.

This in turn has a more practical consequence. Every semantics induces an equivalences on $\mathsf{Rec}_\Sigma(\mathsf{Var})$ given by: $t \approx_{\mathbf{C}[\![_]\!]} t'$ iff $\mathbf{C}[\![t]\!] \cong \mathbf{C}[\![t']\!]$.

In such situation one can prove that $\approx_{C[\![_]\!]} = \approx_{C^*[\![_]\!]}$.

Definition of LSS

Labelled skeletal systems are introduced here formally. We want the reader to try to look at and understand them with labelled asynchronous systems in mind.

Definition 8.1.1 A *labelled skeletal system* $S = \langle C, c^*, A, \asymp, I, \partial^0, \partial^1, M, l \rangle$, abbreviated *lss*, consists of

$c, d: C$ a non-empty set of *configurations* of S,

$c^* \in C$ an *initial* configuration,

$a, b: A$ a set of *edges/arrows*,

$\asymp \subseteq A^2$ an equivalence relation on A called *sameness* of S,

$I \subseteq A^2$ a symmetric and irreflexive relation on A — *independency* of S,

$\partial^0, \partial^1: A \to C$ prescribe *beginning/source:* ∂^0, and *end/target:* ∂^1 to each edge,

M a set of *labels*,

$l: A \rightharpoonup M$ a partial *labelling* function,

such that for $a, a', b, b' \in A$ the following conditions are satisfied:

(i) $\partial^0(a) = \partial^0(b) \wedge a \asymp b$ implies $a = b$,

(ii) $a \asymp a' \; I \; b' \asymp b$ implies $a \; I \; b$,

(iii) $a \; I \; b \wedge \partial^1(a) = \partial^0(b)$ implies $\exists a', b' \in A \,.\, a' \asymp a \wedge b' \asymp b \wedge \partial^0(a) = \partial^0(b') \wedge \partial^1(b') = \partial^0(a') \wedge \partial^1(a') = \partial^1(b)$,

(iv) $a \asymp b$ implies $l(a) = l(b)$.

A lss S is called *forward stable* if, additionally,

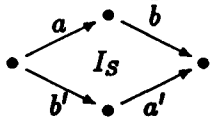
(v) $\partial^0(a) = \partial^0(b) \wedge a \; I \; b$ implies
$\exists a', b' \in A \,.\, a \asymp a' \wedge b \asymp b' \wedge \partial^1(a) = \partial^0(b') \wedge \partial^1(b) = \partial^0(a') \wedge \partial^1(a') = \partial^1(b')$.

As usual we shall refer to the items building a lss S by adding subscript S to the generic term. Thus, e.g., C_S denotes the set of configurations of S, while c^*_S its initial configuration, etc. It is better to develop graphical presentation for objects of our discourse. It should be obvious that each lss S determines $G_S = \langle C_S, c^*_S, A_S, \partial^0_S, \partial^1_S \rangle$ — its *underlying* pointed graph.

If the complexity of the objects under consideration seems too big the reader should wait until the equivalence with labelled asynchronous systems is established which should help see the skeletal systems as an alternative way of presentation of asynchronous systems. It is hoped, though, that a visual representation is helpful in understanding the conditions of the Definition 8.1.1.

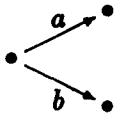
Configurations are represented either by their names or as blobs of ink. The latter stand for anonymous configurations, i.e., ones whose real names are of no interest to us. The initial

configuration is sometimes depicted as a star. An edge a is represented as an arrow from the blob that represents $\partial_S^0(a)$ to the representation of $\partial_S^1(a)$.

Diagram 

is used not only to abbreviate that e.g., the source configuration of b coincides with the target of a, etc., but that $a \asymp_S a'$, $b \asymp_S b'$ and $a\ I_S\ b$ as well. Note that from 8.1.1(ii) it then follows that both a-s are independent of both b-s.

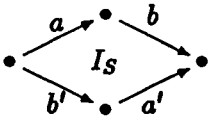
The first condition of Definition 8.1.1 can be rewritten as:



$\wedge\ a \asymp_S b$ implies $a = b$.

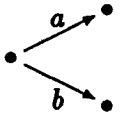
The condition 8.1.1(iii) in turn says:

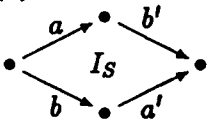


$\wedge\ a\ I_S\ b$ implies 

for some $a', b' \in A_S$.

And similarly, the condition 8.1.1(v) can be written as:



$\wedge\ a\ I_S\ b$ implies 

for some $a', b' \in A_S$.

In accordance with the graphical representation we will denote: $c = \partial_S^0(a), d = \partial_S^1(a)$, by: $c \xrightarrow{a} d$.

We extend the above relation $_ \xrightarrow{} _ \subseteq C_S \times A_S \times C_S$ onto $C_S \times A_S^* \times C_S$, by taking the least relation that satisfies:

(i) $c \xrightarrow{\varepsilon} c$, for $c \in C_S$;

(ii) If $c = \partial_S^0(a), d = \partial_S^1(a)$ and $c' \xrightarrow{x} d$ then $c \xrightarrow{ax} d$, for $c, d \in C_S, a \in A_S, x \in A_S^*$.

Now let us introduce an appropriate notion of morphisms between skeletal systems.

Definition 8.1.2 Let S, T be lss-s. A *morphism f from S to T*, denoted $f: S \to T$, is a triple $f = \langle f_C, f_A, f_M \rangle$, where

$f_C: \langle C_S, c_S^* \rangle \to \langle C_T, c_T^* \rangle$ is a morphism in $\mathbf{Set}_*$, i.e., a (total) function that preserves the distinguished points;

$f_A: A_S \rightharpoonup A_T$ is a partial function;

$f_M: M_S \rightharpoonup M_T$ is a partial function;

such that for $a, b \in A_S$ the following conditions are satisfied:

(i) $f_A(a)\downarrow$ implies $f_C(\partial_S^i(a)) = \partial_T^i(f_A(a))$, for $i = 0, 1$, and
$f_A(a)\uparrow$ implies $f_C(\partial_S^0(a)) = f_C(\partial_S^1(a))$;

(ii) $f_A(a)\downarrow \wedge f_A(b)\downarrow$ implies $a\ I_S\ b \Rightarrow f_A(a)\ I_T\ f_A(b)$;

(iii) $a \asymp_S b$ implies $f_A(a)\downarrow \iff f_A(b)\downarrow$ and $f_A(a)\downarrow \wedge f_A(b)\downarrow \Rightarrow f_A(a) \asymp_T f_A(b)$;

(iv) $f_M \circ l_S = l_T \circ f_A$.

The triples of identity functions form the *identity* morphisms.
The *composition* of morphisms is defined componentwise.

It is evident that the pair $\langle f_C, f_A \rangle$ if a morphism f between labelled skeletal systems determines a morphism of the underlying pointed graphs. The last three conditions of the Definition 8.1.2 say that the morphisms *preserve* the additional structure of lss-s. The following result is immediate.

Fact 8.1.3 *Labelled skeletal systems and their morphisms form a category, denoted* **LSS**.

Proof. It is a simple exercise to check that the axioms of a category are satisfied. □

By **fsLSS** we denote the full subcategory of **LSS** with forward stable systems.

Full Subsystems

Often it is important to consider only part of a lss. We restrict attention to only one such situation. Namely, given sets $C \subseteq C_S$ and $A \subseteq A_S$ of a lss S we ask when they can constitute a set of configurations, and a set of edges of a lss with other items inherited from S.

Formally, let $S_{C,A} = \langle C, c_S^*, A, \asymp, I, \partial^0, \partial^1, M_S, l \rangle$ with

$\asymp = \asymp_S \cap A \times A$;

$I = I_S \cap A \times A$;

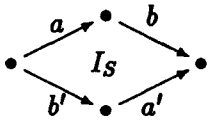
$\partial^0, \partial^1, l$ are restrictions of $\partial_S^0, \partial_S^1$, and l_S, respectively.

The necessary conditions for this construction to work are simple.

Fact 8.1.4 *Let S be a (forward stable) asynchronous system, and $C \subseteq C_S$ and $A \subseteq A_S$. Then the following conditions ensure that $S_{C,A}$ is also a (forward stable) asynchronous system:*

(i) $c_S^* \in C$,

(ii) $\partial_S^0(A), \partial_S^1(A) \subseteq C$,

(iii) If 

in S then

a) $a, b \in A$ *implies* $a', b' \in A$,

b) $a, b' \in A$ *implies* $a', b \in A$,

Moreover, the triple $f = \langle \imath_C, \imath_A, id_{M_S} \rangle$, with $\imath_C : C \hookrightarrow C_S$, $\imath_A : A \hookrightarrow A_S$ inclusions, is a **LSS**-*morphism.*

Proof. Follows easily from the definitions. □

We shall refer to $S_{C,A}$ as the *full sub-lss*, or *full subsystem* or simply *subsystem of S generated by C and A*, whenever the three conditions above are satisfied.

In the future we shall often meet uniform procedures of constructing full subsystem of a class of lss-s. Then we shall be interested in the question if the procedure is functorial. The following simple result is then of value.

Fact 8.1.5 *Let $f = \langle f_C, f_A, f_M\rangle: S \to S' \in \mathbf{Hom}_{\mathbf{LSS}}$, and let $S_{C,A}$, $S'_{C',A'}$ be subsystems of S and S' generated by C and A, and C' and A', respectively. If $f_C(C) \subseteq C'$ and $f_A(A) \subseteq A'$ then the triple $f' = \langle f'_C, f'_A, f_M\rangle$ with f'_C, f'_A restrictions of f_C and f_A, resp., constitutes a morphism $f': S_{C,A} \to S'_{C',A'}$.*

Proof. Follows easily from the definitions. □

The terminology 'full subsystems' is used to stress that the structure of the subsystems is completely determined by the choice of the subsets of configurations, and the subsets of edges.

Among simple consequences of the definition we have the following.

Lemma 8.1.6 *Let S be a lss, while $C' \subseteq C \subseteq C_S$ and $A' \subseteq A \subseteq A_S$ are such that $S_{C,A}$ and $S_{C',A'}$ are full subsystems of S. Then $S_{C',A'}$ is a full subsystem of $S_{C,A}$, and $S_{C',A'} = (S_{C,A})_{C',A'}$.*

Proof. Elementary. □

In other words full subsystems are inherited from full subsystems.

Equivalence between LSS and LAS

For the purpose of future applications we have chosen **LSS** to be the target of our further studies. But in fact, we will be working with labelled asynchronous systems in disguise — the two categories are equivalent.

At this stage the reader should have developed some intuitions about asynchronous systems. Thus, the way in which labelled asynchronous systems give rise to labelled skeletal systems should help to understand the definition of the latter.

Here, we show that the categories **LSS** and **LAS** are equivalent.

To this end a functor $\mathrm{LSS}: \mathbf{LAS} \to \mathbf{LSS}$ is defined as follows.

Let $S = \langle C_S, c^*_S, A_S, I_S, \to, M_S, l_S\rangle$ be a las. Then

$\mathrm{LSS}\, S = \langle C_S, c^*_S, A, \asymp, I, \partial^0, \partial^1, M_S, l\rangle$ where

$A = \to$, $\quad \langle c, a, d\rangle \in A$ will be denoted by $c \xrightarrow{a} d$,

$c \xrightarrow{a} d \asymp c' \xrightarrow{a'} d'$ iff $\quad a = a'$,

$c \xrightarrow{a} d \; I \; c' \xrightarrow{a'} d'$ iff $\quad a \; I_S \; a'$,

$\partial^0(c \xrightarrow{a} d) = c$,

$\partial^1(c \xrightarrow{a} d) = d$,

$l(c \xrightarrow{a} d) = l_S(a)$.

Let $f: S \to S'$ be a morphism in **LSS**. Then

$\text{LSS}\, f = \langle f_C, f'_A, f_M \rangle$ where

$$f'_A(c \xrightarrow{a} d) = f_C(c) \xrightarrow{f_A(a)} f_C(d), \quad \text{for } c, d \in C_S,\ a \in A_S.$$

According to our convention $f'_A(c \xrightarrow{a} d)$ is defined iff $f_A(a)$ is.

Now we are ready to formulate the main result of this paragraph.

Proposition 8.1.7 *(i)* $\text{LSS}: \mathbf{LAS} \to \mathbf{LSS}$ *is a full and faithful functor. It cuts down to a functor between subcategories with forward stable systems.*

(ii) For any $S \in \mathbf{LSS}$ *there exists* $S' \in \mathbf{LAS}$ *s.t.,* $\text{LSS}\, S' \simeq S$.

(iii) Categories **LAS** *and* **LSS** *are equivalent.*

Proof. (i) It is practically evident that $\text{LSS}\, S$ is a lss for a labelled asynchronous system S. It satisfies 8.1.1(i) since the transition system underlying S is deterministic. Condition 8.1.1(ii) is a property of the construction. There is a natural correspondence between the remaining conditions, and the details are omitted.

Similarly with morphisms. Only the proof that LSS is full is presented — all the others are left to the reader.

Let $g: \text{LSS}\, S \to \text{LSS}\, S'$ be a morphism in **LSS**. Morphism $f: S \to S'$ is defined by $f = \langle g_C, f_A, g_M \rangle$ where

$$f_A(a) = a' \quad \text{iff} \quad \exists c, d \in C_S\, \exists c', d' \in C_{S'}\, .\, c \xrightarrow{a} d \ \wedge\ g_A(c \xrightarrow{a} d) = c' \xrightarrow{a'} d'.$$

Then f_A is well-defined since g satisfies 8.1.2(iii). Each of the remaining three conditions of the Definition 8.1.2 ensures that its counterpart in the Definition 4.1.6 holds. Thus, f is a morphism in **LAS**.

(ii) What we are about to construct is in fact the object part of a functor inverse to LSS. If S is a lss then we define S' as follows.

$S' = \langle C_S, c^*_S, A, I, \to, M_S, l \rangle$ where

$A = A_S/\asymp_S$ i.e., we take the set of equivalence classes of edges w.r.t., $\asymp_S$,

$[a]_{\asymp_S}\ I\ [b]_{\asymp_S}$ iff $a'\ I_S\ b'$, for some $a \asymp_S a'$, $b \asymp_S b'$,

$c \xrightarrow{[a]_{\asymp_S}} d$ iff $c = \partial^0(a') \ \wedge\ d = \partial^1(a')$, for some $a' \asymp_S a$,

$l([a]_{\asymp_S}) = l_S(a)$.

Since S satisfies 8.1.1(ii) it follows that alternatively we could have defined independence between the equivalence classes $[a]_{\asymp_S}$ and $[b]_{\asymp_S}$ by demanding that any pair of representatives is related by I_S in S. This property makes the proof that S' is a labelled asynchronous system easy.

Finally, let $f_A: A_S \to A$ be defined by: $f_A(a) = \partial^0(a) \xrightarrow{[a]_{\asymp_S}} \partial^1(a)$, for $a \in A_S$. The reader may wish to verify that the triple $f = \langle id_{C_S}, f_A, id_{M_S} \rangle$ is an isomorphism of labelled skeletal systems.

(iii) Follows immediately from (i) and (ii) by Theorem 1 p.91 in [Mac Lane 71]. □

8.1.2 Properties of LSS

We study here the properties of **LSS**. This is done by relating **LSS** to categories studied previously by means of a forgetful functor $\mathrm{U}: \mathbf{LSS} \to \mathbf{ppGrph} \times \mathbf{Pfn}$. We show that many categorical properties of **LSS** are 'imported', i.e., created, via this functor. In particular we show that it creates ω-colimits, zero-objects, products and coproducts. We can infer therefore that ω-colimits commute with binary products in **LSS**, since this property is 'imported' as well.

Let us start from the definition of the forgetful functor.

Definition 8.1.8 Let $\mathrm{U}: \mathbf{LSS} \to \mathbf{ppGrph} \times \mathbf{Pfn}$ be defined on objects:

$\mathrm{U}\,S = \langle G_S, M_S \rangle$ where $G_S = \langle C_S, c^*_S, A_S, \partial^0_S, \partial^1_S \rangle$ is the underlying pointed graph.

and on morphisms:

$\mathrm{U}\,f = f$

As we see U indeed forgets a lot of structural information — it forgets sameness and independency in objects, and also that the morphisms preserve them, and the labelling. We state without proof the following obvious fact.

Fact 8.1.9 U *defined above is a functor.*

Two derived functors obtained by composing U with appropriate projection functor will also be of interest. We define $\mathrm{U}_M: \mathbf{LSS} \to \mathbf{Pfn}$ on objects:

$$\mathrm{U}_M(S) = M_S,$$

and on morphisms:

$$\mathrm{U}_M(f) = f_M.$$

Similarly, functor U_G is defined as the composition of U with the projection on **ppGrph**: $\mathrm{U}_G(S) = G_S$, and $\mathrm{U}_G(f) = \langle f_C, f_A \rangle$.

The remaining part of this section is devoted to the study of the properties of U. We assume familiarity with categorical constructions on (pointed) graphs as described in Section 2.3.2, and often do not bother to remind the reader where are they coming from. Even more so in the case of constructions in **Set**, **Pfn**, etc.

U Creates Products in LSS

Let S, T be lss-s. We consider $S \amalg T = \langle C_\amalg, c^*_\amalg, A_\amalg, \asymp_\amalg, I_\amalg, \partial^0_\amalg, \partial^1_\amalg, M_\amalg, l_\amalg \rangle$, where

$G_\amalg = \langle C_\amalg, c^*_\amalg, A_\amalg, \partial^0_\amalg, \partial^1_\amalg \rangle$ is the binary product of $\mathrm{U}_G(S)$ and $\mathrm{U}_G(T)$ in **ppGrph** with projections: $\pi^S_G: G_\amalg \to \mathrm{U}_G(S), \pi^T_G: G_\amalg \to \mathrm{U}_G(T)$, as constructed in Section 2.3.2;

$M_\amalg = M_S + M_S \times M_T + M_T$ is a product of $\mathrm{U}_M(S)$ with $\mathrm{U}_M(T)$ in **Pfn** with corresponding projections: $\pi^S_M: M_\amalg \to M_S$, $\pi^T_M: M_\amalg \to M_T$,

and where the remaining items are defined as follows.

$\asymp_\Pi \subseteq A_\Pi \times A_\Pi$, is equal $\asymp^\Pi = \asymp^\Pi_S \cap \asymp^\Pi_T$ where $\asymp^\Pi_S$ is given by:

$a_\Pi \asymp^\Pi_S b_\Pi$ iff $\pi^S_A(a_\Pi)\downarrow \iff \pi^S_A(b_\Pi)\downarrow$ and $\pi^S_A(a_\Pi)\downarrow \ \wedge\ \pi^S_A(b_\Pi)\downarrow$ implies $\pi^S_A(a_\Pi) \asymp_S \pi^S_A(b_\Pi)$,

and where $\asymp^\Pi_T$ is defined analogically.

$I_\Pi \subseteq A_\Pi \times A_\Pi$, is equal $I^\Pi = I^\Pi_S \cap I^\Pi_T$, where

$a_\Pi\ I^\Pi_S\ b_\Pi$ iff $\pi^S_A(a_\Pi)\downarrow \wedge \pi^S_A(b_\Pi)\downarrow$ implies $\pi^S_A(a_\Pi)\ I_S\ \pi^S_A(b_\Pi)$, and

$a_\Pi\ I^\Pi_T\ b_\Pi$ iff $\pi^T_A(a_\Pi)\downarrow \wedge \pi^T_A(b_\Pi)\downarrow$ implies $\pi^T_A(a_\Pi)\ I_T\ \pi^T_A(b_\Pi)$.

$l_\Pi : A_\Pi \rightharpoonup M_\Pi$ is defined as the unique partial function that satisfies:

$\pi^S_M \circ l_\Pi = l_S \circ \pi^S_A$ and $\pi^T_M \circ l_\Pi = l_T \circ \pi^T_A$.

In other words, l_Π is defined from the categorical properties of the product.
Finally, let $\pi_S = \langle \pi^S_C, \pi^S_A, \pi^S_M \rangle$ and $\pi_T = \langle \pi^T_C, \pi^T_A, \pi^T_M \rangle$.

The remaining part of this paragraph is devoted to the proof of the proposition that assures that the above construction gives us a binary product in **LSS**.

Proposition 8.1.10 *Let S, T be lss-s, and $S \Pi T$ as defined above. Then*

(i) $S \Pi T$ is a lss,

(ii) $S \Pi T$ is forward stable whenever both S and T are forward stable,

(iii) $S \Pi T$ is categorical coproduct of S and T in **LSS** *with projection morphism: $\pi_S : S \Pi T \to S$ and $\pi_T : S \Pi T \to T$,*

(iv) **LSS** *has binary products created by* U.

Proof. (i) We note first that all the assumptions on the components of a lss from the Definition 8.1.1 are satisfied by $S \Pi T$. In particular I_Π is irreflexive. Next, we should check that the conditions of the Definition 8.1.1 are satisfied as well. This is a particularly tiresome business as there are many cases to inspect for each condition. For this reason we feel happy to sketch the proofs rather than go deeply into their details.

To see that 8.1.1(i) is satisfied we need to consider three cases, two of them symmetric.

Let $a_\Pi, b_\Pi \in A_\Pi$, $\partial^0_\Pi(a_\Pi) = \partial^0_\Pi(b_\Pi)$ and $a_\Pi \asymp_\Pi b_\Pi$. If $\pi^S_A(a_\Pi)\uparrow$ then also $\pi^S_A(b_\Pi)\uparrow$ as well. This in turn implies $\pi^T_A(a_\Pi)\downarrow$, and $\pi^T_A(a_\Pi) \asymp_T \pi^T_A(b_\Pi)$ and so for $c = \pi^S_C(\partial^0_\Pi(a_\Pi)) = \pi^S_C(\partial^0_\Pi(b_\Pi))$ we have $a_\Pi = \langle c, \pi^T_A(a_\Pi)\rangle, b_\Pi = \langle c, \pi^T_A(b_\Pi)\rangle$. But $\partial^0_\Pi(a_\Pi) = \partial^0_\Pi(b_\Pi)$ implies

$$\partial^0_T(\pi^T_A(a_\Pi)) = \pi^T_C(\partial^0_\Pi(a_\Pi)) = \pi^T_C(\partial^0_\Pi(b_\Pi)) = \partial^0_T(\pi^T_A(b_\Pi)),$$

so that $\pi^T_A(a_\Pi) = \pi^T_A(b_\Pi)$ is a consequence of T satisfying 8.1.1(i). Thus we see that $a_\Pi = b_\Pi$.

The other nonsymmetric case where $\pi^S_A(a_\Pi)\downarrow$, and $\pi^S_A(b_\Pi)\downarrow$ is simpler to deal with.

The proofs that $S \sqcap T$ satisfies 8.1.1(ii) and 8.1.1(iv) are omitted. As for 8.1.1(iii) we should consider seven cases. In fact many of them are symmetric so that we need to take care of four. Even that number seems too much, so that we deal with one example case only. The proofs of the others are quite similar and left to the reader. Thus, assume $a_\Pi \; I_\Pi \; b_\Pi$ and $\partial^1_\Pi(a_\Pi) = \partial^0_\Pi(b_\Pi)$, for some $a_\Pi, b_\Pi \in A_\Pi$. And assume $a_\Pi = \langle a_S, a_T \rangle, b_\Pi = \langle b_S, d_T \rangle$, where $a_S, b_S \in A_S$, $a_T \in A_T$, $d_T \in C_T$ are such that $a_S \; I_S \; b_S$.

From $\partial^1_\Pi(a_\Pi) = \partial^0_\Pi(b_\Pi)$, by definition of ∂^i_Π, it follows that $\partial^1_S(a_S) = \partial^0_S(b_S)$ and $\partial^1_T(a_T) = d_T$. Therefore there are some $a'_S, b'_S \in A_S$ such that $a_S \prec_S a'_S$, $b_S \prec_S b'_S$, $\partial^0_S(a_S) = \partial^0_S(b'_S)$, $\partial^1_S(b'_S) = \partial^0_S(a'_S)$ and $\partial^1_S(a'_S) = \partial^1_S(b_S)$. It is easy to show that then the two pairs: $a'_\Pi = \langle a'_S, d_T \rangle$ and $b'_\Pi = \langle b'_S, \partial^0_T(a_T) \rangle$ are what we are looking for.

(ii) The proof of this property is very similar to the proof that $S \sqcap T$ satisfies 8.1.1(iii), as sketched above.

(iii) First we verify that π_S and π_T are morphisms in **LSS**. They do satisfy 8.1.2(i) since the pairs $\langle \pi^S_C, \pi^S_A \rangle$, $\langle \pi^T_C, \pi^T_A \rangle$ constitute projection morphisms in **ppGrph**.

As for the conditions 8.1.2(ii) and 8.1.2(iii) they are satisfied automatically, by definition.

Finally, condition 8.1.2(iv) holds by the definition of l_Π, for both: π_S and π_T.

To finish with this part of the proof one has to check that for any lss S_0 and any **LSS**-morphisms $f: S_0 \to S$, $g: S_0 \to T$, the triple $h = \langle h_C, h_A, h_M \rangle$ built from the mediating morphisms between derived diagrams in **ppGrph** and **Pfn**, is a morphism in **LSS**. This is pretty obvious. The condition 8.1.2(i) is inherited from **ppGrph**. The proofs of the next two are based on the observation that the relations: I_Π and $\prec_\Pi$ are defined in the *minimal* way that ensures that π_S, π_T satisfy conditions 8.1.2(ii) and 8.1.2(iii).

The last one is proved as follows.

For $m_\Pi, n_\Pi \in M_\Pi$ one has $m_\Pi = n_\Pi$ iff $\pi^S_M(m_\Pi) = \pi^S_M(n_\Pi) \; \wedge \; \pi^T_M(m_\Pi) = \pi^T_M(n_\Pi)$. Therefore, for $a_0 \in A_0$ it is enough to show that $\pi^S_M(l_\Pi(h_A(a_0))) = \pi^S_M(h_M(l_0(a_0)))$ and, symmetrically, $\pi^T_M(l_\Pi(h_A(a_0))) = \pi^T_M(h_M(l_0(a_0)))$.

We have

$$\begin{aligned}
\pi^S_M(l_\Pi(h_A(a_0))) &= \pi^S_M \circ l_\Pi \circ h_A(a_0) \\
&= l_S \circ \pi^S_A \circ h_A(a_0) && \text{by def., of } l_\Pi \\
&= l_S \circ f_A(a_0) && \text{by def., of } h_A \\
&= f_M \circ l_0(a_0) && \text{since } f \text{ a morphism} \\
&= \pi^S_M \circ h_M \circ l_0(a_0) && \text{by def., of } h_M \\
&= \pi^S_M(h_M(l_0(a_0))),
\end{aligned}$$

as required. Similarly in the other case.

(iv) Evident from the construction. □

More generally we see that U creates arbitrary products.

Proposition 8.1.11 **LSS** *has (small) products created by* U. *In particular the lss* $\mathbf{1}_{\mathbf{LSS}} = \langle\{\star\}, \star, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset\rangle$ *is terminal in* **LSS**.

Proof. It is straightforward to generalise the construction from the binary to arbitrary (small) products, and we leave this task to the reader. It is easier to check simply that $\mathbf{1}_{\mathbf{LSS}}$ is terminal in **LSS**. □

U Creates Coproducts in LSS

Assume S, T are lss-s. The coproduct $M_{\amalg}$ of $\mathrm{U}_M(S)$ and $\mathrm{U}_M(T)$ is, simply, their disjoint union, with injections $in^S_M: M_S \rightarrow M_{\amalg} \leftarrow M_T : in^T_M$. Let us recall how the coproduct $G_{\amalg}$ of $\mathrm{U}_G(S)$ with $\mathrm{U}_G(T)$ looks like. We have seen that the coproducts in **ppGrph** are created via the forgetful functor $\mathrm{U}: \mathbf{ppGrph} \rightarrow \mathbf{Set}_\star \times \mathbf{Pfn}$. Thus, $C_{\amalg}$ is the disjoint union of C_S and C_T with $c^\star_S$ glued together with $c^\star_T$, while $A_{\amalg}$ is simply a disjoint union of A_S and A_T. Recall moreover that $\partial^i_{\amalg}$, for $i = 0, 1$, are defined by: $\partial^i_{\amalg} = \partial^i_S \amalg \partial^i_T$, i.e., on the basis of binary coproduct being a bifunctor in **Pfn**. It is crucial that coproduct of two total functions in **Pfn** is itself a total function. In more elementary terms we have

$$\partial^i_{\amalg}(a_{\amalg}) = \begin{cases} in^S_C(\partial^i_S(a)) & a_{\amalg} = in^S_A(a) \\ in^T_C(\partial^i_T(a)) & a_{\amalg} = in^T_A(a) \end{cases}.$$

Recall also that the injections in **ppGrph** are pairs of injective (total) functions, e.g., $in^S_G = \langle in^S_C, in^S_A\rangle$.

What we have to do is to augment the data already given by defining sameness, independency and labelling, so as to end up with a coproduct in **LSS**.

We proceed as follows.

$\asymp_{\amalg} \subseteq A_{\amalg} \times A_{\amalg}$ is given by:

$$a_{\amalg} \asymp_{\amalg} b_{\amalg} \quad \text{iff} \quad \exists a, b \in A_S \,.\, a \asymp_S b \wedge a_{\amalg} = in^S_A(a) \wedge b_{\amalg} = in^S_A(b),$$
$$\text{or} \quad \exists a, b \in A_T \,.\, a \asymp_T b \wedge a_{\amalg} = in^T_A(a) \wedge b_{\amalg} = in^T_A(b).$$

$I_{\amalg} \subseteq A_{\amalg} \times A_{\amalg}$ is given by:

$$a_{\amalg}\, I_{\amalg}\, b_{\amalg} \quad \text{iff} \quad \exists a, b \in A_S \,.\, a\, I_S\, b \wedge a_{\amalg} = in^S_A(a) \wedge b_{\amalg} = in^S_A(b),$$
$$\text{or} \quad \exists a, b \in A_T \,.\, a\, I_T\, b \wedge a_{\amalg} = in^T_A(a) \wedge b_{\amalg} = in^T_A(b).$$

$l_{\amalg}: A_{\amalg} \rightharpoonup M_{\amalg}$ is the unique function that satisfies:

$$l_{\amalg} \circ in^S_A = in^S_M \circ l_S \quad \text{and} \quad l_{\amalg} \circ in^T_A = in^T_M \circ l_T.$$

Finally, we put $S \amalg T = \langle C_{\amalg}, c^\star_{\amalg}, A_{\amalg}, \asymp_{\amalg}, I_{\amalg}, \partial^0_{\amalg}, \partial^1_{\amalg}, M_{\amalg}, l_{\amalg}\rangle$.
Moreover, let $in_S = \langle in^S_C, in^S_A, in^S_M\rangle$ and $in_T = \langle in^T_C, in^T_A, in^T_M\rangle$.
We are ready to show that we have constructed categorical coproduct of S and T.

Proposition 8.1.12 *Let S and T be lss-s, and $S \amalg T$ as constructed above. Then*

(i) $S \amalg T$ *is a lss,*

(ii) $S \amalg T$ *is forward stable provided both* S *and* T *are forward stable as well,*

(iii) $S \amalg T$ *is a categorical coproduct of* S *and* T *with* $in_S: S \to S \amalg T$ *and* $in_T: T \to S \amalg T$ *as categorical injections,*

(iv) **LSS** *has binary coproducts created via* U.

Proof. (i) Assumptions and conditions of Definition 8.1.1 have to be verified. As for the assumptions one readily sees that $\asymp_{\amalg}$ is an equivalence relation, while $I_{\amalg}$ is an independency relation, i.e., it is irreflexive and symmetric.

Next, we verify that the condition 8.1.1(iii), say, is satisfied by $S \amalg T$. To this end assume $a_{\amalg} \; I_{\amalg} \; b_{\amalg} \; \wedge \; \partial^1_{\amalg}(a_{\amalg}) = \partial^0_{\amalg}(b_{\amalg})$. By the definition of $I_{\amalg}$ and without loss of generality we can assume that for some $a, b \in A_S$ such that $a \; I_S \; b$ we have: $a_{\amalg} = in^S_A(a) \wedge b_{\amalg} = in^S_A(b)$.

By definition of $\partial^i_{\amalg}$, for $i = 0, 1$, it follows therefore that:

$$in^S_A(\partial^1_S(a)) = \partial^1_{\amalg}(a_{\amalg}) = \partial^0_{\amalg}(b_{\amalg}) = in^S_A(\partial^0_S(b)).$$

Thus, $\partial^1_S(a) = \partial^0_S(b)$, since in^S_A is 1–1. We can now find $a', b' \in A_S$ such that $\partial^0_S(a) = \partial^0_S(b')$, $\partial^1_S(b') = \partial^0_S(a')$ and $\partial^1_S(a') = \partial^1_S(b)$, since S satisfies 8.1.1(iii).

Finally, we can take $a'_{\amalg} = in^S_A(a'), b'_{\amalg} = in^S_A(b')$ as they enjoy the required property.

The verification of the remaining conditions is left to the reader.

(ii) Similar to the above.

(iii) First of all we have to check that in_S, in_T are **LSS**-morphisms. This is easy and left to the reader. She/he can skip the proof of 8.1.2(i), as this is automatically satisfied since in^S_G, in^T_G are morphisms in **ppGrph**.

Moreover, since the components of in_S, in_T constitute injections in the appropriate categories, all we need to show is that for any $f: S \to S_0$, $g: T \to S_0$ the triple $h = \langle h_C, h_A, h_M \rangle$, where

$$h_C(in^S_C(c)) = f_C(c) \quad \text{and} \quad h_C(in^T_C(c)) = g_C(c);$$
$$h_A(in^S_A(a)) = f_A(a) \quad \text{and} \quad h_A(in^T_A(a)) = g_A(a);$$
$$h_M(in^S_M(m)) = f_M(m) \quad \text{and} \quad h_M(in^T_M(m)) = g_M(m).$$

is a morphism. As usual condition (i) of the Definition 8.1.2 is for free, since $\langle h_C, h_A \rangle$is the mediating morphism in **ppGrph**. Let us show that one of the other two conditions holds as well, the remaining two being left to the reader as an exercise.

Assume we wish to check that h is indeed labelling preserving, i.e., that it satisfies 8.1.2(iv). If $a_{\amalg} = in^T_A(a)$ then

$$
\begin{array}{rcll}
h_M(l_{\mathrm{II}}(a_{\mathrm{II}})) & = & h_M(in_M^T(l_T(a))) & \text{by def. of } l_{\mathrm{II}} \\
& = & g_M(l_T(a)) & \text{by def. of } h_M \\
& = & l_0(g_A(a)) & g \text{ is a morphism ,} \\
& = & l_0(h_A(in_A^T(a))) & \text{by def of } h_A \\
& = & l_0(h_A(a_{\mathrm{II}})) &
\end{array}
$$

as required. The proof of the remaining case is symmetric.

(iv) Immediate from the construction. □

The following result is a natural extension of the previous one.

Proposition 8.1.13 LSS *has (small) coproducts created via* U. *In particular the lss* $\mathbf{1}_{\mathbf{LSS}} = \langle \{\star\}, \star, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$ *is initial in* **LSS**.

Proof. It is straightforward to generalise the construction from the binary to arbitrary (small) coproducts, and we leave this task to the reader. It should be even easier to check simply that $\mathbf{1}_{\mathbf{LSS}}$ is initial in **LSS**. □

U Creates ω-colimits in LSS

Let $\Gamma = S_0 \stackrel{f^0}{\to} S_1 \stackrel{f^1}{\to} \ldots$ be a ω-diagram in **LSS**, where $f^n = \langle f_C^n, f_A^n, f_M^n \rangle$, and $S_n = \langle C_n, c_n^\star, A_n, \asymp_n, I_n, \partial_n^0, \partial_n^1, M_n, l_n \rangle$, for $n \in \omega$. Above we depart from the convention that g^n stands for the n-fold composition of g with itself.

Let us take the derived ω-diagrams: $\mathrm{U}_G(\Gamma)$ and $\mathrm{U}_M(\Gamma)$, in **ppGrph** and **Pfn**, respectively. From Chapter 2 it follows that they do have colimiting cocones: $\phi_G \colon \mathrm{U}_G(\Gamma) \to G_\to$ and $\phi_M \colon \mathrm{U}_M(\Gamma) \to M_\to$.

With assumption as above by $f_A^{k,l} \colon A_l \rightharpoonup A_k$ we denoted composition: $f_A^{k-1} \circ \ldots \circ f_A^l$, for $k \geq l$ where $f^{l,l} = id_{A_l}$. With this convention the ω-colimits are defined as follows.

$G_\to = \langle C_\to, c_\to^\star, A_\to, \partial_\to^0, \partial_\to^1 \rangle$ where

$C_\to = (\coprod_{n\in\omega} C_n)/{\approx_C}$ where $\approx_C$ is an equivalence relation given by:

$$c \approx_C d \quad \text{iff} \quad \exists k \geq m,n \,.\, f_C^{k,m}(c) = f_A^{k,n}(d), \quad \text{for } c \in C_m, d \in C_n,$$

$c_\to^\star = [c_0^\star]_{\approx_C}$,

$A_\to = (\coprod_{n\in\omega} A_n)/{\approx_A}$ where $\approx_A$ is a partial equivalence relation given by:

$$a \in \mathrm{dom}(\approx_A) \quad \text{iff} \quad \forall m \in \omega \,.\, f_A^{m+n,n}(a)\!\downarrow, \quad \text{for } a \in A_n, \text{ and}$$
$$a \approx_A b \quad \text{iff} \quad a,b \in \mathrm{dom}(\approx_A) \wedge \exists k \geq m,n \,.\, f_A^{k,m}(a) = f_A^{k,n}(b),$$
$$\text{for } a \in A_m, b \in A_n,$$

$\partial_\to^i([a]_{\approx_A}) = [\partial_n^i(a)]_{\approx_C}$ for $a \in A_n \cap \mathrm{dom}(\approx_A), i = 0,1$.

Moreover, $\phi_n \colon G_n \to G_\to$ is defined by:

$\phi_C^n(c) = [c]_{\approx_C}$, for $c \in C_n$;

$\phi_A^n(a) = [a]_{\approx_A}$, for $a \in A_n$, i.e., $\phi_A^n(a)\!\downarrow$ iff $a \notin \mathrm{dom}(\approx_A)$, by our convention.

Similarly,

$M_{\rightarrow} = (\coprod_{n\in\omega} M_n)/{\approx_M}$ where $\approx_M$ is defined analogously to $\approx_A$,

and with cocone given by

$\phi^n_M(m) = [m]_{\approx_M}$, for $m \in M_n$.

It remains to augment this data with appropriate definitions of $\asymp_{\rightarrow}, I_{\rightarrow}$ and $l_{\rightarrow}$ so that the resulting $S_{\rightarrow}$ becomes a lss, while for $n \in \omega$ triples $\phi_n = \langle \phi^n_C, \phi^n_A, \phi^n_M \rangle$ become LSS-morphisms, and the required universal properties are satisfied. We propose the following candidates.

Below, $a \in A_m, b \in A_n$. Then

$\asymp_{\rightarrow} \subseteq A_{\rightarrow} \times A_{\rightarrow}$ is given by:

$[a]_{\approx_A} \asymp_{\rightarrow} [b]_{\approx_A}$ iff $\exists k \geq m,n \exists a', b' \in A_k \,.\, a \approx_A a' \wedge b \approx_A b' \wedge a' \asymp_k b'$,

$I_{\rightarrow} \subseteq A_{\rightarrow} \times A_{\rightarrow}$ is given by:

$[a]_{\approx_A} I_{\rightarrow} [b]_{\approx_A}$ iff $\exists k \geq m,n \exists a', b' \in A_k \,.\, a \approx_A a' \wedge b \approx_A b' \wedge a' I_k b'$,

$l_{\rightarrow}: A_{\rightarrow} \rightarrow M_{\rightarrow}$ is defined by:

$l_{\rightarrow}([a]_{\approx_A}) = [l_m(a)]_{\approx_M}$.

The following proposition collects the properties of the construction carried out above.

Proposition 8.1.14 *With the above notation the following hold:*

(i) $S_{\rightarrow} = \langle C_{\rightarrow}, c^*_{\rightarrow}, A_{\rightarrow}, \asymp_{\rightarrow}, I_{\rightarrow}, \partial^0_{\rightarrow}, \partial^1_{\rightarrow}, M_{\rightarrow}, l_{\rightarrow} \rangle$ *is a lss,*

(ii) $S_{\rightarrow}$ *is forward stable provided* S_n *are forward stable, for* $n \in \omega$,

(iii) $\phi: \Gamma \rightarrow S_{\rightarrow}$ *is a cocone in* **LSS**, *where* $\phi = \{\phi_n \mid n \in \omega\}$ *with* $\phi_n = \langle \phi^n_C, \phi^n_A, \phi^n_M \rangle$

(iv) ϕ *is colimiting, and for any other cocone* $\psi: \Gamma \rightarrow S$ *the mediating morphism* $f: \phi \rightarrow \psi$ *is given by:*

$$f_C([c]_{\approx_C}) = \psi^n_C(c), \quad \text{for } c \in C_n,$$
$$f_A([a]_{\approx_A}) = \psi^n_A(a), \quad \text{for } a \in A_n,$$
$$f_M([m]_{\approx_M}) = \psi^n_M(m), \quad \text{for } m \in M_n,$$

(v) **LSS** *has* ω*-colimits which are created via* U.

Proof. (i) We have to show that $S_{\rightarrow}$ satisfies conditions of the Definition 8.1.1. Here we prove only that it satisfies 8.1.1(iii). Proofs of the remaining conditions are similar, and left to the reader.

Let $\partial^0_{\rightarrow}(a_{\rightarrow}) = \partial^1_{\rightarrow}(b_{\rightarrow})$ and $a_{\rightarrow} \asymp_{\rightarrow} b_{\rightarrow}$, for some $a_{\rightarrow}, b_{\rightarrow} \in A_{\rightarrow}$. Then for some $m \in \omega, a, b \in A_m \cap \mathrm{dom}(\approx_A)$ we have: $a \in a_{\rightarrow}, b \in b_{\rightarrow}, \partial^0_m(a) \approx_C \partial^1_m(b)$ and $a \asymp_m b$. From the definition of $\approx_C$ it follows that for some $n \geq m$ we have: $f^{n,m}_C(\partial^0_m(a)) = f^{n,m}_C(\partial^0_m(b))$. Now, $a' = f^{n,m}_A(a), b' = f^{n,m}_A(b)$ are defined, since $a, b \in \mathrm{dom}(\approx_A)$. Moreover, $a' \asymp_n b'$ follows by induction, since $f^m, \ldots, f^n$ are morphisms.

We conclude that $a' = b'$, since S_n satisfies condition 8.1.1(i). Thus $a \approx_A b$ follows by transitivity of $\approx_A$, which ends the proof.

(ii) Omitted.

(iii) We have to show that each $\phi_n = \langle \phi_C^n, \phi_A^n, \phi_M^n \rangle$ is a **LSS**-morphism $\phi: S_n \to S_{\rightarrow}$. Moreover, they must satisfy: $\phi_{n+1} \circ f^n = \phi_n$, for $n \in \omega$.

The second property follows easily, since it holds in the underlying categories, and the composition in **LSS** is inherited from **ppGrph** and **Pfn**.

Back to the first property, we note that the first condition, see Definition 8.1.2, is automatically satisfied by ϕ_n since $\phi_G^n = \langle \phi_C^n, \phi_A^n \rangle$ is a morphism in **ppGrph**. We prove that ϕ_n satisfies the next condition, and leave the proofs of the remaining two to the reader.

Assume that $a, b \in A_n$ and $\phi_A^n(a)\downarrow, \phi_A^n(b)\downarrow$, so that $a, b \in \mathrm{dom}(\approx_A)$ follows. Let moreover $a\ I_n\ b$. Then $\phi_A^n(a) = [a]_{\approx_A}\ I_{\rightarrow}\ [b]_{\approx_A} = \phi_A^n(b)$, by the definition of $I_{\rightarrow}$.

(iv) We know that the derived cocones: $\phi_G: \mathrm{U}_G(\Gamma) \to \mathrm{U}_G(S_{\rightarrow})$ in **ppGrph**, and $\phi_M: \mathrm{U}_M(\Gamma) \to \mathrm{U}_M(S_{\rightarrow})$ in **Pfn**, are colimiting there. Moreover, for the derived cocones: $\psi_G = \mathrm{U}_G(\psi)$ and $\psi_M = \mathrm{U}_M(\psi)$, the maps: $f_G = \langle f_C, f_A \rangle: \mathrm{U}_G(S_{\rightarrow}) \to \mathrm{U}_G(S)$ and $f_M: \mathrm{U}_M(S_{\rightarrow}) \to \mathrm{U}_M(S)$, are mediating. Since the composition of morphisms in **LSS** is inherited from **ppGrph**$\times$**Pfn** it follows immediately that there is at most one morphism $g: S_{\rightarrow} \to S$ such that $g \circ \phi_n = \psi_n$, for $n \in \omega$. Moreover, if it exists it has to be equal f. All that remains to be done is to show that $f: S_{\rightarrow} \to S$ is a morphism in **LSS**.

As before we notice that f automatically satisfies condition 8.1.1(i), since $f_G = \langle f_C, f_A \rangle$ is a morphism in **ppGrph**. This time, for a change, we show that f satisfies 8.1.1(iii), and skip the proofs of the two others conditions.

Let $a_{\rightarrow}, b_{\rightarrow} \in A_{\rightarrow}$ and $a_{\rightarrow} \asymp_{\rightarrow} b_{\rightarrow}$. Then for some $n \in \omega, a, b \in A_n, a \in a_{\rightarrow}, b \in b_{\rightarrow}$ we have $a \asymp_n b$. If $f_A(a_{\rightarrow})\downarrow$, i.e., $\psi_A^n(a)\downarrow$ then $\psi_A^n(b)\downarrow$, since $\psi_n: S_n \to S$ is a **LSS**-morphism. So, $f_A(b_{\rightarrow})\downarrow$.

And if $f_A(a_{\rightarrow})\downarrow$ then

$$f_A(a_{\rightarrow}) = \psi_A^n(a) \asymp_S \psi_A^n(b) = f_A(b_{\rightarrow}),$$

as required, since ψ_n is a morphism.

(v) It is evident by construction that U creates ω-colimits. Since both **ppGrph** and **Pfn**, are cocomplete the result follows by Fact 2.2.1(i). □

In LSS Binary Products and Coproducts Commute with ω-colimits

As a consequence of both ω-colimits and binary products being created via U we obtain the following lemma.

Lemma 8.1.15 *Binary products commute with ω-colimits, i.e., the bifunctor* $_ \amalg _ : \mathbf{LSS} \times \mathbf{LSS} \to \mathbf{LSS}$ *is a ω-functor.*

Proof. Follows immediately from Proposition 8.1.10(iv) and Proposition 8.1.14(v), by application of Fact 2.2.1(iv). □

On the other hand the continuity of the coproduct bifunctor is an example of general nonsense. We have the following lemma.

Lemma 8.1.16 *Binary coproducts commute with ω-colimits, i.e., the bifunctor* $_ \amalg _ : \mathbf{LSS} \times \mathbf{LSS} \to \mathbf{LSS}$ *is a ω-functor.*

Proof. The proof is an exaple of a 'general nonsense', see [Mac Lane 71][p.115]. By Proposition 8.1.12 **LSS** has binary coproducts, which is equivalent to the diagonal functor $\Delta^2_{\mathbf{C}} : \mathbf{LSS} \to \mathbf{LSS}^2$ having a left-adjoint. This left-adjoint sends pairs of lss-s to their coproduct. As a left-adjoint it automatically preserves all existing colimits, so ω-colimits in particular. □

LSS Is Complete. Is It Cocomplete?

We finish this paragraph with some remarks on the (co)completeness of **LSS**.

First, the following result is given without a proof.

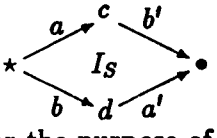
Lemma 8.1.17 *Functor U creates equalisers in* **LSS**.

As an immediate consequence it gives a completeness result for **LSS**.

Corollary 8.1.18 **LSS** *is (small) complete.*

Proof. Follows from Proposition 8.1.11 and Lemma 8.1.17 by result dual to Corollary 2 in [Mac Lane 71][p109]. □

It is not known if **LSS** is cocomplete as well. All that we do know is that coequalizers are not created via U.

Example 8.1.19 Let S be the following lss: 

— the labelling part is not described. It is to a large extent irrelevant. For the purpose of the argument let $M_S \neq \emptyset$, and let $l_S : A_S \to M_S$ be a constant (total) function.

We consider two morphisms $f, g : S \to S$. Let f be the identity of S. Let $g = \langle g_C, g_A, id_{M_S} \rangle$. Here, g_C swaps c and d, and returns the argument otherwise. g_A swaps a-s with b-s appropriately: $g_A(a) = b$, $g_A(b) = a$, $g_A(a') = b'$ and $g_A(b') = a'$.

The reader may verify that the above two are morphisms.

The coequalizer $h : G_S \to G$ of $\mathrm{U}_G f$ and $\mathrm{U}_G g$ in **ppGrph** exists. G has the following form:

$$\star \xrightarrow{a=b} c=d \xrightarrow{a'=b'} \bullet$$

The notation above suggests that $h = \langle h_C, h_A \rangle$ is a pair of quotient maps corresponding to equivalence relations generated by: $\{\langle c, d \rangle\}$, for h_C, and $\{\langle a, b \rangle, \langle a'b' \rangle\}$, for h_A.

Now, the problem is that one cannot enrich the structure of G to obtain a lss S_G, and have h as a morphism $h: S \to S_G$ at the same time.

Since $a\ I_S\ b$ in S the assumption that h is a morphism leads to $h_A(a)\ I_{S_G}\ h_A(b)$ in S_G. But $h_A(a) = h_A(b)$ contradicts irreflexivity of I_{S_G}. ◇

8.1.3 c^*-generated lss-s.

The basic feature of labelled skeletal systems inherited from Keller's transition systems is that one can use them to simulate dynamic behaviour of real systems. Thus, a lss S can have designated *present* state c, and it can in one step *evolve* to another state d, provided there exists some edge a in S such that $\partial_S^0(a) = c$ and $\partial_S^1(a) = d$.

So far the role played by the initial configuration remained somewhat obscure. In many applications it is natural to distinguish such a configuration. This is to stress that the systems are always started from this state. With the idea of dynamic behaviour in mind it is easy to realise that only the states that are reachable from the initial one during the process of evolution are of any real interest. This ideas are formalised by introducing the notion of c^*-generated systems.

Definition 8.1.20 A lss S is called *c^*-generated* iff $C_S = \{c \in C_S \mid \exists x \in A_S^* \,.\, c_S^* \xrightarrow{x} c\}$

Let **c^*-genLSS** denote the full subcategory of **LSS** determined by c^*-generated lss-s.

The notion of a c^*-generated system is not specific to labelled skeletal systems. In fact it can be defined for pointed graphs. The reader may notice that we have already faced a construction of c^*-generated Petri systems, and asynchronous systems from Petri nets.

We shall often use an alternative definition that appeals to the notions of a *reachable configuration* and a *reachable edge.*

Definition 8.1.21 Let S be a lss. Define, by simultaneous induction, the sets: $C_S^{[\infty]} \subseteq C_S$ and $A_S^{[\infty]} \subseteq A_S$ as the least such that:

(i) $c_S^* \in C_S^{[\infty]}$;

(ii) if $c \in C_S^{[\infty]}$ and $a \in A_S$ are such that $c = \partial_S^0(a)$ then $\partial_S^1(a) \in C_S^{[\infty]}$ and $a \in A_S^{[\infty]}$.

Now, it is obvious that for any lss S we have: S is c^*-generated iff $C_S = C_S^{[\infty]}$. Moreover, S c^*-generated implies $A_S = A_S^{[\infty]}$

The following result is often used when dealing with c^*-generated lss-s.

Lemma 8.1.22 *Let $f: S \to T$ be* **LSS***-morphism. Then*

$f_C(C_S^{[\infty]}) \subseteq C_T^{[\infty]}$ *and* $f_A(A_S^{[\infty]}) \subseteq A_T^{[\infty]}$.

Proof. Let $C = f_C^{-1}(C_T^{[\infty]}), A = \{a \in A_S | \partial_S^0(a) \in C\}$. First, note that $\langle C, A\rangle$ satisfies the conditions of the Definition 8.1.21, and therefore: $C_S^{[\infty]} \subseteq C$ and $A_S^{[\infty]} \subseteq A$.

From this it immediately follows that $f_C(C_S^{[\infty]}) \subseteq f_C(C) = f_C \circ f_C^{-1}(C_T^{[\infty]}) \subseteq C_T^{[\infty]}$.

Secondly, $f_A(A) \subseteq A_T^{[\infty]}$. For let $a \in A$ be such that $f_A(a)\downarrow$. Then

$\partial_T^0(f_A(a)) = f_C(\partial_S^0(a))$, by Definition 8.1.1.(i).
So, $\partial_T^0(f_A(a)) \in C_T^{[\infty]}$, by definition of C, since $\partial_S^0(a) \in C$.
Thus, $f_A(a) \in A_T^{[\infty]}$, by Definition 8.1.21.(ii).

As required. □

As an immediate consequence we obtain the following property that enables us to conclude that some lss-s are c^*-generated. It says, roughly, that a lss 'covered' by a family of c^*-generated systems is itself c^*-generated.

Corollary 8.1.23 *Let S be a lss, and $\mathcal{F}$ a nonempty family of* **LSS***-morphisms with c^*-generated domains and S as codomain. And assume that the following condition holds:*

$$\forall c \in C_S \exists f: T \to S \in \mathcal{F} \exists d \in C_T \,.\, c = f_C(d)\,.$$

Then S is c^*-generated.

Proof. Follows immediately from Lemma 8.1.22. □

Now, we show that c^*-**genLSS** is a (full) *coreflective* subcategory of **LSS**. To this end we show that the procedure of taking a 'reachable part' of a lss S is functorial and the functor we arrive at is the coreflector we are looking for.

Definition 8.1.24 Define c^*–GEN: **LSS** $\to$ c^*-**genLSS** on objects:

c^*–GEN$(S) = \langle C, c_S^*, A, \asymp, I, \partial^0, \partial^1, M_S, l\rangle$ where
$C = C_S^{[\infty]}$,
$A = A_S^{[\infty]}$,
$\asymp = \asymp_S \cap A^2$,
$I = I_S \cap A^2$,
$\partial^0, \partial^1: A \to C$ are given by $\partial^i(a) = \partial_S^i(a)$, for $i = 0, 1$, $a \in A$,
$l: A \to M_S$ is the restriction of l_S to A.

On morphisms: c^*–GEN$(f: S \to T) = \langle g_C, g_A, f_M\rangle$ where
$g_C: C_S^{[\infty]} \to C_T^{[\infty]}$ is given by: $g_C(c) = f_C(c)$, for $c \in C_S^{[\infty]}$,
$g_A: A_S^{[\infty]} \to A_T^{[\infty]}$ is given by: $g_A(a) = f_A(a)$, for $a \in A_S^{[\infty]}$.

Evidently, the object part of c^*–GEN extracts from a system a full subsystem generated by the sets of its reachable states and edges. The properties of c^*–GEN are collected bellow.

Proposition 8.1.25 *(i) c^*–GEN: **LSS** $\to$ c^*-**genLSS** is a functor,*

(ii) $c^*\text{–GEN}(S) = S$, *for* c^**-generated* S,

(iii) $c^*\text{–GEN}(S)$ *is forward stable, for forward stable* S;

(iv) $c^*\text{–GEN}$ *is a coreflector,*

(v) $c^*\text{–GEN}(S \amalg T) = c^*\text{–GEN}(S) \amalg c^*\text{–GEN}(T)$, *in particular* $c^*\text{–GEN}$ *preserves binary products,*

(vi) $c^*\text{–GEN}(S \amalg T) = c^*\text{–GEN}(S) \amalg c^*\text{–GEN}(T)$, *in particular* $c^*\text{–GEN}$ *preserves binary coproducts,*

(vii) $c^*\text{–GEN}(\mathbf{1}_{\mathbf{LSS}}) = \mathbf{1}_{\mathbf{LSS}}$ *is initial and terminal in* c^***-genLSS**, *so that* $c^*\text{–GEN}$ *preserves all finite products and coproducts in* c^***-genLSS**,

(viii) $c^*\text{–GEN}(S_{\rightarrow}) = \operatorname{colim} c^*\text{–GEN}(\Gamma)$ *where* $\Gamma = S_0 \xrightarrow{f^0} S_1 \xrightarrow{f^1} \dots$ *is a* ω*-diagram, and* $\phi\colon \Gamma \to S_{\rightarrow}$ *a colimiting cocone, i.e.,* $c^*\text{–GEN}$ *preserves* ω*-colimits.*

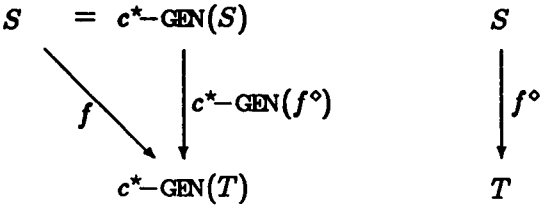
Proof. (i) One readily verifies that $\partial^i_S(A_S^{[\infty]}) \subseteq C_S^{[\infty]}$, for $i = 0, 1$. Hence, $c^*\text{–GEN}\, S$ is a lss as a full subsystem of S, by Fact 8.1.4, since the remaining conditions hold as well.

From Lemma 8.1.22, by Fact 8.1.4 it follows that $c^*\text{–GEN}\, f\colon c^*\text{–GEN}\, S \to c^*\text{–GEN}\, S'$, for $f\colon S \to S'$. The result now follows, since $c^*\text{–GEN}$ preserves identity, and composition of morphisms.

(ii) Obvious.

(iii) Similar to the proof that $c^*\text{–GEN}$ satisfies 8.1.1(iii). Note that for $a, b \in A_S^{[\infty]}$ the edges a', b' with appropriate properties that exist in S by Definition 8.1.1(v) do belong to $A_S^{[\infty]}$ as well. The rest is again trivial.

(iv) It is immediate to see that $c^*\text{–GEN}$ is right adjoint to the inclusion functor: c^***-genLSS** $\hookrightarrow$ **LSS**. For we see that for a c^*-generated S, a lss T, and any $f\colon S \to c^*\text{–GEN}\, T$ the diagram



commutes for a unique f°.

One readily sees that this f° has the components of f extended onto larger codomains.

(v) All we have to show is that:

$$C^{[\infty]}_{S \amalg T} = C^{[\infty]}_S \times C^{[\infty]}_T, \quad \text{and}$$
$$A^{[\infty]}_{S \amalg T} = A^{[\infty]}_S \times C^{[\infty]}_T \cup A^{[\infty]}_S \times A^{[\infty]}_T \cup C^{[\infty]}_S \times A^{[\infty]}_T.$$

Then the remaining items that built $c^*\text{–GEN}(S \amalg T)$ and $c^*\text{–GEN}(S) \amalg c^*\text{–GEN}(T)$, resp., are automatically equal.

To this end we have to show that the sets on the right above are minimal among those that do satisfy conditions in the Definition 8.1.21. We leave the task of showing that they indeed satisfy those conditions to the reader.

As for the minimality we note first that $C^{[\infty]}_{S\amalg T} \supseteq C^{[\infty]}_S \times C^{[\infty]}_T$. For any $c_\amalg = \langle c_S, c_T\rangle \in C^{[\infty]}_S \times C^{[\infty]}_T$ with $c^*_S \xrightarrow{x} c_S$ and $c^*_T \xrightarrow{y} c_T$ we have $c^*_\amalg \xrightarrow{z} c_\amalg$ where $z = x_\amalg y^\amalg$ with

$$x_\amalg = \langle a_1, c^*_T\rangle \ldots \langle a_m, c^*_T\rangle \qquad \text{where } x = a_1 \ldots a_m, \text{ and}$$
$$y^\amalg = \langle c_S, b_1\rangle \ldots \langle c_S, b_n\rangle \qquad \text{where } y = b_1 \ldots b_n.$$

Secondly, $A^{[\infty]}_{S\amalg T} \supseteq A^{[\infty]}_S \times C^{[\infty]}_T \cup A^{[\infty]}_S \times A^{[\infty]}_T \cup C^{[\infty]}_S \times A^{[\infty]}_T$. We concentrate attention on one case only, as the other two are quite similar. Let $a_\amalg = \langle a, d\rangle \in A^{[\infty]}_S \times C^{[\infty]}_T$ with $c_\amalg = \langle c_S, c_T\rangle = \partial^0_\amalg(a_\amalg)$. Then $c_S = \partial^0_S(a)$ and $c_T = d$, by definition of $\partial^0_\amalg$. Thus, $c_\amalg \in C^{[\infty]}_S \times C^{[\infty]}_T = C^{[\infty]}_{S\amalg T}$, so $a_\amalg \in A^{[\infty]}_{S\amalg T}$.

(vi) By the construction of binary coproducts in **LSS** we can apply Corollary 8.1.23 to the family $\{in_S, in_T\}$ to conclude that the binary coproduct of two c^*-generated systems is c^*-generated. The proof that the functor c^*–GEN actually preserves binary coproduct is based on the same idea as the one underlying the proof of (v) above. The details are left to the reader.

(vii) Trivial.

(viii) Another application of Corollary 8.1.23shows that a ω-colimit of c^*-generated systems is c^*-generated. Given a ω-diagram $\Gamma = S_0 \xrightarrow{f^0} S_1 \xrightarrow{f^1} \ldots$ let us take its colimiting cocone $\phi\colon \Gamma \to S_\rightarrow$ as constructed in the previous section. Then for $c_\rightarrow \in C_\rightarrow$ we have $c_\rightarrow = [c]_{\approx_\sigma}$, for some $n \in \omega$, $c \in C_n$, i.e., $c_\rightarrow = \phi_n(c)$. Hence, we can take $\mathcal{F} = \phi$.

As before, elementary calculations help to verify that the functor c^*–GEN preserves colimiting cocones of ω-diagrams. □

There are many consequences of the above proposition. At this stage it is important for us that the basic categorical constructions and properties are inherited by c^*-**genLSS** from **LSS**. In particular it is obvious that finite products commute in c^*-**genLSS** with ω-colimits. We obtain the following corollary.

Corollary 8.1.26 c^*-**genLSS** *has finite products, coproducts and ω-colimits inherited from* **LSS**, *i.e., the inclusion functor:* c^*-**genLSS** $\hookrightarrow$ **LSS** *creates them.*

Proof. Follows immediately from 8.1.25(v), 8.1.25(vi) and 8.1.25(viii), by 8.1.25(ii). □

Remark. It may be worthwhile to notice that although c^*-**genLSS** has all small products they are not, in general, inherited from **LSS** in the way finite products are. As a consequence of Proposition 8.1.25 we know how to compute them: we have to take the product in **LSS** and subsequently remove the unreachable configurations and edges with the help of c^*–GEN.

Functor c^*–GEN as a right adjoint preserves all limits that exist in **LSS**. Since the latter is (small) complete it follows that so is c^*-**genLSS**.

8.1.4 Labelled Skeletal Systems over a Fixed Alphabet

Throughout this subsection we let M, M', etc., run over alphabets, i.e., sets.

We define $\mathbf{LSS}^M$, the category of labelled skeletal systems over M, as the subcategory of **LSS** with objects sharing the set M as the set of labels, and those morphisms f that have id_M, i.e., the identity on M, as the relabelling part f_M. Similarly, the category of forward stable lss-s over M is defined.

Here is a formal definition.

Definition 8.1.27 Subcategory $\mathbf{LSS}^M$ of **LSS** of labelled skeletal systems *over* M, has as objects systems S that satisfy: $M_S = M$, and those morphism $f: S \to S'$ for which $f_M = id_M$.

$\mathbf{fsLSS}^M$ will denote the full subcategory of $\mathbf{LSS}^M$ with forward stable systems.

By convention we shall often forget to mention the M-part of a morphism when dealing with $\mathbf{LSS}^M$, and its subcategories.

The definition of $\mathbf{LSS}^M$ is an example of a more general categorical construction.

Let $F: \mathbf{C} \to \mathbf{D}$ be a functor. Then for any $D \in \mathbf{D}$ one can define a subcategory $\mathbf{C}^D$ of $\mathbf{C}$ by taking those objects which are mapped to D by F, and those morphisms which are mapped onto id_D. Category $\mathbf{C}^D$ is often called a *fibre* of $\mathbf{C}$ over D.

It should be evident that each $\mathbf{LSS}^M$ is just a fibre over M w.r.t., $\mathrm{U}_M: \mathbf{LSS} \to \mathbf{Pfn}$.

We are interested in properties of F sufficient to guarantee that each morphism $g: D \to D'$ in $\mathbf{D}$ determines a functor $g^\heartsuit: \mathbf{C}^{D'} \to \mathbf{C}^D$, and/or a functor $g_\heartsuit: \mathbf{C}^D \to \mathbf{C}^{D'}$. Such criteria exist.

Definition 8.1.28 **[Hyland et al 87]** A functor $F: \mathbf{C} \to \mathbf{D}$ is called *fibration* iff for any $g: D \to F\,C$ there exists $f: C' \to C$ such that

(i) $F\,f = g$, and

(ii) f is *cartesian*, i.e., for any $f': C'' \to C$ in $\mathbf{C}$, $g'': F\,C'' \to F\,C'$ such that $F\,f' = (F\,f) \circ g''$ in $\mathbf{D}$ there exists a unique $f'': C'' \to C'$ in $\mathbf{C}$ such that

a) $F\,f'' = g''$, and

b) $f' = f \circ f''$.

Dually, we arrive at the definition of the notion of op-fibration. Nevertheless, it is presented below.

Definition 8.1.29 A functor $F: \mathbf{C} \to \mathbf{D}$ is called *op-fibration* iff for any $g: F\,C \to D$ there exists $f: C \to C'$ such that

(i) $F\,f = g$, and

(ii) f is *cocartesian*, i.e., for any $f': C \to C''$ in $\mathbf{C}$, $g'': F\,C' \to F\,C''$ such that $F\,f' = g'' \circ F\,f$ in $\mathbf{D}$ there exists a unique $f'': C' \to C''$ in $\mathbf{C}$ such that

a) $F f'' = g''$, and

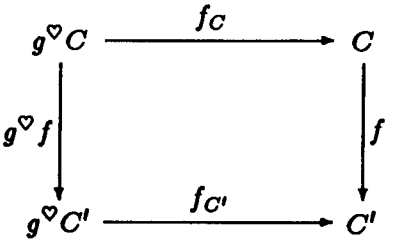
b) $f' = f'' \circ f$.

If $F: \mathbf{C} \to \mathbf{D}$ is a fibration then each morphism $g: D \to D'$ in $\mathbf{D}$ gives rise to a *pulling back* functor $g^{\heartsuit}: \mathbf{C}^{D'} \to \mathbf{C}^{D}$ defined as follows.

For any $C \in \mathbf{C}^{D'}$ let f_C be a fixed cartesian morphism with C as codomain, and s.t., $F f = g$.

We take its domain as the image of C via $g^{\heartsuit}$, i.e., $f_C: g^{\heartsuit}(C) \to C$.

If $f: C \to C'$ in $\mathbf{C}^{D'}$ then $g^{\heartsuit} f$ is the unique morphism that makes the following diagram commutes:



It exists since $f_{C'}$ is cartesian.

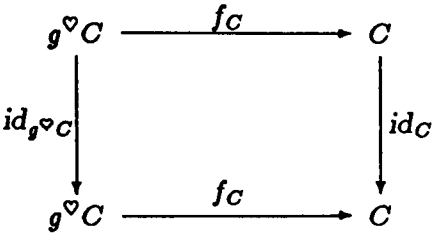
The construction of $g_{\heartsuit}: \mathbf{C}^{D} \to \mathbf{C}^{D'}$ resulting from an op-fibration $F: \mathbf{C} \to \mathbf{D}$ is dual.

The following result can be found in [Hyland et al 87].

Lemma 8.1.30 *(i) If $F: \mathbf{C} \to \mathbf{D}$ is a fibration then $g^{\heartsuit}: \mathbf{C}^{D'} \to \mathbf{C}^{D}$ is a functor, for any $g: D \to D'$.*

(ii) If $F: \mathbf{C} \to \mathbf{D}$ is an op-fibration then $g_{\heartsuit}: \mathbf{C}^{D} \to \mathbf{C}^{D'}$ is a functor, for any $g: D \to D'$.

Proof. (i) Standard. For example, to see that it preserves identities it is enough to check that the diagram



commutes, which is obviously the case. The equality $g^{\heartsuit}(id_C) = id_{g^{\heartsuit}(C)}$ follows then from the uniqueness property.

(ii) Dual to (i). □

The first result is almost immediate.

Proposition 8.1.31 $\mathbf{U}_M: \mathbf{LSS} \to \mathbf{Pfn}$ *is an op-fibration.*

Proof. Let S be a lss, M a set, and let $R: M_S \rightharpoonup M$ be a partial function.

Define $S[R] = \langle C_S, c^*_S, A_S, \asymp_S, I_S, \partial^0_S, \partial^1_S, M, R \circ l_S \rangle$, and $f_S = \langle id_{C_S}, id_{A_S}, R \rangle$.

Then, obviously, $S[R]$ is a lss, $f_S: S \to S[R]$ is a **LSS**-morphism, and $\mathrm{U}_M f_S = R$. It remains to verify that f_S is cocartesian.

Let us assume that $f': S[R] \to S''$, and $g'': M \to M_{S''}$ are such that $\mathrm{U}_M f' = g'' \circ \mathrm{U}_M f_S$.

Let $f'': S[R] \to S''$ be defined by: $f'' = \langle f'_C, f'_A, g'' \rangle$. It is a morphism. The only condition ever in doubt is 8.1.2(iv). But,

$$g'' \circ l_{S[R]} = g'' \circ R \circ l_S = f'_M \circ l_S,$$

as required.

Moreover, $f' = f'' \circ f_S$, by definition, and f'' is the only morphism that satisfies the required properties. □

It will be instructive to have the explicit form of the functor $R_\heartsuit: \mathbf{LSS}^M \to \mathbf{LSS}^{M'}$, for $R: M \rightharpoonup M'$ recorded.

Corollary 8.1.32 *For $R: M \rightharpoonup M'$ the functor $R_\heartsuit: \mathbf{LSS}^M \to \mathbf{LSS}^{M'}$ is defined by:*

$R_\heartsuit S = S[R]$ *on objects, and*

$R_\heartsuit f = \langle f_C, f_A, id_{M'} \rangle$ *on morphisms.*

Proof. Easy verification. □

It is not known if U_M is a fibration. What we can offer is a proof that it is a fibration when appropriately modified. Luckily, this will be of interest later.

Let $\mathbf{Pfn}^!$ be a subcategory of **Pfn** with sets as objects, and partial injective functions as morphisms. Let us remind that a partial function $f: X \rightharpoonup Y$ is *injective* iff $f(x)\downarrow \wedge f(x) = f(x') \Rightarrow x = x'$, for $x, x' \in X$.

The class of partial injective functions contains all monos of **Pfn**, i.e., total injective functions, and is closed w.r.t., composition. Hence, $\mathbf{Pfn}^!$ is indeed a category.

Moreover, it is closed under taking inverses, i.e., for a partial injective $f: X \rightharpoonup Y$ its inverse f^{-1} is also a partial injective $f^{-1}: Y \rightharpoonup X$. If $f: X \rightharpoonup Y$ is injective, and $g: Z \rightharpoonup X$ satisfies $g(Z) \subseteq \mathrm{dom}\, f$ then $g = f^{-1} \circ f \circ g$.

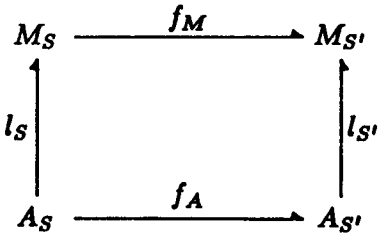
A partial characteristic function $\chi_X: Y \rightharpoonup Y$, for $X \subseteq Y$, is a typical injective. It is defined by:

$$\chi_X(y) = y \quad \text{if } y \in X, \quad \text{for } y \in Y.$$

Let $\mathbf{LSS}^!$ denote the following modification of **LSS**. It has the same objects as **LSS**, but different morphisms. As in the case of **LSS**, a morphism $f: S \to S'$ in $\mathbf{LSS}^!$ is a triple $f = \langle f_C, f_A, f_M \rangle$, where f_C and f_A are as before, and satisfy conditions (i)–(iii) of Definition 8.1.2. Only injective partial functions are allowed as the relabelling part of a morphism. Finally, condition 8.1.2(iv) is replaced by a weaker requirement:

$$l_{S'} \circ f_A(a)\downarrow \quad \text{implies} \quad l_{S'} \circ f_A(a) = F_M \circ l_S(a), \quad \text{for } a \in A_S. \tag{8.1}$$

Pictorially, we do not require that the diagram:



commutes, as we did before. Only when the path $l_{S'} \circ f_A$ is defined so should be the other, and the results should be equal.

The proof of the following result is left to the reader.

Lemma 8.1.33 $\mathbf{LSS}^!$ *with componentwise composition of morphisms, and identities inherited from* $\mathbf{LSS}$ *is a category. Moreover,* $\mathrm{U}_M\colon \mathbf{LSS}^! \to \mathbf{Pfn}^!$ *defined as before is a functor.*

Proof. Omitted. □

The reader should notice that any $\mathbf{LSS}$-morphism with the relabelling part of morphisms being injective is automatically a $\mathbf{LSS}^!$-morphism. Therefore, the fibre over M determined by $\mathrm{U}_M\colon \mathbf{LSS}^! \to \mathbf{Pfn}^!$ contains $\mathbf{LSS}^M$ as a subcategory.

We can show that after this modification U_M becomes a fibration.

Proposition 8.1.34 $\mathrm{U}_M\colon \mathbf{LSS}^! \to \mathbf{Pfn}^!$ *is a fibration.*

Proof. Let $S \in \mathbf{LSS}^!$, and $R\colon M \rightarrowtail M_S$ be injective.

Define $S' = \langle C_S, c_S^\star, A, \asymp, I, \partial^0, \partial^1, M_S, l\rangle$ to be the full subsystem of S generated by C_S and A, with

$$A = \{a \in A_S \mid l_S(a)\uparrow \ \vee\, l_S(a) \in \mathrm{dom}\, R^{-1}\}.$$

From Fact 8.1.4 it follows that S' is a lss, and that $f'_S = \langle id_{C_S}, \imath_A, id_{M_S}\rangle$ where $\imath_A\colon A \hookrightarrow A_S$ an inclusion, is a morphism $f'_S\colon S' \to S$ in $\mathbf{LSS}$. Since id_{M_S} is obviously injective it follows that f'_S is a $\mathbf{LSS}^!$-morphism.

Let us define $S \setminus\!\setminus R = S'[R^{-1}]$, i.e.,

$$S \setminus\!\setminus R = \langle C_S, c_S^\star, A, \asymp, I, \partial^0, \partial^1, M, R^{-1} \circ l\rangle.$$

From Corollary 8.1.32 it follows that $S \setminus\!\setminus R$ is a lss over M.

Triple $f_S = \langle id_{C_S}, \imath_A, R\rangle$ is a morphism. Since $l(A) \subseteq \mathrm{dom}\, R^{-1}$ implies $R \circ (R^{-1} \circ l) = l = l_S \circ \imath_A$ it follows that the condition 8.1.2(iv) holds, and so (8.1) as well. Satisfaction of the remaining conditions is inherited by f_S from f'_S.

Let us verify that f_S is cartesian.

Let $f\colon S'' \to S$ be a $\mathbf{LSS}^!$-morphism, and $g''\colon M_{S''} \rightarrowtail M$ injective, s.t., $\mathrm{U}_M f = \mathrm{U}_M f_S \circ g''$, i.e., $f_M = R \circ g''$. We have to find a unique $f''\colon S'' \to S \setminus\!\setminus R$ such that $f = f_S \circ f''$ and $\mathrm{U}_M f'' = f''_M = g''$.

It is easy to see that such a morphism is unique, if it exists. For then $f''_M = g''$, $f''_C = f_C$, and f''_A must be a restriction of f_A to the codomain A. Let us see that such a triple determines a morphism.

First, we must check that f''_A is well-defined, i.e., that $f_A(A_{S''}) \subseteq A$. To this end let $a \in A_{S''}$ be such that $f_A(a)\downarrow$. There are two cases to consider.

If $l_S(f_A(a))\uparrow$ then $f_A(a) \in A$ follows directly from the definition of A.

If $l_S(f_A(a))\downarrow$ then, since f is a morphism,

$$l_S(f_A(a)) = f_M(l_{S''}(a)) = R(g''(l_{S''}(a))) \in \mathrm{dom}\, R^{-1}.$$

Hence, $f_A(a) \in A$ in this case as well.

Moreover, f'' is a morphism in $\mathbf{LSS}^*$. The first three conditions of the Definition 8.1.2 are satisfied since, except for the parts related to labelling, f'' is a restriction of f, and $S \setminus\setminus R$ is a full subsystem of S.

Condition (8.1) is also satisfied since if $l_{S\setminus\setminus R} \circ f''(a)\downarrow$ then also $l_S(f_A(a))\downarrow$, since $l_S(f_A(a) = l(f''_A(a))$, and $l_{S\setminus\setminus R} = R^{-1} \circ l$. Therefore,

$$l_{S\setminus\setminus R} \circ f''_A(a) = R^{-1} \circ l_S \circ f_A(a) = R^{-1} \circ R \circ g'' \circ l_{S''}(a) = g'' \circ l_{S''}(a).$$

as required. The last equality holds since $R(g''(l_{S''}(a))) \in \mathrm{dom}\, R^{-1}$, as shown already before. □

Remark. The cartesian morphism constructed in the proof above does not satisfy 8.1.2(iv) in general. It would be more elegant to show that the original functor is a fibration, but this seems unlikely. Alternatively, we could perhaps develop a theory based on more general morphisms — with (8.1) in place of 8.1.2(iv). The author does not know if this can be done conservatively, i.e., with all the results established so far being valid, and with minimal changes to the proofs. Therefore, the temptation to make the theory look better was abandoned. Besides, it is not certain that the proof that U_M is a fibration in this general case would go, i.e., if one can generalise from injective to arbitrary partial function.

With the notation from the last proof we can characterise the functor $R^\heartsuit$ generated by injective $R\colon M \rightharpoonup M'$ as follows.

Corollary 8.1.35 *Functor* $R^\heartsuit\colon \mathbf{LSS}^{M'} \to \mathbf{LSS}^M$ *for injective* $R\colon M \rightharpoonup M'$, *is defined by:*

$R^\heartsuit S = S \setminus\setminus R$, *on objects, and*

$R^\heartsuit f = \langle f_C, \overline{f}_A, \mathrm{id}_M \rangle$, *with* $\overline{f}_A$ *an appropriate restriction of* f_A *— on morphisms.*

Proof. Straightforward verification. □

Our next result shows that every $\mathbf{LSS}^M$ is a ω-category.

Lemma 8.1.36 *Let* M *be a set. Then* $\mathbf{LSS}^M$ *and* $\mathbf{fsLSS}^M$ *are* ω*-categories. More specifically, the inclusions:* $\mathbf{LSS}^M \hookrightarrow \mathbf{LSS}$ *and* $\mathbf{fsLSS}^M \hookrightarrow \mathbf{fsLSS}$ *create* ω*-colimits.*

Proof. We consider the case of $\mathbf{LSS}^M$. It implies that the result holds for $\mathbf{fsLSS}^M$ by Proposition 8.1.14(ii).

Let $\Gamma = S_0 \xrightarrow{f^0} S_1 \xrightarrow{f^1} \ldots$ be a ω-diagram in $\mathbf{LSS}^M$. It is a ω-diagram in $\mathbf{LSS}$, so has a colimiting cocone $\phi: \Gamma \rightarrow S_{\rightarrow}$, as asserted by Proposition 8.1.14. Because of the very special form of the morphisms involved, i.e., $f_M = id_M$, one readily sees that, with notation of the Proposition 8.1.14, $M_{\rightarrow}$ is isomorphic to M via $h_M: M_{\rightarrow} \rightarrow M$, where

$$h_M([m]_{\approx_M}) = m, \qquad \text{for } m \in M.$$

It is trivial to conclude that $h = \langle id_{C_{\rightarrow}}, id_{A_{\rightarrow}}, h_M \rangle$ is an isomorphism of $S_{\rightarrow}$ and $S^M_{\rightarrow} = \langle C_{\rightarrow}, c^*_{\rightarrow}, A_{\rightarrow}, \asymp_{\rightarrow}, I_{\rightarrow}, \partial^0_{\rightarrow}, \partial^1_{\rightarrow}, M, l \rangle$, where

$$l([a]_{\approx_A}) = l_n(a), \qquad \text{for } a \in A_n \cap \mathrm{dom}(\approx_A).$$

The result now follows. □

Our next observation shows that every $\mathbf{LSS}^M$ and every $\mathbf{fsLSS}^M$ has a zero object. These are obviously not inherited from $\mathbf{LSS}$, unless $M = \emptyset$. We give the following result without proof.

Lemma 8.1.37 *A forward stable lss $\mathbf{1}_{A_M} = \langle \{\star\}, \star, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, M, \emptyset \rangle$ is an initial object in $\mathbf{LSS}^M$, and therefore in $\mathbf{fsLSS}^M$ as well.*

We shall not pursue the studies of $\mathbf{LSS}^M$ or $\mathbf{fsLSS}^M$ any further. In particular, we leave the question of existence in $\mathbf{LSS}^M$ of various (co)limits unanswered. Although, it can be easily shown that neither of them is complete or cocomplete.

Let us only remark that the categories do not, in general, inherit neither coproducts, nor products from $\mathbf{LSS}$ and $\mathbf{fsLSS}$, respectively. Moreover, Example 8.1.19 can be modified to show that no $\mathbf{LSS}^M$, for $M \neq \emptyset$, has coequalisers — simply because the pair of parallel morphisms given there cannot be coequalised, i.e., made equal by extending each with the same morphism. This also shows that $\mathbf{LSS}^M$ has no terminal objects.

On the other hand, and it will be crucial later, both product and coproduct, cut down to continuous bifunctors on fibres. We have the following result.

Lemma 8.1.38 *Let M, M' be sets. Then*

(i) $_ \amalg _ : \mathbf{LSS}^M \times \mathbf{LSS}^{M'} \rightarrow \mathbf{LSS}^{M \amalg_{\mathbf{p}} M'}$ *is a continuous functor.*

(ii) $_ \amalg _ : \mathbf{LSS}^M \times \mathbf{LSS}^{M'} \rightarrow \mathbf{LSS}^{M \amalg_{\mathbf{p}} M'}$ *is a ω-functor.*

(iii) We can replace $\mathbf{LSS}^M$ by $\mathbf{fsLSS}^M$ in (i) and (ii) above.

Proof. (i) One method would be to calculate directly from the definitions that for morphisms: $f \in \mathbf{Hom}_{\mathbf{LSS}^M}$ and $g \in \mathbf{Hom}_{\mathbf{LSS}^{M'}}$, with $f_M = id_M$, $g_M = id_{M'}$ one has $(f \amalg g)_M = id_{M \amalg_{\mathbf{p}} M'}$. This, however, follows immediately from the fact that $\mathbf{U}_M: \mathbf{LSS} \rightarrow \mathbf{Pfn}$ preserves, since $\mathbf{U} = \mathbf{U}_G \times \mathbf{U}_M$ creates, products.

As for the continuity is concerned it follows immediately from the continuity of binary products in **LSS**, Lemma 8.1.15, and the recently proved Lemma 8.1.36. The last one assures that for ω-diagrams: Γ, Γ' in $\mathbf{LSS}^M$ and $\mathbf{LSS}^{M'}$, resp., both $\mathrm{colim}(\Gamma \amalg \Gamma')$ and $(\mathrm{colim}\,\Gamma) \amalg (\mathrm{colim}\,\Gamma')$ are in $\mathbf{LSS}^{M \amalg_{\mathrm{p}} M'}$.

(ii) The proof here is an evident modification of the (i), based on Lemma 8.1.16.

(iii) Follows from (i) and (ii) since the products and coproducts of forward stable lss-s are forward stable, by Proposition 8.1.10(ii) and Proposition 8.1.12(ii). □

8.1.5 $\mathcal{A}_M$

Finally, we can define a class of ω-categories $\mathcal{A}_M$, for every set of labels M. One of those categories, namely $\mathcal{A}_M$, will be used in the next Section to give $\mathcal{L}$ denotational semantics.

Definition 8.1.39 Let M be a set. Define $\mathcal{A}_M$ to be a full subcategory of $\mathbf{fsLSS}^M$ with c^*-generated labelled skeletal systems over the alphabet M.

In the next section one of them, more specifically $\mathcal{A}_M$, will be used to give $\mathcal{L}$ an interpretation. In fact though, many of them will be used indirectly. We finish this paragraph by proving a series of lemmas. They say that the restrictions of the product and coproduct bifunctors are continuous. But first we have to show that every $\mathcal{A}_M$ is indeed a ω-category.

Lemma 8.1.40 *$\mathcal{A}_M$ is a ω-category, with $\mathbf{1}_{\mathcal{A}_M}$ as initial object.*

Proof. From Lemma 8.1.36 we know that $\mathbf{fsLSS}^M$ inherits the ω-colimits from **fsLSS**. By Corollary 8.1.26 it follows that the same is true about $\mathcal{A}_M$ in turn: it inherits ω-colimits from $\mathbf{fsLSS}^M$.

We have noticed that $\mathbf{1}_{\mathcal{A}_M}$ is a zero object in $\mathbf{fsLSS}^M$. But it is trivially c^*-generated, so $\mathbf{1}_{\mathcal{A}_M} \in \mathcal{A}_M$, as required. □

Lemma 8.1.41 $_ \amalg _ : \mathcal{A}_M \times \mathcal{A}_{M'} \to \mathcal{A}_{M \amalg_{\mathrm{p}} M'}$ *is a ω-functor.*

Proof. From Lemma 8.1.38(iii) we know that $_ \amalg _ : \mathbf{fsLSS}^M \times \mathbf{fsLSS}^{M'} \to \mathbf{fsLSS}^{M \amalg_{\mathrm{p}} M'}$ is a continuous functor. Since for any M'' the category $\mathcal{A}_{M''}$ inherits ω-colimits from $\mathbf{fsLSS}^{M''}$ the result follows. □

Lemma 8.1.42 $_ \amalg _ : \mathcal{A}_M \times \mathcal{A}_{M'} \to \mathcal{A}_{M \amalg_{\mathrm{p}} M'}$ *is a ω-functor.*

Proof. Similar to the proof of Lemma 8.1.41 □

8.2 Interpretation of $\mathcal{L}$ in $\mathcal{A}_M$

We apply the results of the previous section to give $\mathcal{L}$ a semantics in $\mathcal{A}_M$. In the following paragraphs we shall define interpretation for each of the $\mathcal{L}$ operators as a ω-functor on $\mathcal{A}_M$ of appropriate arity. From the general theory discussed in Chapter 6 it follows that it is all we ought to do.

In what follows we comply to the tradition that the functors bear the same names as the symbols they interpret. In that way the intentional use becomes clear. We hope, with the others, that it will be always obvious from the context whether we are dealing with syntax, or its semantic counterpart.

By convention, the relabelling part of morphisms in categories $\mathcal{A}_M$ is not mentioned. After all in any $\mathcal{A}_M$-morphism f we have $f_M = id_M$.

8.2.1 Interpretation of Ω

Since Ω is a nullary symbol we should give it as an interpretation a ω-functor of the type: $\Omega : \mathbf{1_{Cat}} \to \mathcal{A}_M$, where $\mathbf{1_{Cat}} = \mathcal{A}^0_M$ is a category with one object and one morphism – a terminal object in Cat.

Any functor $F : \mathbf{1_{Cat}} \to \mathbf{C}$, where C is a category, is in fact a constant functor: $F = K_C$ where $C \in \mathbf{C}$ is the image of the unique object of $\mathbf{1_{Cat}}$. Conversely, any object $C \in \mathbf{C}$ determines the continuous constant functor K_C.

We are therefore entitled to define:

$$\Omega = K_{\mathbf{1}_{\mathcal{A}_M}}.$$

The observations made above entail the following result.

Lemma 8.2.1 $\Omega : \mathbf{1_{Cat}} \to \mathcal{A}_M$ *is a continuous functor.*

Proof. As remarked in Chapter 6 the category $\mathbf{1_{Cat}}$ is a ω-category, and every constant functor is continuous. □

8.2.2 Interpretation of $\mu\cdot_$

Semantically, prefixing and creating the null lss, are the basic and most important operators in our language. All the other operators, as we shall see later, only manipulate what was earlier created.

The rôle of Ω can be expressed quickly as creation of a simple system that can do nothing. All it has is an initial configuration. The process of creation of new states and actions is left entirely to the prefixing operator: $\mu\cdot_ : \mathcal{A}_M \to \mathcal{A}_M$.

The idea is simple. Assume we are given a lss S. To obtain $\mu\cdot S$ we add a new initial state $\mathsf{new}_C(S)$ and a new edge $\mathsf{new}_A(S)$ labelled by μ which leads from the new initial state to the old one. We assume that new_C, and new_A act as functions with the property that $\mathsf{new}_C(S) \notin C_S$, and $\mathsf{new}_A(S) \notin A_S$, for any lss S. Otherwise we leave the structure of S undisturbed.

In particular we want the resulting system to have the sameness and independency relations being conservative extensions of $\asymp_S$ and I_S, respectively. This determines $\mu\cdot S$ uniquely.

The following definition aims at formalising the above ideas.

Definition 8.2.2 Let $\mu \in \mathcal{M}$. Define functor $\mu\cdot_ : \mathcal{A}_{\mathcal{M}} \to \mathcal{A}_{\mathcal{M}}$.

On objects: $\mu\cdot S = S'$ where

$C_{S'} = C_S \cup \{\mathsf{new}_C(S)\}$ an extra configuration is added,

$c^*_{S'} = \mathsf{new}_C(S)$ and made initial,

$A_{S'} = A_S \cup \{\mathsf{new}_A(S)\}$ a new edge is added,

$\asymp_{S'} = \asymp_S \cup \{\langle \mathsf{new}_A(S), \mathsf{new}_A(S)\rangle\}$,

$I_{S'} = I_S$,

$$\partial^0_{S'}(a) = \begin{cases} \mathsf{new}_C(S) & a = \mathsf{new}_A(S) \\ \partial^0_S(a) & \textit{otherwise} \end{cases} \qquad \partial^1_{S'}(a) = \begin{cases} c^*_S & a = \mathsf{new}_A(S) \\ \partial^1_S(a) & \textit{otherwise} \end{cases},$$

$M_{S'} = M_S = \mathcal{M}$,

$$l_{S'}(a) = \begin{cases} \mu & a = \mathsf{new}_A(S) \\ l_S(a) & \textit{otherwise} \end{cases}.$$

On morphisms: $\mu\cdot(f : S \to T) = \langle f'_C, f'_A\rangle$ where

$$f'_C(c) = \begin{cases} \mathsf{new}_C(T) & c = \mathsf{new}_C(S) \\ f_C(c) & \textit{otherwise} \end{cases}, \qquad f'_A(a) = \begin{cases} \mathsf{new}_A(T) & a = \mathsf{new}_A(S) \\ f_A(a) & \textit{otherwise} \end{cases}.$$

The result that now follows shows not only that we have defined a functor, but a continuous one as well.

Lemma 8.2.3 *$\mu\cdot_ : \mathcal{A}_{\mathcal{M}} \to \mathcal{A}_{\mathcal{M}}$ is a ω-functor, for any $\mu \in \mathcal{M}$.*

Proof. We omit what by now should become standard and easy, i.e., the verification that $\mu\cdot_$ is a functor, and concentrate on more interesting task that remains.

Assume $\Gamma = S_0 \xrightarrow{f^0} S_1 \xrightarrow{f^1} \ldots$ is a ω-diagram. Then so is its image via functor $\mu\cdot_$: $\overline{\Gamma} = \overline{S_0} \xrightarrow{\overline{f^0}} \overline{S_1} \xrightarrow{\overline{f^1}} \ldots$, where $\overline{S_n} = \mu\cdot S_n$, and $\overline{f^n} = \mu\cdot f^n$, for $n \in \omega$.

Let us take the colimiting cocones $\phi : \Gamma \to S_\rightarrow$ and $\overline{\phi} : \overline{\Gamma} \to \overline{S_\rightarrow}$, as constructed in the previous section. Then clearly $\psi : \overline{\Gamma} \to \mu\cdot S_\rightarrow$, where $\psi_n = \mu\cdot\phi_n$, is a cocone. Therefore we have the mediating morphism $h : \overline{S_\rightarrow} \to S_\rightarrow$ which, with the notation from the previous section, is given by:

$$h_C([c]_{\approx_{\overline{C}}}) = \begin{cases} \mathsf{new}_C(S_\rightarrow) & c = \mathsf{new}_C(S_n) \\ [c]_{\approx_C} & \textit{otherwise} \end{cases}, \qquad \text{for } c \in C_n, \text{ and}$$

$$h_A([a]_{\approx_{\overline{A}}}) = \begin{cases} \mathsf{new}_A(S_\rightarrow) & a = \mathsf{new}_A(S_n) \\ [a]_{\approx_A} & \textit{otherwise} \end{cases}, \qquad \text{for } a \in A_n.$$

First observation is that the functions defined above are bijections, with h_A total. They are onto: $\mathsf{new}_C(S_\rightarrow) = h_C([\mathsf{new}_C(S_0)]_{\approx_{\overline{C}}})$ and $[c]_{\approx_C} = h_C([c]_{\approx_{\overline{C}}})$, and similarly in case of h_A. To see that they are 1–1 one should notice that for $c \in C_{\overline{S_n}}$ we have: $c \approx_{\overline{C}} \mathsf{new}_C(S_m)$ iff $c = \mathsf{new}_C(S_n)$, and analogically in case of h_A.

And finally, the pair $h^{-1} = \langle h_C^{-1}, h_A^{-1} \rangle$ constitutes a morphism. The conditions of the Definition 8.1.2 are clearly satisfied. We leave the details to the reader. □

8.2.3 Interpretation of $_ \setminus L$

Let $L \subseteq \mathcal{M}$. The interpretation of the operation of restriction with respect to L should be a ω-functor: $_ \setminus L : \mathcal{A}_{\mathcal{M}} \to \mathcal{A}_{\mathcal{M}}$.

The first idea is to take a lss S and remove all its edges which have labels in L. Unfortunately, this can, and usually does, produce systems which are not c^*-generated.

Example 8.2.4 Consider the following labelled skeletal systems:

S $\qquad\qquad$ S'



Both have empty sameness and independency relations.

Let us assume moreover that a is the only edge labelled with μ, and that $L = \{\mu\}$. Then S after removal of a gives the non-c^*-generated lss S'. ◇

Thus, we approach the problem in two stages. First, we define a functor that formalises our rough idea. Secondly, we shall take care of what was made unreachable with the help of c^*–GEN.

Definition 8.2.5 Let $L \subseteq \mathcal{M}$. Define a functor $_ \setminus\setminus L : \mathcal{A}_{\mathcal{M}} \to \mathbf{fsLSS}^{\mathcal{M}}$ as follows.

On objects: $S \setminus\setminus L = S'$ where

$C_{S'} = C_S$;

$c^*_{S'} = c^*_S$;

$A_{S'} = A_S - l_S^{-1}(L)$;

$\asymp_{S'} = \asymp_S \cap A^2_{S'}$;

$I_{S'} = I_S \cap A^2_{S'}$;

$\partial^i_{S'}(a) = \partial^i_S(a)$; for $i = 0, 1$, $a \in A_{S'}$;

$M_{S'} = M_S = \mathcal{M}$;

$l_{S'}(a) = l_S(a)$; for $a \in A_{S'}$.

And on morphisms: $(f : S \to T) \setminus\setminus L = \langle f'_C, f'_A \rangle$ where

$f'_C = f_C$, and

$f'_A(a) = f_A(a)$, for $a \in A_{S'}$.

The reader should notice that the definition of the functor $_ \setminus\setminus L$ coincides with that of the functor $\chi^{\heartsuit}_{\mathcal{M}-L}$ resulting from the fibration $\mathbf{U}_M$: $\mathbf{LSS}^* \to \mathbf{Pfn}^*$. Above, $\chi_{\mathcal{M}-L}$: $\mathcal{M} \to \mathcal{M}$ denotes the partial characteristic function of the set $\mathcal{M} - L$ in $\mathcal{M}$.

The result that follows confirms that $_ \setminus\setminus L$ is a continuous functor.

Lemma 8.2.6 *$_ \setminus\setminus L : \mathcal{A}_{\mathcal{M}} \to \mathbf{fsLSS}^{\mathcal{M}}$ is a ω-functor, for any $L \subseteq \mathcal{M}$.*

Proof. It is easy to check that $S \setminus\setminus L$ is a forward stable lss, for a forward stable lss S. We leave this and the verification that $_ \setminus\setminus L$ is a functor to the reader.

Assume $\Gamma = S_0 \xrightarrow{f^0} S_1 \xrightarrow{f^1} \ldots$ is a ω-diagram in $\mathbf{fsLSS}^{\mathcal{M}}$. Then so is its image via functor $_ \setminus\setminus L$: $\overline{\Gamma} = \overline{S_0} \xrightarrow{\overline{f^0}} \overline{S_1} \xrightarrow{\overline{f^1}} \ldots$, where $\overline{S_n} = S_n \setminus\setminus L$, and $\overline{f^n} = f^n \setminus\setminus L$, for $n \in \omega$.

Let us take the colimiting cocones $\phi : \Gamma \to S_{\rightarrow}$ and $\overline{\phi} : \overline{\Gamma} \to \overline{S_{\rightarrow}}$, as constructed in the previous section. Then clearly $\psi : \overline{\Gamma} \to S_{\rightarrow} \setminus\setminus L$, where $\psi_n = \phi_n \setminus\setminus L$, is a cocone. Therefore we have the mediating morphism $h : \overline{S_{\rightarrow}} \to S_{\rightarrow}$ which, with the notation from the previous section, is given by:

$$h_C([c]_{\approx_{\overline{C}}}) = [c]_{\approx_C}, \qquad \text{for } c \in C_n, \text{ and}$$
$$h_A([a]_{\approx_{\overline{A}}}) = [a]_{\approx_A}, \qquad \text{for } a \in A_n.$$

It is now easy to see that h is an isomorphism. The details are left as an exercise. □

Now, we are ready to define the interpretation of the restriction w.r.t., L -operator, for any set of labels L. We put

$$_ \setminus L = c^*\text{-GEN} \circ _ \setminus\setminus L.$$

We can assert immediately that $_ \setminus L$ has the desired property.

Lemma 8.2.7 *For any $L \subseteq \mathcal{M}$ the functor $_ \setminus L : \mathcal{A}_{\mathcal{M}} \to \mathcal{A}_{\mathcal{M}}$ is a ω-functor.*

Proof. $_ \setminus L$ is continuous as a composition of continuous functors, by Lemma 8.1.25 and Lemma 8.2.6. □

8.2.4 Interpretation of $_[R]$

We define here a construction more general that is necessary to interpret just renaming operator. This generality appears very handy when it comes to interpret the coproduct: $_ \# _$, and product: $_ \,\&\, _$ -operators of $\mathcal{L}$.

For any partial map $R: M \rightharpoonup M'$ we define a ω-functor $_[R]: \mathcal{A}_M \to \mathcal{A}_{M'}$. In the case $R: \mathcal{M} \rightharpoonup \mathcal{M}$ this will give us the required interpretation of the renaming-hiding operator.

As usual, we use the same notation for the syntactic as well as semantic objects.

Definition 8.2.8 Let $R: M \rightharpoonup M'$. A functor $_[R]: \mathcal{A}_M \to \mathcal{A}_{M'}$ is defined as follows.

On objects: $S[R] = \langle C_S, c^*_S, A_S, \asymp_S, I_S, \partial^0_S, \partial^1_S, M', R \circ l_S \rangle$, and

on morphisms: $(f: S \to T)[R] = \langle f_C, f_A \rangle : S[R] \to T[R]$.

Again, we note that the above definition coincides with the definition of the functor $R_{\heartsuit}: \mathbf{LSS}^M \to \mathbf{LSS}^{M'}$ that results from $\mathbf{U}_M: \mathbf{LSS}^{\rightarrow}\mathbf{Pfn}$ being an op-fibration. Since such $R_{\heartsuit}$ changes only the part of the system related to labelling it is evident that it cuts down

to a functor $R_\heartsuit: \mathcal{A}_M \to \mathcal{A}_{M'}$ — forward stability and c^*-generatedness do not depend on the labelling part of the definition of a lss.

For the same reason it is continuous.

Lemma 8.2.9 *For any $R: M \rightharpoonup M'$ the functor $_[R]: \mathcal{A}_M \to \mathcal{A}_{M'}$ is a ω-functor.*

Proof. In the proof of Lemma 8.1.36 it was argued that colimits in $\mathbf{LSS}^M$ are inherited from **LSS**. Thus, $\mathcal{A}_M$ inherits them as well, by 8.1.14(i), 8.1.25(viii) and 8.1.25(ii). In other words the labelling plays no role in deciding whether or not a cocone is colimiting. Therefore, its change has no effect — colimiting cocone remains colimiting. □

8.2.5 Interpretation of _ # _

Interpretation of _ # _ operator of $\mathcal{L}$ is defined in terms of coproduct bifunctor and an appropriate relabeling.

Recall that $\mathcal{M}$ is a $\Sigma^{\mathcal{M}}$-algebra, for $\Sigma^{\mathcal{M}} = \Sigma_1^{\mathcal{M}} \cup \Sigma_2^{\mathcal{M}}$ where

$\Sigma_1^{\mathcal{M}} = \{\mathtt{inl}_, \mathtt{inr}_, \langle _, \star\rangle, \langle \star, _\rangle\}$ and

$\Sigma_2^{\mathcal{M}} = \{\langle _, _\rangle\}$

$\mathcal{M}$ was specially defined so that we can interpret both $\mathcal{M} \amalg_p \mathcal{M}$ and $\mathcal{M} \amalg_p \mathcal{M}$ as 'subsets' of $\mathcal{M}$ itself. To formalise the former let us define an injective $i_{\mathcal{M}}: \mathcal{M} \amalg_p \mathcal{M} \rightharpoonup \mathcal{M}$ by:

$$i^{\#}_{\mathcal{M}}(m) = \begin{cases} \mathtt{inl}(m') & m = in_l(m') \\ \mathtt{inr}(m') & m = in_r(m') \end{cases}.$$

Now we are ready to formulate our definition of the functor interpreting _ # _.

Definition 8.2.10 Put $_ \# _ : \mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\mathcal{M}} \to \mathcal{A}_{\mathcal{M}}$ to be equal $_ \# _ = (_[i^{\#}_{\mathcal{M}}]) \circ (_ \amalg _)$.

Immediately, we conclude that the above is a ω-functor.

Lemma 8.2.11 *_ # _ is a ω-functor.*

Proof. _ # _ is a ω-functor as a composition of ω-functors:

$_ \amalg _ : \mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\mathcal{M}} \to \mathcal{A}_{\mathcal{M} \amalg_p \mathcal{M}}$, by Lemma 8.1.42;

$_[i^{\#}_{\mathcal{M}}] : \mathcal{A}_{\mathcal{M} \amalg_p \mathcal{M}} \to \mathcal{A}_{\mathcal{M}}$, by Lemma 8.2.9.

□

8.2.6 Interpretation of _ & _

The interpretation of _ & _ follows the same pattern.

First, we define a partial injective $i^{\&}_{\mathcal{M}} : \mathcal{M} \amalg_p \mathcal{M} \rightharpoonup \mathcal{M}$ by:

$$i^{\&}_{\mathcal{M}}(m) = \begin{cases} \langle n, n'\rangle & n = \pi_l(m) \wedge n' = \pi_r(m) \\ \langle n, \star\rangle & n = \pi_l(m) \wedge \pi_r(m)\uparrow \\ \langle \star, n'\rangle & \pi_l(m)\uparrow \wedge n' = \pi_r(m) \end{cases}$$

We can now formulate our definition of the functor interpreting _ & _.

Definition 8.2.12 Put $_ \,\&\, _ : \mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\mathcal{M}} \to \mathcal{A}_{\mathcal{M}}$ to be equal $_ \,\&\, _ = (_ [i^{\&}_{\mathcal{M}}]) \circ (_ \,\Pi\, _)$.

Immediately, we conclude that the above is a ω-functor.

Lemma 8.2.13 *$_ \,\&\, _$ is a ω-functor.*

Proof. $_ \,\&\, _$ is a ω-functor as a composition of ω-functors:

$$\begin{aligned} &_ \,\Pi\, _ : \mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\mathcal{M}} \to \mathcal{A}_{\mathcal{M}\,\Pi_p\,\mathcal{M}}, && \text{by Lemma 8.1.41;} \\ &_ [i^{\&}_{\mathcal{M}}] : \mathcal{A}_{\mathcal{M}\,\Pi_p\,\mathcal{M}} \to \mathcal{A}_{\mathcal{M}}, && \text{by Lemma 8.2.9.} \end{aligned}$$

□

8.2.7 Interpretation of $\mathcal{L}$

Let us finally collect all the pieces in one place. The (categorical) interpretation of $\mathcal{L}$:

$$\mathcal{A}_{\mathcal{M}}[\![_]\!] : \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var}) \to [\mathbf{Env}_{\mathcal{A}_{\mathcal{M}}} \to_{\omega} \mathcal{A}_{\mathcal{M}}]$$

is defined inductively for $x \in \mathsf{Var}$, $E, E' \in \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$ as follows:

$$\begin{aligned} \mathcal{A}_{\mathcal{M}}[\![x]\!] &= \Psi_x, \\ \mathcal{A}_{\mathcal{M}}[\![\Omega]\!] &= \Omega, \\ \mathcal{A}_{\mathcal{M}}[\![m\cdot E]\!] &= (m\cdot _) \circ \mathcal{A}_{\mathcal{M}}[\![E]\!], \\ \mathcal{A}_{\mathcal{M}}[\![E \setminus L]\!] &= (_ \setminus L) \circ \mathcal{A}_{\mathcal{M}}[\![E]\!], \\ \mathcal{A}_{\mathcal{M}}[\![E[R]]\!] &= (_ [R]) \circ \mathcal{A}_{\mathcal{M}}[\![E]\!], \\ \mathcal{A}_{\mathcal{M}}[\![E \,\#\, E']\!] &= (_ \,\#\, _) \circ (\mathcal{A}_{\mathcal{M}}[\![E]\!] \times \mathcal{A}_{\mathcal{M}}[\![E']\!]), \\ \mathcal{A}_{\mathcal{M}}[\![E \,\&\, E']\!] &= (_ \,\&\, _) \circ (\mathcal{A}_{\mathcal{M}}[\![E]\!] \times \mathcal{A}_{\mathcal{M}}[\![E']\!]), \\ \mathcal{A}_{\mathcal{M}}[\![\mathrm{rec}\, x.\, E]\!] &= \Upsilon_{\mathcal{A}_{\mathcal{M}}} \circ \mathit{curry}(\mathcal{A}_{\mathcal{M}}[\![E]\!] \circ \Phi_x), \end{aligned} \tag{8.2}$$

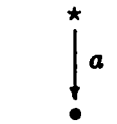
Let us finish this Chapter with several examples of the meaning given via $\mathcal{A}_{\mathcal{M}}[\![_]\!]$ to several processes. As a consequence of Lemma 6.3.13 it follows that for a process E the meaning $\mathcal{A}_{\mathcal{M}}[\![E]\!]$ is a constant functor $\mathcal{A}_{\mathcal{M}}[\![E]\!]$:$\mathbf{Env}_{\mathcal{A}_{\mathcal{M}}} \to_{\omega} \mathcal{A}_{\mathcal{M}}$. Thus, in each example an appropriate lss that determines the functor is presented.

All examples are taken from the sublanguage CCS of $\mathcal{L}$.

Example 8.2.14 Let $E = \Omega$. Then the $\mathcal{A}_{\mathcal{M}}[\![\Omega]\!]$ is determined by $\mathbf{1}_{\mathcal{A}_{\mathcal{M}}}$ — the initial object of $\mathcal{A}_{\mathcal{M}}$. ◇

Example 8.2.15 Next, let us take $E = m\cdot\Omega$. The corresponding lss is given by:

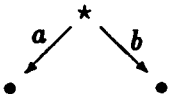
$\mathcal{A}_{\mathcal{M}}[\![m\cdot\Omega]\!]$



with $l(a) = m$. ◇

Example 8.2.16 Let us take $E = m\cdot\Omega +_{CCS} m\cdot\Omega$. The corresponding lss is given by:

$\mathcal{A}_M[\![m\cdot\Omega +_{CCS} m\cdot\Omega]\!]$

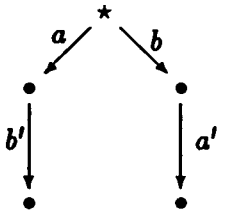


with $l(a) = m = l(b)$. ◇

The next pair of examples shows the difference in the meaning of processes identified by interleaving semantics.

Example 8.2.17 Let $E = m\cdot m'\cdot\Omega +_{CCS} m'\cdot m\cdot\Omega$. The corresponding lss is given by:

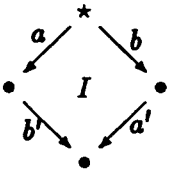
$\mathcal{A}_M[\![m\cdot m'\cdot\Omega +_{CCS} m'\cdot m\cdot\Omega]\!]$



with $l(a) = m = l(a')$, and $l(b) = m' = l(b')$. ◇

Example 8.2.18 Let $E = m\cdot\Omega|_{CCS}m'\cdot\Omega$. The corresponding lss is given by:

$\mathcal{A}_M[\![m\cdot\Omega|_{CCS}m'\cdot\Omega]\!]$



with $l(a) = m = l(a')$, and $l(b) = m' = l(b')$. ◇

Chapter 9

From Operational to Denotational Semantics

So far our language $\mathcal{L}$ was given two kinds of descriptions. Axiomatic operational semantics defined in Chapter 7 formalised the notion of one-step evolution of processes, i.e., the closed terms of $\mathcal{L}$. In Chapter 8 we gave $\mathcal{L}$ denotational semantics in $\mathcal{A}_{\mathcal{M}}$ — category of c^*-generated forward stable labelled skeletal systems over alphabet $\mathcal{M}$. What remains to be done is to relate the two descriptions of $\mathcal{L}$.

The minimal requirement would be to give an indication of an agreement between the two. For instance there is a parallel between the arrows of a lss-s and derivation trees in $\mathcal{L}$. Given a lss S, $c, d \in C_S$, $a \in A_S$ and $L \subseteq \mathcal{M}$ one can show that

$$c \xrightarrow{a} d \text{ in } S \setminus L \quad \text{iff} \quad c \xrightarrow{a} d \text{ in } S \quad \text{and} \quad \neg l_S(a) \in L.$$

This strongly reminds us of the introduction rule for the restriction operator:

$$\frac{E \xrightarrow{\mu} F}{E \setminus L \xrightarrow{\mu} F \setminus L}(\text{res-}L) \qquad \text{for } \mu \notin L \subseteq M.$$

Indeed, this kind of results can be found e.g., in [Winskel 84a]. Although so intuitively appealing, it is not quite satisfactory method of relating the two semantics. It seems difficult to make the argument more formal. What is more worrying, however, in the axiomatic operational description there is no counterpart to the information about concurrency that plays so eminent part in any non-interleaving model. Therefore, at least on the face of it, this information is lost in the operational semantics.

One formal method is to come up with an operationally defined equivalence between processes, and compare it with $\approx_{\mathcal{A}_{\mathcal{M}}[\![-]\!]}$ — the congruence relation on terms determined by the denotational semantics. In the area of our interest the classic examples of operationally defined equivalences include ***bisimulation equivalence***, see [Park 81], [Milner 83], and ***testing equivalences***, [De Nicola&Hennessy 84]. These equivalences, and many derived from them, are ***extensional*** in the sense that they intend to capture the image of a process as perceived by an external observer. The (strong) bisimulation, see [Milner 83] is the strongest among them, see [Brookes&Rounds 83].

Unfortunately, these equivalences do not take account of the concurrency in the systems. In particular, to use an example from CCS, they *reduce concurrency to nondeterministic interleaving*, in the sense that for instance processes: $(m\cdot \mathtt{NIL})|_{CCS}(m'\cdot \mathtt{NIL})$ and $(m\cdot m'\cdot \mathtt{NIL}) +_{CCS} (m'\cdot m\cdot \mathtt{NIL})$ are identified. Since, as we have seen at the end of Chapter 8, they are given different meanings via $\mathcal{A}_{\mathcal{M}}[\![_]\!]$ it is obvious that one cannot hope to achieve an agreement.

One of the big omissions in this thesis is the absence of an account of possible generalisations of the above extensional equivalences that would distinguish between the two processes above. There were many proposals so far, to mention only [Boudol&Castellani 87], [Castellani&Hennessy 87], [Degano et al 87]. In the author's opinion, however, none of them has the immediate intuitive appeal of their precursors from the sequential case.

It is another matter that no such generalisation would do in our case if, as seems reasonable, we insist that it should contain the standard bisimulation equivalence when restricted to the sequential sublanguage of CCS, i.e., with those processes only which do not contain the synchronisation primitive. We have seen that the meanings in $\mathcal{A}_{\mathcal{M}}$ of processes: $m\cdot \mathtt{NIL}$ and $m\cdot \mathtt{NIL} +_{CCS} m\cdot \mathtt{NIL}$ are different, and yet the two processes bisimulate one another.

Therefore, this method of establishing an equivalence between the semantics seems excluded.

Unlike the extensionally motivated equivalences the semantics offered in Chapter 8 gives an *intensional* view of a process. It might be helpful to regard Petri nets, skeletal systems, etc., as very concrete, physical devices. From this perspective one can argue that in Chapter 8 we have provided $\mathcal{L}$ with an alternative operational semantics. It was only the method used that was denotational.

Finally, one can try to *elevate* the axiomatic operational definition up to the level of the denotational. For any process E it would amount to

- a construction of a c^*-generated forward stable lss $S(E)$ from the syntax and operational semantics,
- relating $S(E)$ to $\mathcal{A}_{\mathcal{M}}[\![E]\!]$.

The present Chapter aims at achieving the first goal. We present a construction based on three principles. They are deemed both natural and fundamental.

Principles of the syntactic construction

(i) Closed terms are configurations of the system.

(ii) Derivation trees, i.e., proofs in $\mathcal{L}$, are arrows between the configurations.

(iii) Labels of the conclusion of proofs are the labels of the arrows, with ¿ interpreted as undefined.

The principles give a prescription for a construction of a labelled graph. Thus, apparently, all that is left is to give an account of *concurrency* in the system by specifying *sameness* and *independency* between proofs. The next step will be to relate this construction to the denotational. This is the task of the Chapter 10.

This kind of programme of formalisation of operational semantics is not new. In effect we are pursuing some of the ideas mentioned first in [Plotkin 81], and later in [Abramsky 83].

First, let us show that the problem is nontrivial.

9.1 *CCS* – the Problem

We take Milner's *CCS* as a specific example. This language was provided with non-interleaving semantics many times, to mention only (labelled) safe Petri nets semantics of [Goltz&Mycroft 84], [Winskel 84a], and (labelled) prime event structures semantics in [Winskel 82]. The above attempts are denotational. The first papers contains several mistakes which, in the author's opinion, can be quickly corrected. In what follows we stick to the ideas contained in the paper rather than to the presentation.

Recently, there were attempts to give *CCS*-like languages a Petri net semantics constructed from syntax and axiomatic operational semantics, [Olderog 87], [Degano et al 87a]. They are based on principles similar to ours, but sufficiently different for their constructions to go through. As we shall see later the construction strictly following our principles cannot succeed for *CCS*. It is, however, difficult to conduct a formal comparison for they start with a new axiomatic operational semantics for *CCS* that uses the notion of a *grape* intended to capture the idea of a *sequential subprocess* of a process. The author is not ready to comment on them more extensively yet.

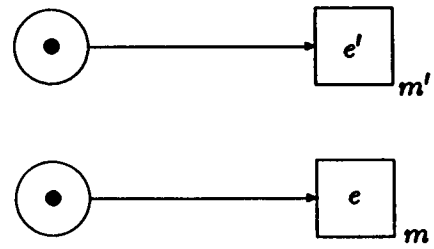
By denotational semantics of *CCS* we mean the one derived from the semantics of $\mathcal{L}$ presented in Chapter 8 be interpreting *CCS* as the sublanguage of CCS, as in Chapter 7. In the following discussion we will identify the denotational meaning of a process, i.e., a closed *CCS*-term, which, by Lemma 6.3.13, is a constant functor with the object that determines it.

We want to extract the principal properties of the denotational construction with respect to concurrency. Next, we intend to demonstrate that it is impossible to carry out our plans following the principles, and obtain a labelled skeletal system. Hence, there are no chances to achieve an equivalence between the denotational semantics of *CCS*, and its axiomatic operational semantics.

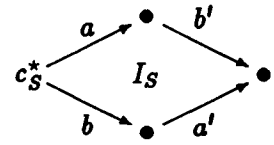
This can be seen as a failure of the original axiomatic operational semantics to incorporate, even in the hidden form, an information of concurrency involved in the evolution of processes. It was this failure that prompted the author to start looking for a 'better' language, one for which our ideas can be carried out, and which at the same time subsumes *CCS*.

The foremost effect of denotational semantics is that it gives a structural view of concur-

rency within the system. In *CCS* it is the _ | _ operator that is the source of concurrency. Assume $m \neq \overline{m'}$, and let us take the term $m\cdot \mathtt{NIL} \,|\, m'\cdot \mathtt{NIL}$. According to Goltz and Mycroft this would result with a labelled net:



with events labelled as suggested. After translation this safe net gives the following lss S:



with $a\ I_S\ b$, $a \asymp_S a'$, $b \asymp_S b'$, $l_S a = m$ and $l_S b = m'$.

We have seen at the end of Chapter 8 that the above lss is (isomorphic to) what we get from the denotational semantics of $\mathcal{L}$ for the derived *CCS* term $m\cdot \Omega|_{CCS} m'\cdot \Omega$.

Now, for any other term E in *CCS*, with $m\cdot \mathtt{NIL} \,|\, m'\cdot \mathtt{NIL}$ as a subterm, if the edges corresponding to a and b exist and belong to the translation of E then they are independent there.

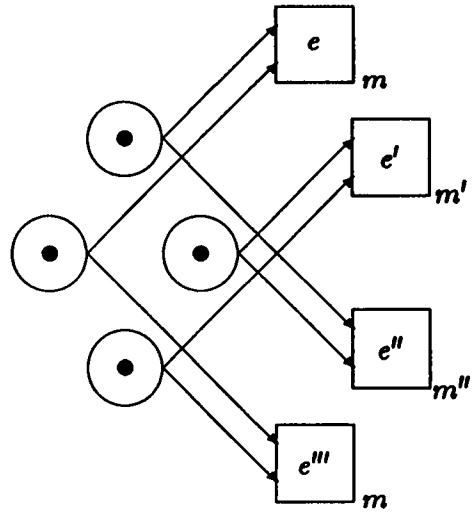
On the other hand the denotation of the sum of two *CCS* processes does not introduce concurrency between the edges of the denotations of these processes. It does, however, retain whatever concurrency existed previously. Perhaps the best idea is to consider another example.

Let $m, m', m'' \in M_{CCS}$ be s.t., $\overline{m'} \neq m \neq \overline{m''}$, and $m' \neq m''$.

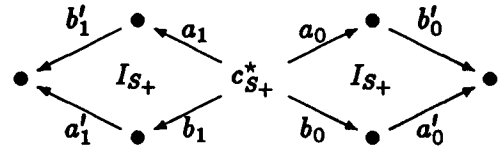
Let E_+ be the following *CCS*-term:

$$E_+ = (m\cdot \mathtt{NIL} \,|\, m'\cdot \mathtt{NIL}) + (m\cdot \mathtt{NIL} \,|\, m''\cdot \mathtt{NIL}).$$

From the nets corresponding to terms $m\cdot \mathtt{NIL} \,|\, m'\cdot \mathtt{NIL}$ and $m\cdot \mathtt{NIL} \,|\, m''\cdot \mathtt{NIL}$ we would obtain the following net as the denotation of E_+:



which in turn gives the following lss S_+:



with $a_i\ I_+\ b_i$, $a_i \asymp_+ a_i'$, $b_i \asymp_+ b_i'$, for $i = 0, 1$, and $l_+a_0 = l_+a_1 = m$, $l_+b_0 = m'$ and $l_+b_1 = m''$.

Again, the lss S_+ is (isomorphic to) the output from the denotational semantics for the corresponding derived term.

Let us see what we get from the syntactic construction.

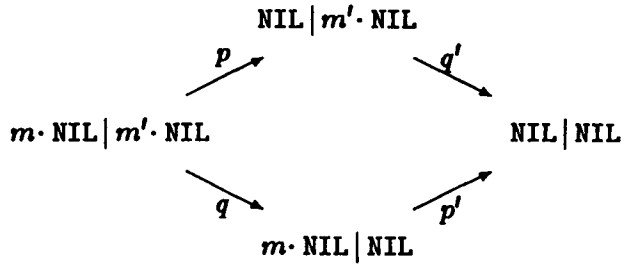
As a result of the three principles every *CCS*-term automatically determines a labelled transition system. There is a huge graph, more formally described later, with all terms of *CCS* as nodes, and all the derivation trees as edges appropriately labelled. Now, all we need is to take for each term E a c^*-generated subgraph of this big graph with E declared as the initial node. Just as we did it for labelled skeletal systems. Here is what we get for particular *CCS*-terms.

First, the term NIL admits *no* derivation rules, so we obtain the initial lss $\langle\{\mathtt{NIL}\}, \mathtt{NIL}, \emptyset, \emptyset, \ldots\rangle$.

Second, the terms of the form $m\cdot E$ should determine a system like the one determined by E, except for the new initial state $m\cdot E$, and a new edge corresponding to the proof $p \vdash m\cdot E \xrightarrow{m} E$. In particular for $m\cdot$ NIL we obtain the following system:

$$m\cdot \mathtt{NIL} \xrightarrow{\ p\ } \mathtt{NIL} \qquad \text{with } p = \frac{}{m\cdot \mathtt{NIL} \xrightarrow{m} \mathtt{NIL}}$$

Next, let us consider the term $m\cdot$ NIL$|\, m'\cdot$ NIL. From our principles we obtain



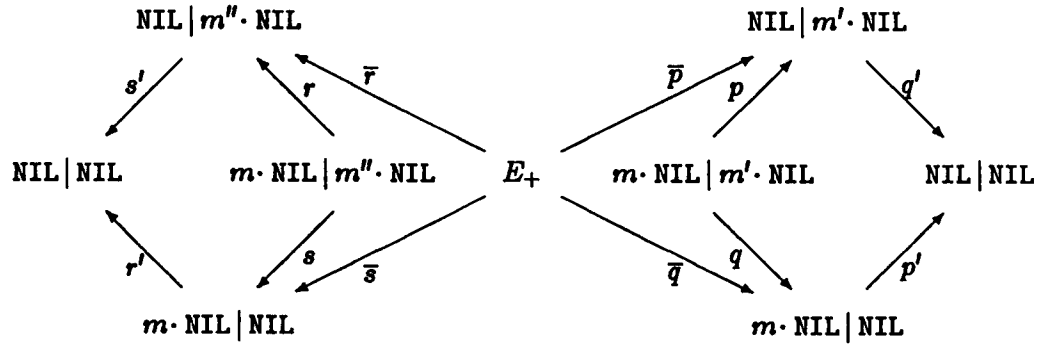
where

$$p = \frac{\overline{m\cdot \mathtt{NIL} \xrightarrow{m} \mathtt{NIL}}}{m\cdot \mathtt{NIL}\,|\,m'\cdot \mathtt{NIL} \xrightarrow{m} m\cdot \mathtt{NIL}\,|\,m'\cdot \mathtt{NIL}} \qquad p' = \frac{\overline{m\cdot \mathtt{NIL} \xrightarrow{m} \mathtt{NIL}}}{m\cdot \mathtt{NIL}\,|\,\mathtt{NIL} \xrightarrow{m} \mathtt{NIL}\,|\,\mathtt{NIL}}$$

$$q = \frac{\overline{m'\cdot \mathtt{NIL} \xrightarrow{m'} \mathtt{NIL}}}{m\cdot \mathtt{NIL}\,|\,m'\cdot \mathtt{NIL} \xrightarrow{m'} m\cdot \mathtt{NIL}\,|\,\mathtt{NIL}} \qquad q' = \frac{\overline{m'\cdot \mathtt{NIL} \xrightarrow{m'} \mathtt{NIL}}}{\mathtt{NIL}\,|\,m'\cdot \mathtt{NIL} \xrightarrow{m'} \mathtt{NIL}\,|\,\mathtt{NIL}}$$

All the time we want to obtain an agreement with denotational semantics, so we are forced to make p independent of q, and relate p with p', and q with q' via sameness relation.

Finally, let us consider the term $E_+ = (m\cdot \mathtt{NIL}\,|\,m'\cdot \mathtt{NIL})+(m\cdot \mathtt{NIL}\,|\,m''\cdot \mathtt{NIL})$ again. From the principles of the syntactic construction we obtain the following graph:



with appropriate proofs $p, p', \overline{p}$, etc.

Once again we aim at achieving an agreement between the system S_+ given denotationally as a meaning of the term E_+, and a construction of a lss from syntax and axiomatic operational semantics. From our previous discussion it follows that we should take $\overline{p} \asymp_+ p' \asymp_+ p$, $\overline{q} \asymp_+ q' \asymp_+ q$ and $\overline{p}\ I_+\ \overline{q}$. Similarly, $\overline{r}\ I_+\ \overline{s}$, $\overline{r} \asymp_+ r'$ and $\overline{s} \asymp_+ s'$. But we see that $p' = r'$, so that $\overline{p} \asymp_+ \overline{r}$, by transitivity and symmetry of $\asymp_+$. Moreover, we have $E_+ \xrightarrow{\overline{p}} \mathtt{NIL}\,|\,m'\cdot \mathtt{NIL}$ and $E_+ \xrightarrow{\overline{r}} \mathtt{NIL}\,|\,m''\cdot \mathtt{NIL}$. Thus, by Definition 8.1.1(i) it follows that $\mathtt{NIL}\,|\,m'\cdot \mathtt{NIL} = \mathtt{NIL}\,|\,m''\cdot \mathtt{NIL}$, so $m' = m''$, contrary to our assumption.

As we see it is impossible to follow the three principles of the syntactic construction, and provide each *CCS*-term with a labelled skeletal system, and obtain a lss isomorphic to the lss provided by the denotational semantics.

The same applies to other non-interleaving models known from the literature. We could simply apply an appropriate translation to reduce the problem to labelled skeletal systems again.

9.2 $\mathcal{L}$ – the Construction

We have seen already that it is impossible to build a labelled skeletal system corresponding to Milner's *CCS*, with construction based on the very natural principles. Here, we show that it is possible to carry out this kind of construction for our language $\mathcal{L}$.

The three principles determine a labelled directed graph $\mathcal{G}$ for $\mathcal{L}$. The natural way in which the items building $\mathcal{G}$ are chosen is worth stressing.

$\mathcal{G} = \langle \mathsf{Rec}_{\mathcal{L}}(\emptyset), \mathsf{Proof}_{\mathcal{L}}, \partial^0, \partial^1, \mathcal{M}, \ell \rangle$ where

$\mathsf{Rec}_{\mathcal{L}}(\emptyset)$ is the set of closed terms, i.e., processes of $\mathcal{L}$;

$\mathsf{Proof}_{\mathcal{L}}$ is the set of derivation trees in $\mathcal{L}$;

$\partial^0(p \vdash E \xrightarrow{\mu} F) = E$;

$\partial^1(p \vdash E \xrightarrow{\mu} F) = F$;

$\ell(p \vdash E \xrightarrow{\mu} F) = \mu$.

In effect we lack here just independency and sameness relations on proofs to have a labelled skeletal system (without the initial point) for $\mathcal{L}$. Now, we intend to concentrate our attention on defining them.

9.2.1 The Independency

Let us start with the independency relation. We intend to use the structure of the set $\mathcal{M}$ of action symbols of $\mathcal{L}$ to give a structural account of *concurrency* between the proofs.

First, the structure of action symbols is used to give a simplified description of the structure of the derivation trees in $\mathcal{L}$. This is done by means of the function str: $\mathsf{Proof}_{\mathcal{L}} \to \mathcal{M}$.

Definition 9.2.1 Let $p \vdash E \xrightarrow{\mu} F$ and $p' \vdash E' \xrightarrow{\mu'} F'$ are derivation trees in $\mathcal{L}$. Function str: $\mathsf{Proof}_{\mathcal{L}} \to \mathcal{M}$ is defined by structural induction:

(i) $str\left(\dfrac{}{m \cdot E \xrightarrow{m} E}(\mathbf{act}-m)\right) = m$ for $m \in M$;

(ii) $str\left(\dfrac{p \vdash E \xrightarrow{\mu} F}{E \setminus L \xrightarrow{\mu} F \setminus L}(\mathbf{res}-L)\right) = str(p)$;

(iii) $str\left(\dfrac{p \vdash E \xrightarrow{\mu} F}{E[R] \xrightarrow{R\mu} F[R]}(\mathbf{ren}-R)\right) = \begin{cases} \mathtt{inl}(str(p)) & R = \mathtt{inl} \\ \mathtt{inr}(str(p)) & R = \mathtt{inr} \\ str(p) & \text{else} \end{cases}$;

(iv) $str\left(\dfrac{p \vdash E \xrightarrow{\mu} F}{E \# E' \xrightarrow{\mathtt{inl}\mu} F[\mathtt{inl}]}(\mathbf{cop}-\mathbf{l})\right) = \mathtt{inl}(str(p))$ and

$str\left(\dfrac{p \vdash E \xrightarrow{\mu} F}{E' \# E \xrightarrow{\mathtt{inr}\mu} F[\mathtt{inr}]}(\mathbf{cop}-\mathbf{r})\right) = \mathtt{inr}(str(p))$;

(v) $str\left(\dfrac{p \vdash E \xrightarrow{\mu} F}{E \,\&\, E' \xrightarrow{\langle \mu, \star \rangle} F \,\&\, E'}(\mathbf{prod}-\mathbf{l})\right) = \langle str(p), \star \rangle$ and

$$str\left(\frac{p \vdash E \xrightarrow{\mu} F}{E' \,\&\, E \xrightarrow{\langle \star, \mu \rangle} E' \,\&\, F}(\mathrm{prod-r})\right) = \langle \star, str(p) \rangle;$$

(vi) $$str\left(\frac{p \vdash E \xrightarrow{\mu} F \qquad p' \vdash E' \xrightarrow{\nu} F'}{E \,\&\, E' \xrightarrow{\langle \mu, \nu \rangle} E' \,\&\, F'}(\mathrm{prod})\right) = \langle str(p), str(p') \rangle;$$

(vii) $$str\left(\frac{p \vdash E[x \mapsto \mathtt{rec}\, x.\, E] \xrightarrow{\mu} F}{\mathtt{rec}\, x.\, E \xrightarrow{\mu} F}(\mathrm{rec})\right) = str(p).$$

The renaming by `inl` and `inr` is strongly related to the coproduct operator by the rules (cop-l) and (cop-r). This is reflected in the above definition.

Our first result shows that the map *str* gives indeed a very faithful account of what is important in the structure of the proofs. It says that any proof p from a term E, i.e., $p \vdash E \rightarrow$, is uniquely determined by its simplified structure $str(p)$.

Lemma 9.2.2 (Extensionality) *If* $p \vdash E \xrightarrow{\mu} F$ *and* $p' \vdash E \xrightarrow{\nu} F'$ *in* $\mathcal{L}$, *and* $str(p) = str(p')$ *then* $p = p'$. *In particular* $F = F'$ *and* $\mu = \nu$.

Proof. By induction on the structure of the proofs p and p'. We consider cases corresponding to the structure of E.

$\boxed{E = m \cdot E'}$ Then both p and p' must be an instance of the rule (act-m), and $F = E'$. This is the base inductive case.

$\boxed{E = \Omega}$ Voidly satisfied.

$\boxed{E = E' \setminus L}$ It must be the case that for some $q, q' \in \mathrm{Proof}_{\mathcal{L}}$ we have

$$p = \frac{q \vdash E' \xrightarrow{\mu} F''}{E' \setminus L \xrightarrow{\mu} F'' \setminus L}, \text{ and } p' = \frac{q \vdash {}'E' \xrightarrow{\nu} F'''}{E' \setminus L \xrightarrow{\nu} F''' \setminus L},$$

where $\mu, \nu \notin L$, $F = F'' \setminus L$ and $F' = F''' \setminus L$. By induction we have $q = q'$, which is enough to conclude that $p = p'$ as well.

$\boxed{E = E'[R]}$ or $\boxed{E = \mathtt{rec}\, x.\, E'}$ The proof in these cases is similar to the above.

$\boxed{E = E' \# E''}$ From $str(p) = str(p')$ it follows that both p and p' have the last inference rule of the same type, i.e., (cop-l) or (cop-r). This enables us to make a call to the inductive hypothesis.

$\boxed{E = E' \,\&\, E''}$ Similar to above. The equality $str(p) = str(p')$ ensures that the last inference rules of p and p' are the same. We can then appeal to induction once, if the rule was (prod-l) or (prod-l), or twice, in case of (prod). Easy details are left to the reader. □

Let us ask again when two derivation trees are to be considered as independent. The following seems to express the natural intuitions:

- ***two proofs are independent iff they correspond to changes in distinct parts of a process, changes which are mutually residual.***

This intuition we intend to formalise. Note that it relates only the derivation trees with conclusions sharing the first process. In reality we need a broader relation. This is done as follows. First, an independency relation on $\mathcal{M}$ is defined. Then, it is imported on $\mathsf{Proof}_{\mathcal{L}}$ via the map *str*.

Definition 9.2.3 (Independency) Let $\mu, \mu', \nu, \nu' \in \mathcal{M}$. Define $\Im \subseteq \mathcal{M} \times \mathcal{M}$ to be the least relation such that

(i) $\langle \mu, \star \rangle \Im \langle \star, \nu \rangle$ and $\langle \star, \nu \rangle \Im \langle \mu, \star \rangle$;

(ii) $\mu \Im \nu$ implies $\mathtt{inl}\,\mu \Im \mathtt{inl}\,\nu$, $\mathtt{inr}\,\mu \Im \mathtt{inr}\,\nu$, $\langle \mu, \star \rangle \Im \langle \nu, \star \rangle$ and $\langle \star, \mu \rangle \Im \langle \star, \nu \rangle$;

(iii) $\mu \Im \nu$ and $\mu' \Im \nu'$ implies $\langle \mu, \mu' \rangle \Im \langle \nu, \nu' \rangle$;

(iv) $\mu \Im \nu$ implies $\langle \mu, \mu' \rangle \Im \langle \nu, \star \rangle$, $\langle \mu, \star \rangle \Im \langle \nu, \nu' \rangle$, $\langle \mu, \mu' \rangle \Im \langle \star, \nu' \rangle$ and $\langle \star, \mu' \rangle \Im \langle \nu, \nu' \rangle$.

We use the same notation to denote a binary relation on $\mathsf{Proof}_{\mathcal{L}}$ defined by:

$$p \Im q \quad \text{iff} \quad \mathit{str}(p) \Im \mathit{str}(q).$$

Our first task is to check that we obtain an independency relation in this way. This is established in the next Proposition. We note that if $\mu \Im \nu$ then this is exactly one among the conditions 9.2.3(i)–(iv) from which it follows.

Proposition 9.2.4 *(i) The relation $\Im \subseteq \mathcal{M} \times \mathcal{M}$ is symmetric and irreflexive.*

(ii) The relation $\Im \subseteq \mathsf{Proof}_{\mathcal{L}} \times \mathsf{Proof}_{\mathcal{L}}$ is symmetric and irreflexive.

Proof. (i) The proof is by induction on the length of the reason behind $\mu'' \Im \nu''$.

If $\mu'' \Im \nu''$ follows from Definition 9.2.3(i), base of induction, then it is obvious.

If it follows from Definition 9.2.3(iii) then $\mu'' = \langle \mu, \mu' \rangle$, $\nu'' = \langle \nu, \nu' \rangle$ with $\mu \Im \nu$ and $\mu' \Im \nu'$. But then $\nu \Im \mu$, $\mu \neq \nu$, and $\nu' \Im \mu'$, $\mu' \neq \nu'$, by induction hypothesis. Therefore $\langle \nu, \nu' \rangle \Im \langle \mu, \mu' \rangle$ from Definition 9.2.3(iii), and $\langle \nu, \nu' \rangle \neq \langle \mu, \mu' \rangle$.

The remaining two cases are left as exercises.

(ii) Follows immediately from (i). □

9.2.2 The Sameness

We are left with the task of defining sameness. The map *str* captured the essential properties of the structure of the derivation trees quite successfully. Thus, the first attempt should be to consider the equivalence relation $\smile$ on $\mathsf{Proof}_{\mathcal{L}}$ given by:

$$p \smile q \quad \text{iff} \quad \mathit{str}(p) = \mathit{str}(q).$$

Unfortunately, it is a failure, and for couple of reasons.

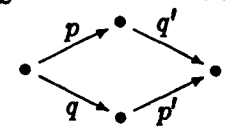
The simplest one is that in general it is not the case that $p \smile q$ implies $\ell(p) = \ell(q)$. Therefore $\mathcal{G}$ equipped with $\smile$ as sameness, and any independency relation fails to satisfy the condition (iv) of Definition 8.1.1.

To see this consider $R\colon \mathcal{M} \rightharpoonup \mathcal{M}$ such that $Rm\uparrow$. Then

$$str\left(\frac{}{m\cdot E \xrightarrow{m} E}\right) = m = str\left(\frac{\overline{m\cdot E \xrightarrow{m} E}}{(m\cdot E)[R] \xrightarrow{i} E[R]}\right).$$

Nevertheless, we are close.

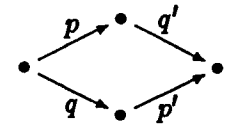
Definition 9.2.5 (Pre-sameness) Let $sob_1(\smile) \subseteq \mathsf{Proof}_{\mathcal{L}} \times \mathsf{Proof}_{\mathcal{L}}$ be defined as follows:

$p\, sob_1(\smile)\, p'$ iff $p \smile p' \wedge \exists q, q' \in \mathsf{Proof}_{\mathcal{L}} . q \smile q' \wedge p \Im q' \wedge$ 

Relation $sob(\smile) = (\, sob_1(\smile) \cup sob_1(\smile)^{-1})^*$ is called the *pre-sameness* relation on proofs, i.e., we take $sob(\smile)$ to be the equivalence relation generated by $sob_1(\smile)$.

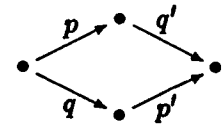
Obviously, $sob(\smile)$ is an equivalence. From the following Lemma it will follow that $\mathcal{G}$ augmented with $\Im$ as independency, and $sob(\smile)$ as sameness is a lss without an initial configuration. Nevertheless, we do not end the quest for sameness here. Our goal is to show that the operational and denotational constructions coincide. For this purpose the pre-sameness relation identifies too little.

Lemma 9.2.6 *(i)* $p \Im q'$ *and* $\partial^1(p) = \partial^0(q')$

implies $\exists q, p' \in \mathsf{Proof}_{\mathcal{L}} . p \smile p' \wedge q \smile q' \wedge$ 

$\wedge\ \ell(p) = \ell(p')$.

(ii) $p \Im q$ *and* $\partial^0(p) = \partial^0(q)$

implies $\exists p', q' \in \mathsf{Proof}_{\mathcal{L}} . p \smile p' \wedge q \smile q' \wedge$ 

.

(iii) $p\, sob_1(\smile)\, p'$ *implies* $\ell(p) = \ell(p')$.

Proof. (i) Let us assume $p \Im q'$ and $\partial^1(p) = \partial^0(q')$. The proof follows by induction on the structure of the proof p. Usually, but not always, we reduce the complexity of both p and q'. We consider in turn the cases of $\partial^0(p)$.

$\boxed{\partial^0(p) = E \,\&\, E'}$ This is the basic case. There are four subcases to consider – depending on the reason behind $p \Im q'$.

If $p \Im q'$ follows from Definition 9.2.3(i) then we can assume w.l.o.g., that

$$p = \frac{p_0 \vdash E \xrightarrow{\mu} F}{E \,\&\, E' \xrightarrow{\langle \mu, * \rangle} F \,\&\, E'} \quad \text{and} \quad q' = \frac{q'_0 \vdash E' \xrightarrow{\nu} F'}{F \,\&\, E' \xrightarrow{\langle *, \nu \rangle} F \,\&\, F'},$$

since $\partial^1(p) = \partial^0(q')$.

Hence, we can put

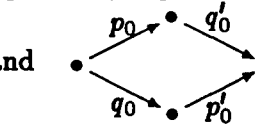
$$q = \frac{q'_0}{E \,\&\, E' \xrightarrow{\langle *, \nu \rangle} E \,\&\, F'} \quad \text{and} \quad p' = \frac{p_0}{E \,\&\, F' \xrightarrow{\langle \mu, * \rangle} F \,\&\, F'}.$$

Evidently, it is the case that $str(p) = str(p')$ and $str(q) = str(q')$, and $\ell(p) = \ell(p')$, as required.

If $p \Im q'$ follows from Definition 9.2.3(iv) then we can assume w.l.o.g., that

$$p = \frac{p_0 \vdash E \xrightarrow{\mu} F \quad p_1 \vdash E' \xrightarrow{\mu'} F'}{E \,\&\, E' \xrightarrow{\langle \mu, \mu' \rangle} F \,\&\, F'} \quad \text{and} \quad q' = \frac{q'_0 \vdash F \xrightarrow{\nu} F''}{F \,\&\, F' \xrightarrow{\langle \nu, \star \rangle} F'' \,\&\, F'},$$

where $p_0 \mathrel{\Im} q'_0$, since $\partial^1(p) = \partial^0(q')$.

By induction hypothesis, as p_0 is simpler than p, there are some $q_0, p'_0 \in \mathsf{Proof}_{\mathcal{L}}$ s.t., $p_0 \smile p'_0$, $q_0 \smile q'_0$ and 

. Let $q_0 \vdash E \xrightarrow{\nu''} E''$ and $p'_0 \vdash E'' \xrightarrow{\mu} F''$. Here $\ell(p'_0) = \ell(p_0)$ by induction hypothesis. Then we can put

$$q = \frac{q_0}{E \,\&\, E' \xrightarrow{\langle \nu'', \star \rangle} E'' \,\&\, E'} \quad \text{and} \quad p' = \frac{p'_0 \qquad p_1}{E'' \,\&\, E' \xrightarrow{\langle \mu, \mu' \rangle} F'' \,\&\, F'}.$$

It is readily seen that we have got the proofs with the desired properties. For we have $str(p') = \langle str(p'_0), str(p_1) \rangle = \langle str(p_0), str(p_1) \rangle = str(p)$. Similarly, $str(q) = str(q')$. It is evident that $\ell(p) = \ell(p')$ holds.

The proofs of the remaining two subcases are simpler, and so left as exercises.

$\boxed{\partial^0(p) = E \# E'}$ We can assume w.l.o.g., that

$$p = \frac{p_0 \vdash E \xrightarrow{\mu} F}{E \# E' \xrightarrow{\mathtt{inl}\,\mu} F[\mathtt{inl}]} \quad \text{and} \quad q' = \frac{q'_0 \vdash F \xrightarrow{\nu} F'}{F[\mathtt{inl}] \xrightarrow{\mathtt{inl}\,\nu} F'[\mathtt{inl}]},$$

since $\partial^1(p) = \partial^0(q')$.

Moreover, from $str(p) = \mathtt{inl}(str(p_0)) \mathrel{\Im} \mathtt{inl}(str(q'_0)) = str(q')$ it follows that $str(p_0) \mathrel{\Im} str(q'_0)$, by Definition 9.2.3(ii). Whence, we can make use of induction to produce $q_0, p'_0 \in \mathsf{Proof}_{\mathcal{L}}$ such that $q_0 \vdash E \xrightarrow{\nu'} E_0$, $p'_0 \vdash E_0 \xrightarrow{\mu} F'$, $str(p_0) = str(p'_0)$ and $str(q_0) = str(q'_0)$.

But then

$$q = \frac{q_0}{E \# E' \xrightarrow{\mathtt{inl}\,\nu'} E_0[\mathtt{inl}]} \quad \text{and} \quad p' = \frac{p'_0}{E_0[\mathtt{inl}] \xrightarrow{\mathtt{inl}\,\mu} F'[\mathtt{inl}]},$$

have the required properties.

The remaining cases are either voidly valid, e.g., for $\partial^0(p) = \Omega$ or $\partial^0(p) = m \cdot E$, or follow easily by induction. We leave all of them, except for recursion, to the reader.

From $\boxed{\partial^0(p) = \mathtt{rec}\, x.\, E}$ and $\partial^1(p) = \partial^0(q')$ we conclude that

$$p = \frac{p_0 \vdash E[x \mapsto \mathtt{rec}\, x.\, E] \xrightarrow{\mu} F}{\mathtt{rec}\, x.\, E \xrightarrow{\mu} F} \quad \text{and} \quad q' \vdash F \xrightarrow{\nu} F'.$$

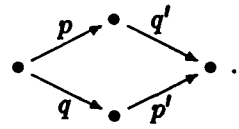
Since $str(p_0) = str(p) \mathrel{\Im} str(q')$ we obtain by induction $q_0, p'_0 \in \mathsf{Proof}_{\mathcal{L}}$, s.t., $q_0 \vdash E[x \mapsto \mathtt{rec}\, x.\, E] \xrightarrow{\nu'} E'$, $p'_0 \vdash E' \xrightarrow{\mu} F'$, $str(p_0) = str(p'_0)$ and $str(q_0) = str(q'_0)$. To finish it is enough to take

$$q = \frac{q_0}{\mathtt{rec}\, x.\, E \xrightarrow{\ell(q_0)} \partial^0(p'_0)} \quad \text{and} \quad p' = p'_0.$$

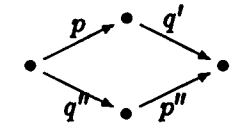
It is this case in which only the complexity of p is being reduced before applying to induction hypothesis.

(ii) Omitted. It is very similar to the proof of (i).

(iii) From $p\, sob_1(\smile)\, p'$ we get some $q, q' \in \mathsf{Proof}_{\mathcal{L}}$ s.t.,

$$p \,\Im\, q' \wedge p \smile p' \wedge q \smile q' \wedge$$ 

We can apply (i) to p and q' to obtain p'' and q'' such that

$$p \smile p'' \wedge q'' \smile q' \wedge \wedge\, \ell(p) = \ell(p'') \wedge$$ 

From Extensionality Lemma, Lemma 9.2.2, it follows now that $q = q''$, since $str(q) = str(q') = str(q'')$. But then Extensionality Lemma applies to p' and p'' as well, so that $p' = p''$. Thus, $\ell(p) = \ell(p') = \ell(p'')$. □

A modification of the relation $sob(\smile)$ will be taken as the sameness relation. To ease the formal aspects of the definition let us introduce first some notational conventions.

Except for the prefixing, and obviously the nullary Ω, all the other constructors give rise to partial (unary and one binary) operators: $_[R]$, $_ \setminus L$, $E \,\#\, _$, $_ \,\#\, E$, $\mathbf{rec}\, x.\, E(_)$, $E \,\&\, _$, $_ \,\&\, F$ and $_ \,\&\, _$, on the set $\mathsf{Proof}_{\mathcal{L}}$ of all derivation trees in $\mathcal{L}$. Each one corresponds to a unique inference rule of $\mathcal{L}$. Thus, we obtain $_[R]$, for each $R \colon \mathcal{M} \rightharpoonup \mathcal{M}$, defined by:

$$p[R] = \frac{p}{E[R] \overset{R\mu}{\rightharpoonup} F[R]}, \quad \text{where } p \vdash E \overset{\mu}{\rightharpoonup} F.$$

The operator $_ \setminus L$ is one of those partially defined, and $p \setminus L\!\uparrow$ iff $\ell(p) \in L$:

$$p \setminus L = \frac{p}{E \setminus L \overset{\mu}{\rightharpoonup} F \setminus L}, \quad \text{where } p \vdash E \overset{\mu}{\rightharpoonup} F, \text{ and } \mu \notin L.$$

The other one corresponds to recursion:

$$\mathbf{rec}\, x.\, E(p) = \frac{p}{\mathbf{rec}\, x.\, E \overset{\mu}{\rightharpoonup} F}, \quad \text{where } p \vdash E[x \mapsto \mathbf{rec}\, x.\, E] \overset{\mu}{\rightharpoonup} F.$$

That is $\mathbf{rec}\, x.\, E(p)$ is defined iff $\partial^0(p) = E[x \mapsto \mathbf{rec}\, x.\, E]$, and then defined as above.

The operator $_ \,\&\, _$ is the only binary among them, and is given by:

$$p \,\&\, q = \frac{p \quad q}{E \,\&\, E' \overset{\langle \mu, \nu \rangle}{\rightharpoonup} F \,\&\, F'}, \quad \text{where } p \vdash E \overset{\mu}{\rightharpoonup} F \text{ and } q \vdash E' \overset{\nu}{\rightharpoonup} F'.$$

The remaining operators are defined as follows.

$$p \,\&\, F = \frac{p}{E \,\&\, E' \overset{\langle \mu, \star \rangle}{\rightharpoonup} F \,\&\, E'}, \quad \text{where } p \vdash E \overset{\mu}{\rightharpoonup} F,$$

$$E \,\&\, q = \frac{q}{E \,\&\, E' \overset{\langle \star, \nu \rangle}{\rightharpoonup} E \,\&\, F'}, \quad \text{where } q \vdash E' \overset{\nu}{\rightharpoonup} F',$$

$$p \# F = \frac{p}{E \# E' \overset{\mathtt{inl}}{\to}{}^{\mu} F[\mathtt{inl}]}, \quad \text{where } p \vdash E \overset{\mu}{\to} F,$$

$$E \# q = \frac{q}{E \# E' \overset{\mathtt{inr}}{\to}{}^{\nu} F'[\mathtt{inr}]}, \quad \text{where } q \vdash E' \overset{\nu}{\to} F',$$

Definition 9.2.7 (Sameness) A relation $\asymp \subseteq \mathsf{Proof}_{\mathcal{L}} \times \mathsf{Proof}_{\mathcal{L}}$, called *sameness*, is the least equivalence relation such that:

(i) $sob_1(\smile) \subseteq \asymp$,

(ii) $\mathtt{rec}\, x.\, E(p) \asymp p$, for $p \in \mathrm{dom}(\mathtt{rec}\, x.\, E(_))$,

(iii) $E \# p \asymp p[\mathtt{inr}]$ and $p \# E \asymp p[\mathtt{inl}]$,

(iv) $p \asymp q$ implies $p \setminus L \asymp q \setminus L$, for $p, q \in \mathrm{dom}(_ \setminus L)$,

(v) $p \asymp q$ implies $F\,p \asymp F\,q$, for $F \in \{_[R], E \# _, _ \# E, _ \,\&\, E, E \,\&\, _\}$,

(vi) $p \asymp q$ and $p' \asymp q'$ implies $p \,\&\, q \asymp p' \,\&\, q'$,

Armed with this equivalence relation we can finish the definition of the syntactic construction. But first, some results.

Fact 9.2.8 $sob(\smile) \subseteq \asymp \subseteq \smile$.

Proof. The first inclusion is evident. The second can be proved by induction on the reason why $p \asymp q$ holds. □

The following properties will be important in the next Chapter.

Proposition 9.2.9 *Let $p, p', q, q' \in \mathsf{Proof}_{\mathcal{L}}$. Then*

(i) $p \asymp q$ *iff* $F\,p \asymp F\,q$, *for* $F \in \{_[R], E \# _, _ \# E, _ \,\&\, E, E \,\&\, _\}$,

(ii) $p \asymp q$ *iff* $p \setminus L \asymp q \setminus L$, *for* $p, q \in \mathrm{dom}(_ \setminus L)$,

(iii) $p \asymp q$ *and* $p' \asymp q'$ *iff* $p \,\&\, q \asymp p' \,\&\, q'$,

(iv) $p \asymp q$ *iff* $E \# p \asymp q[\mathtt{inr}]$, *and* $p \asymp q$ *iff* $p \# E \asymp p[\mathtt{inl}]$.

Proof. Implications from left to right follow easily by definition. In fact, all but (iii) are instances of the definition of $\asymp$.

The implications in the other direction are proved by simultaneous induction on the length of the proof that the premise holds. We consider only one case as an example.

Let us assume that $p \,\&\, q \asymp p' \,\&\, q'$ holds. If it is because $p \,\&\, q\ sob_1(\smile)\ p' \,\&\, q'$ then there are three possibilities. One is that $p\ sob_1(\smile)\ p'$ and $q\ sob_1(\smile)\ q'$. The other two are: $p\ sob_1(\smile)\ p' \wedge q = q'$ and $q\ sob_1(\smile)\ q' \wedge p = p'$. In all three cases we have $p \asymp p'$ and $q \asymp q'$. If $p \,\&\, q \asymp p' \,\&\, q'$ follows for other reasons the result follows easily by induction. □

9.2.3 $\mathcal{S}$ – a LSS for $\mathcal{L}$ from Syntax and Operational Semantics

Now, we are prepared to collect our efforts in the definition of a labelled skeletal system $\mathcal{S}$, built from the syntax and axiomatic operational semantics of $\mathcal{L}$. We will refer to it as a lss despite the fact that it lacks initial configuration.

Let $\mathcal{S} = \langle \mathsf{Rec}_{\mathcal{L}}(\emptyset), \mathsf{Proof}_{\mathcal{L}}, \asymp, \Im, \partial^0, \partial^1, \mathcal{M}, \ell \rangle$.

Proposition 9.2.10 *$\mathcal{S}$ is a forward stable labelled skeletal system.*

Proof. Let us check in turn that $\mathcal{S}$ satisfies all the properties required.

$\mathcal{S}$ satisfies 8.1.1(i) Since $\asymp \subseteq \frown$, this result follows be Extensionality Lemma, Lemma 9.2.2.

$\mathcal{S}$ satisfies 8.1.1(ii) Assume $p \asymp p' \Im q' \asymp q$. Again, since $\asymp \subseteq \frown$ it follows that $p \Im q$, by definition of $\Im$.

$\mathcal{S}$ satisfies 8.1.1(iii) Follows from Lemma 9.2.6(i), since $sob_1(\frown) \subseteq \asymp$.

$\mathcal{S}$ satisfies 8.1.1(iv) Follows by induction on the length of the reason why $p \asymp q$ holds. In one of the basic cases: $p\, sob_1(\frown)\, q$ we apply Lemma 9.2.6(iii).

$\mathcal{S}$ satisfies 8.1.1(v) Follows from Lemma 9.2.6(ii). □

It should be evident that one could take $sob(\frown)$ in place of $\asymp$ in the definition of $\mathcal{S}$, and the result corresponding to the above would go with small simplifications.

The almost-lss $\mathcal{S}$ constructed above can be used to provide every process, i.e., closed term of $\mathcal{L}$ with a fully flagged forward stable labelled skeletal system. For a process E we can take simply:

$$\langle \mathcal{S}, E \rangle = \langle \mathsf{Rec}_{\mathcal{L}}(\emptyset), E, \mathsf{Proof}_{\mathcal{L}}, \asymp, \Im, \partial^0, \partial^1, \mathcal{M}, \ell \rangle.$$

The above does not yield a $\mathcal{A}_{\mathcal{M}}$-object, however. This is because of the junk of unreachable from E states in every $\langle \mathcal{S}, E \rangle$.

This, however, can be easily repaired.

Definition 9.2.11 The *syntactic* forward stable, c^*-generated labelled skeletal systems are defined by:

$$\mathcal{S}_E = c^*\text{–}\mathrm{GEN}\langle \mathcal{S}, E \rangle, \qquad \text{for } E \in \mathsf{Rec}_{\mathcal{L}}(\emptyset).$$

After this adjustment we obtain the following Lemma.

Lemma 9.2.12 *$\mathcal{S}_E \in \mathcal{A}_{\mathcal{M}}$, for $E \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)$.*

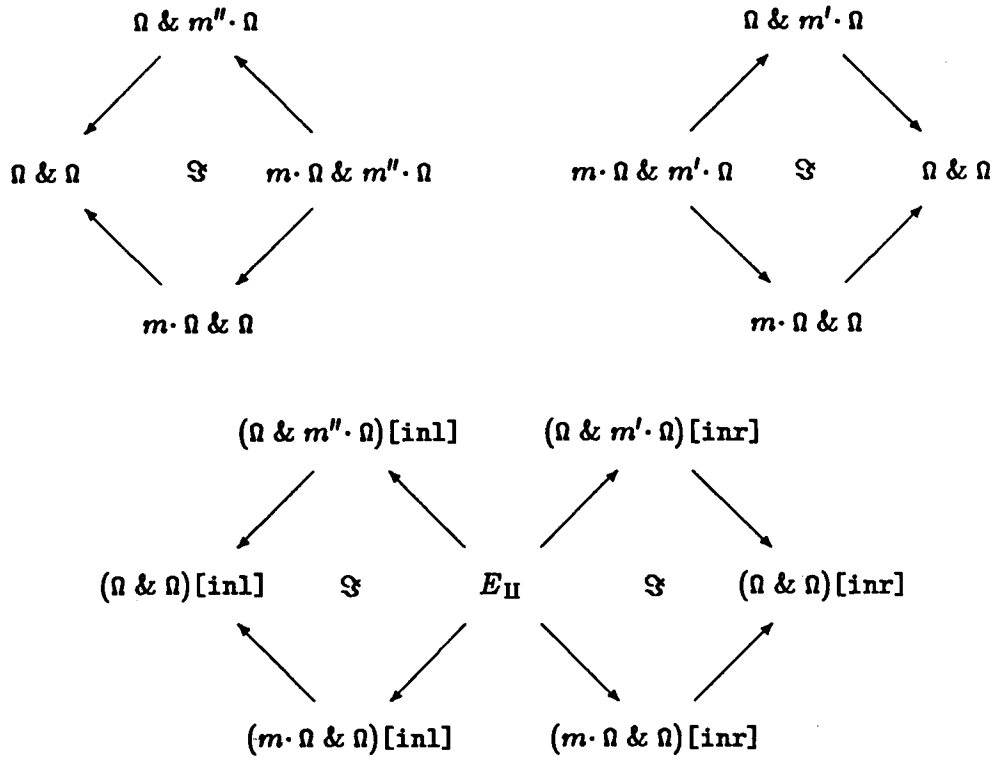
Proof. The lss $\langle \mathcal{S}, E \rangle$ is forward stable over $\mathcal{M}$, by the last Proposition.

Thus, $c^*\text{–}\mathrm{GEN}(\langle \mathcal{S}, E \rangle)$ is c^*-generated forward stable lss over $\mathcal{M}$, by Proposition 8.1.25. That is $\mathcal{S}_E \in \mathcal{A}_{\mathcal{M}}$. □

9.2.4 Counter-example Revisited

Let us reconsider the $\mathcal{L}$'s counterpart of the example that caused problems for *CCS*.

The term E_+ after translation becomes $E_{\text{II}} = (m\cdot \Omega \,\&\, m'\cdot \Omega) \,\#\, (m\cdot \Omega \,\&\, m''\cdot \Omega)$ again. We obtain from the syntactic construction the following graph:



with the names of the proofs omitted.

Now we see that the danger of identification of certain proofs is avoided at the price of marking the 'outcome' of a choice made at the terms of the form $E \,\#\, F$ with a label `inl` or `inr`.

Chapter 10

Equivalence

The preliminary work required to relate the axiomatic operational semantics with the denotational semantics was accomplished in the last Chapter. Now, we are ready to make precise our ideas expressed there. It was suggested to enrich the syntactic construction so that a formal comparison with the denotational semantics becomes possible.

To this end category $\mathcal{A}_{\mathcal{M}}$ is put forward as a specific domain of denotations.

In Chapter 8 it was shown that $\mathcal{A}_{\mathcal{M}}$ is a (categorical) $\Sigma^{\mathcal{L}}$-domain, and therefore we have obtained the interpretation functor $\mathcal{A}_{\mathcal{M}}[\![E]\!]$ for every term E of $\mathcal{L}$. From Lemma 6.3.13 it follows that this functor is constant for processes, i.e., closed terms of $\mathcal{L}$.

Now, let us assume that for every process E of $\mathcal{L}$ a labelled skeletal system $S(E)$ in $\mathcal{A}_{\mathcal{M}}$ is constructed from operational semantics. Ideally, $S(E) = S_E$, as constructed in Chapter 9.

One way to say that the two constructions are equivalent is to prove that

$$\mathcal{A}_{\mathcal{M}}[\![E]\!]\rho = S(E), \quad \text{for } E \in \mathsf{Rec}_{\mathcal{L}}(\emptyset). \tag{10.1}$$

In fact it seems necessary to prove a stronger result.

Every closed substitution ϱ gives rise to an environment $\tilde{\varrho}$ defined via operational semantics by: $\tilde{\varrho}\,x = S(\varrho\,x)$. Now, we are interested in showing that the following result is valid.

$$\mathcal{A}_{\mathcal{M}}[\![E]\!]\tilde{\varrho} = S(E\,\varrho), \quad \text{for } \varrho\colon \mathsf{Var} \to \mathsf{Rec}_{\mathcal{L}}(\emptyset), E \in \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var}). \tag{10.2}$$

That the implication (10.2) $\Rightarrow$ (10.1) holds should be, hopefully, obvious.

Historically, the above was the first kind of equivalence between the operational and denotational semantics considered, see e.g., [Milner 76]. For an imperative language considered by Milner it is natural to take the set of valuations of program variables as the set S of *states*. It is made into a flat cpo by adding an extra bottom element. The set of continuous maps $S \to_\omega S$ is then a natural domain of interpretations for programs. Milner shows that the two functions corresponding to a given program, one defined operationally, the other denotationally, coincide.

The author's strong belief is that $\mathcal{A}_{\mathcal{M}}$ plays for the *CCS*-like languages a role comparable to the role played by domain $S \to_\omega S$ for simple imperative languages.

For the clarity of argument we proceed in stages.

First, we consider the case of *syntactically finite terms of* $\mathcal{L}$, i.e., the attention is restricted to terms in $\mathsf{T}_{\mathcal{L}}$. A number of results is established that enables us to show the equivalence in this case. Luckily, we can take $S(E) = S_E$, for $E \in \mathsf{T}_{\mathcal{L}}$.

Next, we turn attention to a broader sublanguage of $\mathcal{L}$. A sublanguage $g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$ of $\mathcal{L}$-terms with *guarded recursion* is considered. This time certain preparations are necessary for, in general, $S_E \not\simeq \mathcal{A}_{\mathcal{M}}[\![E]\!]\rho$, even for the processes from the sublanguage now considered. As a remedy a syntactic translation $\{\!|_|\!\}\colon \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var}) \to \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$ is defined, and its properties established.

For instance it is an invariant of the denotational semantics: $\mathcal{A}_{\mathcal{M}}[\![E]\!] \cong \mathcal{A}_{\mathcal{M}}[\![\{\!|E|\!\}]\!]$. Moreover, it commutes with substitution, and cuts down to $\{\!|_|\!\}\colon g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var}) \to g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$. Thus, we can take $S(E) = S_{\{\!|E|\!\}}$, for $E \in g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$, and prove the equivalence.

We end the second section with a conjecture that the similar result can be obtained for the full language.

Finally, the problem of *full-abstraction* between the two semantics is posed. We show that it reduces to the problem of expressiveness of $\mathcal{L}$.

10.1 $\mathcal{L}$ without Recursion

Let us start with a simpler task of providing a formal comparison between the two constructions for a very restricted version of $\mathcal{L}$. Here, we consider the sublanguage comprising only *syntactically finite* terms of $\mathcal{L}$, i.e., those which do not have recursive subterms. In the nomenclature of Chapter 6 these are elements of $\mathsf{T}_{\mathcal{L}}$ – the ground terms of $\mathcal{L}$.

10.1.1 Preliminaries

Our task is relatively simple this time. First, a number of technical results is established. Each one deals with a particular constructor σ of $\mathcal{L}$. And each has the form $S_{\sigma(E_1,\ldots,E_n)} = \sigma(S_{E_1},\ldots,S\,E_n)$ where $E_1, \ldots, E_n$ are *processes*, i.e., closed terms of $\mathcal{L}$. The symbol σ on the right stands for the semantic functor interpreting the constructor, in accordance with our convention. We do not require the processes to be ground terms of $\mathcal{L}$. This extra generality appears useful in the next section. These results, after specialisation when necessary, are the foundation on which the proof of equivalence is built.

Let us fix the notation for the components of $S_E = c^*\text{–}\mathrm{GEN}\langle S, E\rangle$:

$$S_E = \langle C_E, E, \mathsf{Proof}_E, \asymp_E, \Im_E, \partial^0_E, \partial^1_E, \mathcal{M}, \ell_E\rangle, \qquad \text{where}$$

C_E – the set of configurations reachable from E in $\langle S, E\rangle$;

$\mathsf{Proof}_E = \{p \in \mathsf{Proof}_{\mathcal{L}} \mid \exists F, F' \in C_E \,.\, p \vdash F \to F'\}$;

$\asymp_E = \asymp \cap\, \mathsf{Proof}_E \times \mathsf{Proof}_E$;

$\Im_E = \Im \cap\, \mathsf{Proof}_E \times \mathsf{Proof}_E$;

$\partial^0_E, \partial^1_E$ – the restrictions of ∂^0 and ∂^1 to $\mathsf{Proof}_{\mathcal{L}}$ and C_E;

ℓ_E – the restriction of ℓ to $\mathsf{Proof}_{\mathcal{L}}$.

For a lss S a relation $\prec_S \subseteq C_S \times C_S$ is defined as a transitive closure of the one-step, or elementary action relation:

$$E \prec_S F \quad \text{iff} \quad \exists x \in \mathsf{Proof}^*_{\mathcal{L}} . |x| \geq 1 \wedge E \xrightarrow{x} F.$$

Next, a measure of complexity of $\mathsf{T}_{\mathcal{L}}$-terms is introduced. Its task is to facilitate the proof that $\prec_S$ is irreflexive when restricted to syntactically finite terms of $\mathcal{L}$. Formally, we define a function $c: \mathsf{T}_{\mathcal{L}} \to \omega$ by structural induction:

$$c(\Omega) = 1;$$

$$c(m \cdot F) = c(F) + 1;$$

$$c(F \setminus L) = c(F[R]) = c(F);$$

$$c(F \# F') = c(F \,\&\, F') = c(F) + c(F').$$

With the help of this measure one proves easily the following result.

Fact 10.1.1 *Let $E, F \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)$. Then*

(i) If $E \in \mathsf{T}_{\mathcal{L}}$ and $\mathcal{L} \vdash E \xrightarrow{\mu} F$ then $F \in \mathsf{T}_{\mathcal{L}}$.

(ii) If $E \in \mathsf{T}_{\mathcal{L}}$ and $\mathcal{L} \vdash E \xrightarrow{\mu} F$ then $c(E) > c(F)$.

(iii) If $E \in \mathsf{T}_{\mathcal{L}}$ and $E \prec_S F$ then $E \neq F$.

Proof. (i) Easy proof by induction on the structure of a derivation tree p with conclusion $\mathcal{L} \vdash E \xrightarrow{\mu} F$.

(ii) From (i) it follows that $c(F)$ is defined. The proof is inductive, as in (i).

(iii) From $E \prec_S F$ it follows that for some $x \in \mathsf{Proof}^*_{\mathcal{L}}$ with $|x| \geq 1$ we have $E \xrightarrow{x} F$. The proof goes now by induction of $|x|$. The result (ii) is used at both stages of the inductive proof. □

It is the last property, 10.1.1(iii), that we are really after. It states that a syntactically finite process cannot change back to itself as a result of a non-trivial evolution. Assumptions to this effect will appear in several technical Lemmas, in this and the following sections. The property ensures that they are automatically satisfied when the attention is restricted to the syntactically finite processes of $\mathcal{L}$.

Operator Ω

Our first task is simple. Recall that the interpretation of the constant Ω is given, denotationally, by the constant functor $\Omega: \mathbf{1} \to \mathcal{A}_{\mathcal{M}}$ determined by the zero object of $\mathcal{A}_{\mathcal{M}}$, i.e., by $\mathbf{1}_{\mathcal{A}_{\mathcal{M}}}$. Thus, we need to establish the following result.

Lemma 10.1.2 $S_\Omega \simeq \mathbf{1}_{\mathcal{A}_{\mathcal{M}}}$.

Proof. Obvious. Just compare the two lss-s:

$$\mathbf{1}_{\mathcal{A}_M} = \langle\{\star\}, \star, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, M, \emptyset\rangle \quad \text{and} \quad S_\Omega = \langle\{\Omega\}, \Omega, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, M, \emptyset\rangle.$$

The first one is the zero object of $\mathcal{A}_M$, by Lemma 8.1.40. On the other hand we see that there are no proofs in Proof_Ω as there are no inference rules introducing Ω. Hence the form of the second. □

Operator $m\cdot\ _$

Let us turn attention to the prefixing operator. This time we have to make certain assumptions to obtain the required isomorphism. The necessity of the assumption is illustrated later in Example 10.2.4.

Lemma 10.1.3 *If $m\cdot E \not\sim_S m\cdot E$ then $S_{mE} \simeq m\cdot S_E$, for $E \in \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$.*

Proof. From the definition of the prefixing functor, Definition 8.2.2, it follows that $m\cdot S_E$ is given by

$$m\cdot S_E = \langle C_E \cup \{\mathsf{new}_C(S_E)\}, \mathsf{new}_C(S_E), \mathsf{Proof}_E \cup \{\mathsf{new}_A(S_E)\}, \asymp_m, \Im_E, \partial^0_m, \partial^1_m, M, \ell_m\rangle$$

where

$$\asymp_m = \asymp_E \cup \{(\mathsf{new}_A(S_E), \mathsf{new}_A(S_E))\};$$

$$\partial^0_m(a) = \begin{cases} \mathsf{new}_C(S) & a = \mathsf{new}_A(S_E) \\ \partial^0_E(a) & \textit{otherwise} \end{cases} \qquad \partial^1_m(a) = \begin{cases} E & a = \mathsf{new}_A(S_E) \\ \partial^1_E(a) & \textit{otherwise} \end{cases};$$

$$\ell_m(a) = \begin{cases} m & a = \mathsf{new}_A(S_E) \\ \ell_E(a) & \textit{otherwise} \end{cases}.$$

We define a pair $\beta = \langle \beta_C, \beta_A\rangle : m\cdot S_E \to S_{mE}$ where:

$$\beta_C(c) = \begin{cases} m\cdot E & c = \mathsf{new}_C(S_E) \\ c & \textit{otherwise} \end{cases}, \quad \text{and}$$

$$\beta_A(a) = \begin{cases} p & p \vdash a = \mathsf{new}_A(S_E) \ \wedge\ m\cdot E \xrightarrow{m} E \\ a & \textit{otherwise} \end{cases}.$$

What remains to show is that β is an isomorphism.

We intend to show first that β_C is a bijection. From this it follows easily that β_A is a bijection as well, which we leave to the reader.

Obviously, β_C is a bijection iff $m\cdot E \notin C_E$. But the latter follows from the assumption that $m\cdot E \not\sim_S m\cdot E$.

We show only that β and $\beta^{-1} = \langle \beta_C^{-1}, \beta_A^{-1}\rangle$ satisfy the condition 8.1.2(iii). The remaining conditions of the Definition 8.1.2 are easier to established, and therefore omitted.

We need to know that for $a, a' \in A_{m S_E}$ the following holds:

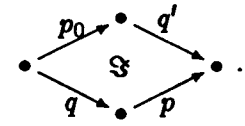
$$a \asymp_m a' \quad \text{iff} \quad \beta_A(a) \asymp_{mE} \beta_A(a').$$

This will follow if we show that $\asymp_{m\cdot E} = \asymp_E \cup \{\langle p_0, p_0\rangle\}$, where $p_0 \vdash m\cdot E \xrightarrow{m} E$.

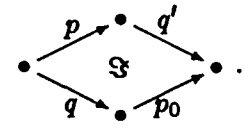
By definition $\asymp_{m\cdot E} = \asymp \cap \mathsf{Proof}_{m\cdot E} \times \mathsf{Proof}_{m\cdot E}$ and $\asymp_E = \asymp \cap \mathsf{Proof}_E$. Moreover, $\mathsf{Proof}_{m\cdot E} = \mathsf{Proof}_E \cup \{p_0\}$ with $p_0 \notin \mathsf{Proof}_E$, since $m\cdot E \not\asymp_S m\cdot E$. Thus, it is enough to show that for every $p \in \mathsf{Proof}_E$ if $p_0 \asymp p$ then $p_0 = p$.

It is immediate that $p \asymp p_0$ may follow only from $p\ sob_1(\smile)\ p_0$, $p_0\ sob_1(\smile)\ p$, or $p = p_0$. Hence, $p = p_0$ follows if we show that neither $p_0\ sob_1(\smile)\ p$ nor $p\ sob_1(\smile)\ p_0$, for any $p \in \mathsf{Proof}_{\mathcal{L}}$.

Assume to the contrary.

If $p_0\ sob_1(\smile)\ p$ then for some $q, q' \in \mathsf{Proof}_{\mathcal{L}}$ we have 

But $q \vdash m\cdot E \rightarrow$ only if $p_0 = q$. Thus, $p_0\ \Im\ q' \smile q = p_0$ implies $p_0\ \Im\ p_0$, by Proposition 9.2.10. We obtain a contradiction with irreflexivity of $\Im$.

If $p\ sob_1(\smile)\ p_0$ then for some $q, q' \in \mathsf{Proof}_{\mathcal{L}}$ we have 

Since $\partial^0(p_0) = m\cdot E$ it follows that $\partial^0(p) = \partial^0(q)$ is either $\mathrm{rec}\, x.\, F$, for some $F \in \mathsf{Rec}_{\mathcal{L}}(\{x\})$, $x \in \mathsf{Var}$, or $m'\cdot F$, for some $F \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)$, $m' \in M$. In the second case we obtain contradiction as before. In the first the result follows by induction. □

The above Lemma can be applied to the ground $\mathcal{L}$ terms easily. Later we shall see that it does not hold in general, compare Example 10.2.4.

Corollary 10.1.4 *$S_{m\cdot E} \simeq m\cdot S_E$, for $E \in \mathsf{T}_{\mathcal{L}}$.*

Proof. By Fact 10.1.1(iii). □

Operator $_[R]$

Lemma 10.1.5 *$S_{E[R]} \simeq S_E[R]$, for $E \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)$.*

Proof. By the definition of the renaming functor we have

$$S_E[R] = \langle C_E, E, \mathsf{Proof}_E, \asymp_E, \Im_E, \partial^0_E, \partial^1_E, M, R \circ \ell_E\rangle.$$

Because of the nature of the inference rule for the renaming operator we have for any $F, F'' \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)$, $R\colon M \to M$, and any $p \in \mathsf{Proof}_{\mathcal{L}}$:

$$p \vdash F[R] \xrightarrow{\mu} F'' \quad \text{implies} \quad \exists p' \vdash F \xrightarrow{\nu} F'.\, p = \frac{p'}{F[R] \xrightarrow{R\nu} F'[R]}. \tag{10.3}$$

Let us define $\xi_C\colon C_E \to C_{E[R]}$, $\xi_A\colon \mathsf{Proof}_E \to \mathsf{Proof}_{E[R]}$ by:

$\xi_C(F) = F[R]$, for $f \in C_E$, and

$\xi_A(p) = p[R]$, for $p \in \mathsf{Proof}_E$.

Evidently, the maps defined above are bijective — the surjectivity of ξ_A follows from (10.3). In fact, they constitute an isomorphism of labelled skeletal systems.

We need to verify only that $\xi = \langle \xi_C, \xi_A \rangle$ and $\xi^{-1} = \langle \xi_C^{-1}, \xi_A^{-1} \rangle$ constitute morphisms: $\xi\colon S_E[R] \to S_{E[R]}$ and $\xi^{-1}\colon S_{E[R]} \to S_E[R]$.

Condition 8.1.2(i) is in both cases satisfied. For ξ it is obvious, while for ξ^{-1} it follows from (10.3).

Condition 8.1.2(ii) holds since from the definition of $\Im$ a more general result follows: $p \Im q \iff p[R] \Im q[R]$, for $p, q \in \mathsf{Proof}_{\mathcal{L}}$.

Condition 8.1.2(iii) follows from 9.2.9(i).

Finally, condition 8.1.2(iv) is satisfied as well, since $l_{S_E[R]} = R \circ \ell_E$.

Details of the proofs of the above properties are left to the reader. □

Operator $_ \setminus L$

Lemma 10.1.6 *$S_{E\setminus L} \simeq S_E \setminus L$, for $E \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)$.*

Proof. We have $S_E \setminus L = c^*\text{–GEN}(S_E \setminus\setminus L)$, and $S_{E\setminus L} = c^*\text{–GEN}\langle S, E \setminus L\rangle$.

We consider a full subsystem S of $\langle S, E \setminus L\rangle$, and show that $S \simeq S_E \setminus\setminus L$. Next, we prove that $S_{E\setminus L} = c^*\text{–GEN}(S)$. The result follows since $S_{E\setminus L} = c^*\text{–GEN}(S) \simeq c^*\text{–GEN}\langle S, E \setminus L\rangle = S_{E\setminus L}$ — functors preserve isomorphisms.

System $S_E \setminus\setminus L$ is, by definition, the full subsystem of S_E generated by C_E and $A_E = \{p \in \mathsf{Proof}_E \mid p \vdash E \xrightarrow{\mu} F \Rightarrow \mu \notin M\}$. Therefore, $p \setminus L$ is defined, for $p \in A_E$. It is not difficult to see that $C_S = \{F \setminus L \mid F \in C_E\}$ and $A_S = \{p \setminus L \mid p \in A_E\}$ generate a full subsystem of $\langle S, E \setminus L\rangle$, denoted S.

Moreover, $C^{[\infty]}_{\langle S, E\setminus L\rangle} = C^{[\infty]}_S \subseteq C_S$, and $A^{[\infty]}_{\langle S, E\setminus L\rangle} = A^{[\infty]}_S \subseteq A_S$. So, $c^*\text{–GEN}(S) = S_{E\setminus L}$ follows by Lemma 8.1.6.

Let $\varsigma_C(F) = F \setminus L$, for $F \in C_E$, and $\varsigma_A(p) = p \setminus L$, for $p \in \mathsf{Proof}_E$. It is evident that $\varsigma_C\colon C_E \to C_S$ and $\varsigma_A\colon \mathsf{Proof}_E \to A_S$ are bijections.

It is left to the reader to verify that $\varsigma\colon S_E \setminus\setminus L \to S$ is an isomorphism. The proof is very similar to the proof that ξ constructed in the proof of Lemma 10.1.5 is an isomorphism. This time there are no problems with showing that ς preserves and reflects independency, and labelling. From 9.2.9(ii) it follows that ς preserves and reflects sameness of the systems involved as well. □

Remark. It is precisely the above lemma that requires $\asymp$ rather than $sob(\smile)$ as sameness in the definition of system S. The reader may verify that with $sob(\smile)$ taken as sameness the systems: $S_{E\setminus L}$ and $S_E \setminus L$, where $E = (a \cdot b \cdot \Omega)\ \&\ (c \cdot \Omega)$ and $L = \{\langle \star, c\rangle, \langle b, c\rangle\}$, are non-isomorphic.

Operator _ # _

We come to the *coproduct* operator of $\mathcal{L}$. According to denotational interpretation it is obtained by relabelling of coproduct taken in **LSS**. Unfortunately, categorical injections there fail to be morphisms in $\mathcal{A}_{\mathcal{M}}$ — they have non-identity functions as relabelling part. For this reason we do not make use of the categorical properties of coproduct explicitly.

The reader should also notice that for the second time we are forced to make additional assumptions. In Example 10.2.5 the reader will find justification for the provisos.

Lemma 10.1.7 *If $E \not\prec_S E$ and $F \not\prec_S F$ then $S_{E\#F} \simeq S_E \# S_F$, for $E, F \in \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$.*

Proof. Let us start by recalling from the definition of the functor _ # _ that the lss $S_{\amalg} = S_E \# S_F$ is obtained from the lss-s S_E and S_F by taking their coproduct in **LSS**, and subsequently relabelling the result. In effect we obtain $S_{\amalg} = \langle C_{\amalg}, c^*_{\amalg}, A_{\amalg}, \asymp_{\amalg}, I_{\amalg}, \partial^0_{\amalg}, \partial^1_{\amalg}, \mathcal{M}, l_{\amalg}\rangle$ where

$$C_{\amalg} = (C_E - \{E\}) + (C_F - \{F\}) + \{\star\};$$
$$c^*_{\amalg} = \star;$$
$$A_{\amalg} = \mathsf{Proof}_E + \mathsf{Proof}_F.$$

Both $\langle C_{\amalg}, \star\rangle$ and $A_{\amalg}$ are coproducts in $\mathbf{Set}_\star$ and **Pfn**, respectively. Therefore, they come equipped with injections: $in^0_C\colon \langle C_E, E\rangle \to \langle C_{\amalg}, \star\rangle$, $in^1_C\colon \langle C_F, F\rangle \to \langle C_{\amalg}, \star\rangle$, and $in^0_A\colon \mathsf{Proof}_E \to A_{\amalg}$ and $in^1_A\colon \mathsf{Proof}_F \to A_{\amalg}$. The other data are defined with the help of these injections, and their properties.

$p_{\amalg} \asymp_{\amalg} q_{\amalg}$ iff
$\quad \exists p, q \in \mathsf{Proof}_E\,.\, p_{\amalg} = in^0_A(p) \wedge q_{\amalg} = in^0_A(q) \wedge p \asymp_E q,$
$\quad$ or $\exists p, q \in \mathsf{Proof}_F\,.\, p_{\amalg} = in^1_A(p) \wedge q_{\amalg} = in^1_A(q) \wedge p \asymp_F q;$

$p_{\amalg}\ I_{\amalg}\ q_{\amalg}$ iff
$\quad \exists p, q \in \mathsf{Proof}_E\,.\, p_{\amalg} = in^0_A(p) \wedge q_{\amalg} = in^0_A(q) \wedge p\ I_E\ q,$
$\quad$ or $\exists p, q \in \mathsf{Proof}_F\,.\, p_{\amalg} = in^1_A(p) \wedge q_{\amalg} = in^1_A(q) \wedge p\ I_F\ q;$

$\partial^i_{\amalg}(in^0_A(p)) = in^0_C(\partial^i_E(p))$ and $\partial^i_{\amalg}(in^1_A(p)) = in^1_C(\partial^i_F(p))$, for $i = 0, 1;$

$l_{\amalg}(in^0_A(p)) = \mathtt{inl}(\ell_E(p))$ and $l_{\amalg}(in^1_A(p)) = \mathtt{inr}(\ell_F(p)).$

Next, we devote some attention to the structure of $S_{E\#F}$.

First, we see that $C_{E\#F} = \{E \# F\} \cup \{E'[\mathtt{inl}] \mid E \prec_S E'\} \cup \{F'[\mathtt{inr}] \mid F \prec_S F'\}$.

Since $E \not\prec_S E$ and $F \not\prec_S F$ it follows that $E[\mathtt{inl}], F[\mathtt{inr}] \notin C_{E\#F}$.

Therefore, the set $\mathsf{Proof}_{E\#F}$ has the following decomposition:

$$\mathsf{Proof}_{E\#F} = \{p \# F \mid p \in \mathsf{Proof}_E \wedge \partial^0(p) = E\} \cup$$
$$\{p[\mathtt{inl}] \mid p \in \mathsf{Proof}_F \wedge \partial^0(p) \neq E\} \cup$$
$$\{E \# p \mid p \in \mathsf{Proof}_F \wedge \partial^0(p) = F\} \cup$$
$$\{p[\mathtt{inr}] \mid p \in \mathsf{Proof}_F \wedge \partial^0(p) \neq F\}.$$

Obviously, the four sets above are pairwise disjoint.

We can define maps $\chi_C: C_{E\#F} \to C_\Pi$, $\chi_A: \mathsf{Proof}_{E\#F} \to A_\Pi$ by:

$$\chi_C(\overline{E}) = \begin{cases} \star = in^0_C(E) = in^1_C(F) & \overline{E} = E \# F \\ in^0_C(E') & \overline{E} = E'[\mathtt{inl}] \\ in^1_C(F') & \overline{E} = F'[\mathtt{inr}] \end{cases}$$

$$\chi_A(p) = \begin{cases} in^0_A(q) & p = q \# F \ \vee \ p = q[\mathtt{inl}] \\ in^1_A(q) & p = E \# q \ \vee \ p = q[\mathtt{inr}] \end{cases}$$

Injectivity of χ_C follows from our observation that $E[\mathtt{inl}], F[\mathtt{inr}] \notin C_{E\#F}$. Otherwise, one readily sees that the functions defined above are bijective.

It is left to the reader to verify that $\chi = \langle \chi_C, \chi_A \rangle: S_{E\#F} \to S_\Pi$ is an isomorphism. Observation that $str(p \# E) = \mathtt{inl}(str(p)) = str(p[\mathtt{inl}])$, and $str(F \# p) = \mathtt{inr}(str(p)) = str(p[\mathtt{inr}])$ helps to show that χ preserves and reflects independency. Preservation and reflection of sameness follows from 9.2.9(i) and 9.2.9(iv). □

Again, for ground terms there are no problems.

Corollary 10.1.8 *$S_{E\#F} \simeq S_E \# S_F$, for $E, F \in \mathsf{T}_\mathcal{L}$.*

Proof. Immediate by Fact 10.1.1(iii). □

Operator _ & _

Lemma 10.1.9 *$S_{E\&F} \simeq S_E \,\&\, S_F$, for $E, F \in \mathsf{Rec}_\mathcal{L}(\emptyset)$.*

Proof. As in the case of the coproduct operator, the denotational meaning of the product operator is the result of taking product in the category **LSS** of all lss-s, and then relabelling the result, so as to find ourselves back into $\mathcal{A}_\mathcal{M}$. For details see definitions and results in Chapter 8, specifically Proposition 8.1.10 and its proof.

In particular we obtain $S_\Pi = \langle C_\Pi, c^*_\Pi, A_\Pi, \asymp_\Pi, I_\Pi, \partial^0_\Pi, \partial^1_\Pi, \mathcal{M}, l_\Pi \rangle$ where the components are defined as follows.

$$\langle C_\Pi, c^*_\Pi \rangle = \langle C_E \times C_F, \langle E, F \rangle \rangle;$$
$$A_\Pi = C_E \times \mathsf{Proof}_F \cup \mathsf{Proof}_E \times \mathsf{Proof}_F \cup \mathsf{Proof}_E \times C_F.$$

Both sets come with projections: $\pi^0_C: C_\Pi \to C_E$, $\pi^1_C: C_\Pi \to C_F$, and $\pi^0_A: A_\Pi \to \mathsf{Proof}_E$ and $\pi^1_A: A_\Pi \to \mathsf{Proof}_F$. Recall that for instance $\pi^0_C\langle E', F' \rangle = E'$, while $\pi^1_A\langle x, y \rangle = y$, provided $y \in \mathsf{Proof}_F$. The remaining components are defined with the help of these projections.

$p_\Pi \asymp_\Pi q_\Pi$ iff

$$(\pi^0_A(p_\Pi)\downarrow \iff \pi^0_A(q_\Pi)\downarrow) \wedge (\pi^0_A(p_\Pi)\downarrow \Rightarrow \pi^0_A(p_\Pi) \asymp_E \pi^0_A(q_\Pi)),$$
$$\text{and} \quad (\pi^1_A(p_\Pi)\downarrow \iff \pi^1_A(q_\Pi)\downarrow) \wedge (\pi^1_A(p_\Pi)\downarrow \Rightarrow \pi^1_A(p_\Pi) \asymp_E \pi^1_A(q_\Pi)),$$

$p_\Pi \; I_\Pi \; q_\Pi$ iff

$(\pi^0_A(p_\Pi)\downarrow \wedge \pi^0_A(q_\Pi)\downarrow) \Rightarrow \pi^0_A(p_\Pi) \Im_E \pi^0_A(q_\Pi))$,

and $(\pi^1_A(p_\Pi)\downarrow \wedge \pi^1_A(q_\Pi)\downarrow) \Rightarrow \pi^1_A(p_\Pi) \Im_E \pi^1_A(q_\Pi))$,

$\pi^0_C(\partial^i_\Pi(p_\Pi)) = \partial^i_E(\pi^0_A(p_\Pi))$ and $\pi^1_C(\partial^i_\Pi(p_\Pi)) = \partial^i_F(\pi^1_A(p_\Pi))$, for $i = 0, 1$;

$$l_\Pi(p_\Pi) = \begin{cases} \langle \ell_E(\pi^0_A(p_\Pi)), \star \rangle & \pi^0_A(p_\Pi)\downarrow \wedge \pi^1_A(p_\Pi)\uparrow \\ \langle \star, \ell_F(\pi^1_A(p_\Pi)) \rangle & \pi^0_A(p_\Pi)\uparrow \wedge \pi^1_A(p_\Pi)\downarrow \\ \langle \ell_E(\pi^0_A(p_\Pi)), \ell_F(\pi^1_A(p_\Pi)) \rangle & \pi^0_A(p_\Pi)\downarrow \wedge \pi^1_A(p_\Pi)\downarrow \end{cases}$$

Now, let us consider $S_{E\&F}$ more closely.

First, let us note that the sets $C_{E\&F}$ and $\mathsf{Proof}_{E\&F}$ have the following structure:

$$C_{E\&F} = \{E' \,\&\, F' \mid E' \in C_E \wedge F' \in C_F\}, \quad \text{and}$$

$$\mathsf{Proof}_{E\&F} = \{p \,\&\, q \mid p \in \mathsf{Proof}_E \wedge q \in \mathsf{Proof}_F\} \cup$$
$$\{p \,\&\, F' \mid p \in \mathsf{Proof}_E \wedge F' \in C_F\} \cup \{E' \,\&\, q \mid E' \in C_E \wedge q \in \mathsf{Proof}_F\}.$$

The proof of the above follows the inductive definition of the sets involved.

The similarity of the above and the structure of C_Π and A_Π is evident. We define $\kappa_C : C_{E\&F} \to C_\Pi$, $\kappa_A : \mathsf{Proof}_{E\&F} \to A_\Pi$ as follows:

$$\kappa_C(E' \,\&\, F') = \langle E', F' \rangle, \text{ and}$$

$$\kappa_A(p) = \begin{cases} \langle p', F' \rangle & p = p' \,\&\, F' \\ \langle E', q' \rangle & p = E' \,\&\, q' \\ \langle p', q' \rangle & p = p' \,\&\, q' \end{cases}$$

It is evident that $\kappa = \langle \kappa_C, \kappa_A \rangle$ consists of bijections. The proof that it is in fact the required isomorphism is standard, and is left to the reader as an exercise. □

10.1.2 Equivalence

We are almost ready to formulate the equivalence result for $\mathcal{L}$ without recursion. It is convenient to collect the results just presented in a single statement.

Lemma 10.1.10 *For every $\sigma \in \Sigma^{\mathcal{L}}$ with* arity $\sigma = n$ *we have*

$$S_{\sigma(F_1,\ldots,F_n)} \simeq \sigma(S_{F_1}, \ldots, S_{F_n}), \quad \text{for } F_1, \ldots, F_n \in \mathsf{T}_{\mathcal{L}}.$$

Proof. Follows from Lemma 10.1.2, Corollary 10.1.4, Lemma 10.1.6, Lemma 10.1.5, Corollary 10.1.8, and Lemma 10.1.9. □

And now comes the main result of this section.

Theorem 10.1.11 (Equivalence – $\mathsf{T}_{\mathcal{L}}$) *For any $E \in \mathsf{T}_{\mathcal{L}}(\mathsf{Var})$, and $\varrho : \mathsf{Var} \to \mathsf{T}_{\mathcal{L}}$:*

$$S_{E\varrho} \simeq A_{\mathcal{M}}[\![E]\!]\tilde{\varrho}.$$

Proof. By induction on the structure of E.

$\boxed{E = x}$ Then

$$S_{E\varrho} = S_{\varrho(x)} = \tilde{\varrho}(x) = \mathcal{A}_{\mathcal{M}}[\![x]\!]\tilde{\varrho} = \mathcal{A}_{\mathcal{M}}[\![E]\!]\tilde{\varrho}.$$

$\boxed{E = \sigma(F_1, \ldots, F_n)}$ Follows easily by induction

$$\begin{aligned}
\mathcal{A}_{\mathcal{M}}[\![E]\!]\tilde{\varrho} &= \sigma(\mathcal{A}_{\mathcal{M}}[\![F_1]\!]\tilde{\varrho}, \ldots, \mathcal{A}_{\mathcal{M}}[\![F_n]\!]\tilde{\varrho}) \\
&\simeq \sigma(S_{F_1\varrho}, \ldots, S_{F_n\varrho}) && \text{by induction } n\text{-times} \\
&\simeq S_{\sigma(F_1\varrho, \ldots, F_n\varrho)} && \text{by Lemma 10.1.10} \\
&= S_{\sigma(F_1, \ldots, F_n)\varrho} && \text{by def., of substitution.}
\end{aligned}$$

□

10.2 $\mathcal{L}$ with Guarded Recursion

10.2.1 The Sublanguage

This time we intend to discuss a larger sublanguage of $\mathcal{L}$. Those terms which are not in it may be thought of as being only *partially* determined. Typical examples include: $\mathtt{rec}\, x.\, x$ and $\mathtt{rec}\, x.\, (m \cdot \Omega)\ \&\ x$.

It was argued that for a recursive process like $\mathtt{rec}\, x.\, E$ its operational meaning is exactly the meaning of its *expansion*, i.e., $E[x \mapsto \mathtt{rec}\, x.\, E]$. This is a consequence of the only inference rule that deals with recursion. Every change that originates in a process, F say, is identified with a proof $p \in \mathsf{Proof}_{\mathcal{L}}$ s.t., $\partial^0(p) = F$. One can think that it stems from the collection of elementary changes of a simple form $m' \cdot F' \xrightarrow{m'} F'$ which lay underneath the leaves of p. Each of $m' \cdot F'$ above is a subprocess of either $F = F_0$, or F_1 that was obtained from F_0 by expansion of some recursive subprocesses as described above, etc.

Moreover, the $m' \cdot F'$-s are not contained, i.e., are not subprocesses, of any other subprocess $m'' \cdot F''$ of appropriate F_n. We call the term F' and all its subterms *guarded* in the term F_n.

Now, to get a complete picture of the possible forms of evolution available from a given process one should carry out the process of expansion of unguarded recursive subprocesses. This may, or may not terminate. For $\mathtt{rec}\, x.\, x$ above it obviously does not.

We conjecture that the sublanguage of $\mathcal{L}$ we are about to define contains precisely those processes $\mathcal{L}$ for which such a complete picture is attainable at any depth of evolution, i.e., for which the procedure of successive expansions terminates.

To formalise the ideas our vocabulary of syntactic notions has to be expanded. We need to talk about particular *occurrences* of one term, usually a variable, within another. This intuitive idea is captured with the help of the notion of a *context*. A context is a term with *holes* in it. Other terms, and contexts, can be *placed* in these holes.

Formally, the set Context of contexts is defined inductively as follows

(i) $[\,]$ is an *empty* context;

(ii) $x \in \mathsf{Var}$ is a context;

(iii) $\sigma \in \Sigma_n^{\mathcal{L}}$, $C_1, \ldots C_n \in \mathsf{Context}$ implies $\sigma(C_1, \ldots, C_n)$ is a context;

(iv) $C \in \mathsf{Context}$, $x \in \mathsf{Var}$ implies $\mathbf{rec}\, x.\, C$ is a context.

Obviously, every term of $\mathcal{L}$ is a context. Another way of looking at contexts is to think of them as obtained by adding the 'hole' as a new variable. The difference is in the way one context is placed in the holes of another, notation $C'[C]$. This is defined inductively, just like the substitution. This time, however, we do not care whether or not some free variables are being captured during the process. Formally,

$$\begin{aligned} [C] &= C \\ x[C] &= x \quad \text{for } x \in \mathsf{Var} \\ \sigma(C_1, \ldots, C_n)[C] &= \sigma(C_1[C], \ldots, C_n[C]) \quad \text{for } \sigma \in \Sigma_n^{\mathcal{L}} \\ (\mathbf{rec}\, x.\, C')[C] &= \mathbf{rec}\, x.\, (C'[C]) \quad \text{for } x \in \mathsf{Var}. \end{aligned}$$

Evidently, if one of the arguments is a term then we get back a term. Even more important is that the operation is associative, i.e., $C''[C'[C]] = (C''[C'])[C]$. Contexts, unlike substitutions, with the operation of placing one in the holes of another form a monoid. The empty context is the obvious identity of this operation.

From now on, only contexts with at most one hole are considered.

Those contexts which do have holes are called *proper*. Proper contexts form a tree when ordered with the relation of occurrence, i.e., C' is greater than C if the latter occurs in the former. For usual terms E and F, F occurs in E formalises the intuitive 'F is a subterm of E'.

An *occurrence of a context C' in a context C''* is a proper context C s.t., $C'' = C[C']$. Then C' is said to *occur* in C''.

An occurrence C of a variable x is ***bound*** iff $C = C'[\mathbf{rec}\, x.\, C'']$, for some contexts C', C''. Otherwise, it is ***free***. It is easy to prove that a variable x has a free occurrence in a term E iff $x \in \mathsf{FV}(E)$.

A free occurrence C of a variable x is ***guarded*** iff $C = C'[m \cdot C'']$, for some contexts C', C''. Otherwise, it is ***unguarded.***

A bound occurrence C of a variable x is ***guarded*** iff C can be decomposed as $C = C'[\mathbf{rec}\, x.\, C'']$, for some contexts C', C'', s.t., C'' is a free and guarded occurrence of x. Otherwise, it is ***unguarded.***

We define the sublanguage $g\mathcal{L}$ of $\mathcal{L}$ as the one containing those terms of $\mathcal{L}$ in which every bound occurrence of any variable is guarded. Its members are referred to as *terms with guarded recursion.* In the future notation $g\mathsf{Rec}_{\mathcal{L}}(X)$ rather than $\mathsf{Rec}_{g\mathcal{L}}(X)$ will be used.

We end this introduction of the sublanguage by quoting a number of results. They are part of the folklore of the subject, and so their proofs are omitted. It is felt that in any case the reader should be able to come up with the proofs her/him-self.

Fact 10.2.1 *Let $E, F \in \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$ and $x \in \mathsf{Var}$. Then*

(i) $E \in g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$ *and* F *occurs in* E *implies* $F \in g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$.

(ii) $\mathbf{rec}\, x.\, E \in g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$ *implies* $E[x \mapsto \mathbf{rec}\, x.\, E] \in g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$.

(iii) $E \in g\mathsf{Rec}_{\mathcal{L}}(\emptyset)$ *and* $\mathcal{L} \vdash E \to F$ *implies* $F \in g\mathsf{Rec}_{\mathcal{L}}(\emptyset)$.

10.2.2 Preliminaries

Next, we prove a technical result that will enable us to cope with recursion. For the third time we are forced to make an extra proviso as to the properties of the terms allowed. Previously, it was done for prefixing and coproduct operators.

At the moment at least we do not have to restrict attention to the terms with guarded recursion. The need for them will come later.

Recursion

Lemma 10.2.2 *Assumption* $\mathbf{rec}\, x.\, E \not\prec_S \mathbf{rec}\, x.\, E$ *implies* $S_{\mathbf{rec}\, x.\, E} \simeq S_{E[x \mapsto \mathbf{rec}\, x.\, E]}$, *for* $x \in \mathsf{Var}$, $E \in \mathsf{Rec}_{\mathcal{L}}(\{x\})$.

Proof. First, let us realise that the assumption is equivalent to $\mathbf{rec}\, x.\, E \not\prec_S E[x \mapsto \mathbf{rec}\, x.\, E]$. For one readily proves that in S:

$$\mathbf{rec}\, x.\, E \prec_S F \quad \text{iff} \quad E[x \mapsto \mathbf{rec}\, x.\, E] \prec_S F.$$

Thus, we can define two bijective maps $\eta_C : C_{E[x \mapsto \mathbf{rec}\, x.\, E]} \to C_{\mathbf{rec}\, x.\, E}$ and $\eta_A : \mathsf{Proof}_{E[x \mapsto \mathbf{rec}\, x.\, E]} \to \mathsf{Proof}_{\mathbf{rec}\, x.\, E}$ by:

$$\eta_C(F) = \begin{cases} \mathbf{rec}\, x.\, E & F = E[x \mapsto \mathbf{rec}\, x.\, E] \\ F & \textit{otherwise} \end{cases}$$

$$\eta_A(p) = \begin{cases} \mathbf{rec}\, x.\, E(p) & p \vdash E[x \mapsto \mathbf{rec}\, x.\, E] \xrightarrow{\mu} F \\ p & \textit{otherwise} \end{cases}$$

It remains to verify that $\eta = \langle \eta_C, \eta_A \rangle$ and $\eta^{-1} = \langle \eta_C^{-1}, \eta_A^{-1} \rangle$ constitute morphisms in the sense of the Definition 8.1.2.

This time condition 8.1.2(iii) is satisfied simply because $p \asymp \mathbf{rec}\, x.\, E(p)$, for $p \vdash E[x \mapsto \mathbf{rec}\, x.\, E] \xrightarrow{\mu} F$. The other conditions follow easily. □

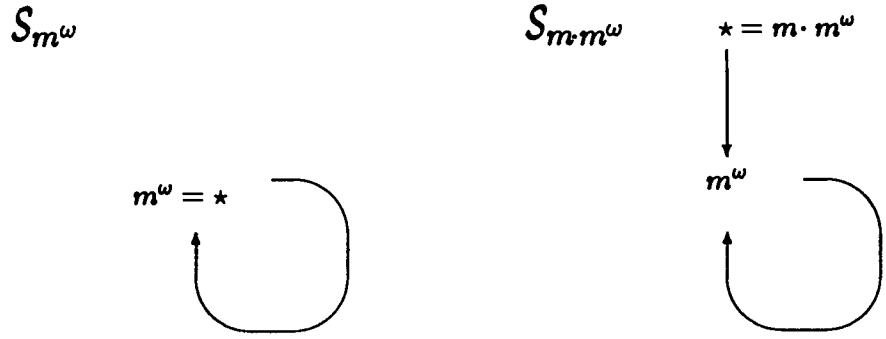
Examples

It seems a right place to demonstrate that the extra assumptions made in the above Lemma, and earlier in Lemma 10.1.3 and Lemma10.1.7, were necessary. The examples will demonstrate that the restriction to terms with guarded recursion is not going to help. Indeed, all the terms considered are from $g\mathcal{L}$. Thus, the problem has its sources elsewhere.

In all examples the information on sameness and independency in the systems is trivial, and therefore passed over. The real identity and labelling of edges should be also apparent, except in the last example when it is provided.

Let us start with an example that refers to the Lemma just proved.

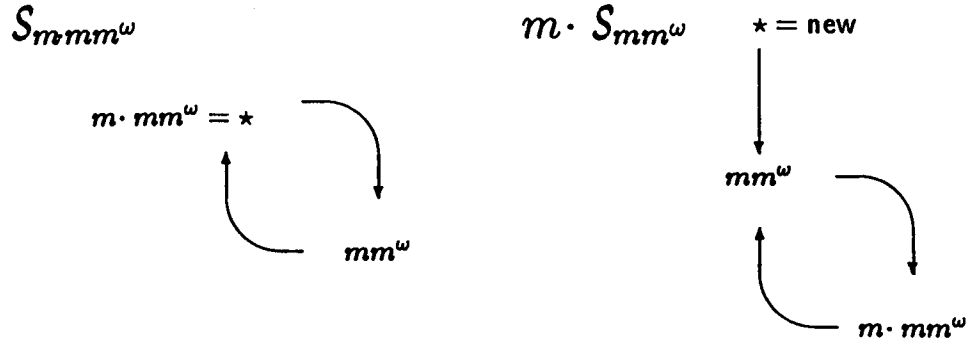
Example 10.2.3 $[S_{\mathbf{rec}\, x.\, E} \not\simeq S_{E[x \mapsto \mathbf{rec}\, x.\, E]}]$ Let $E = m \cdot x$. We abbreviate the term $\mathbf{rec}\, x.\, m \cdot x$ by m^ω. Then $m \cdot m^\omega = E[x \mapsto \mathbf{rec}\, x.\, E]$, and the following labelled skeletal systems correspond to S_{m^ω} and S_{mm^ω}.



In this example $\mathbf{rec}\, x.\, E \prec_S \mathbf{rec}\, x.\, E$. ◇

The next example refers to Lemma 10.1.3.

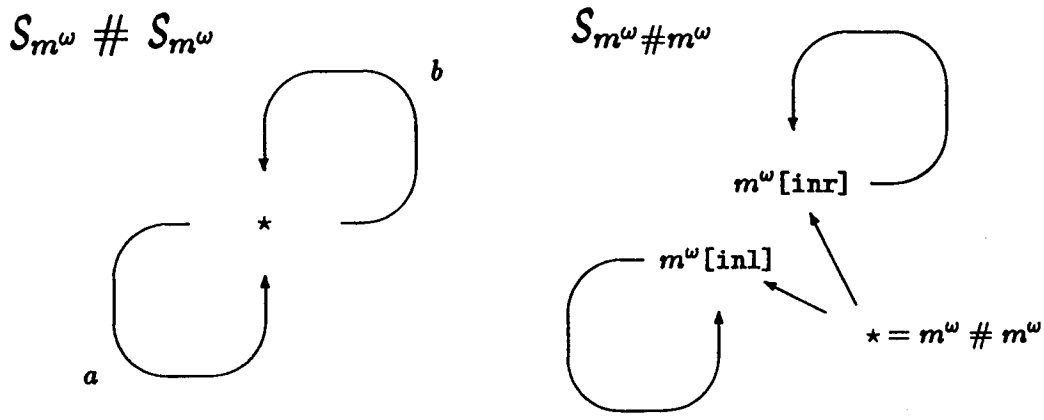
Example 10.2.4 $[m \cdot S_E \not\simeq S_{mE}]$ Let mm^ω abbreviate the term $E = \mathbf{rec}\, x.\, m \cdot m \cdot x$. The first lss below is obtained from the syntactic construction for the term $m \cdot E$. The second as a result of application functor $m \cdot _$ to SL_E.



We have $m \cdot E \prec_S m \cdot E$. ◇

And finally an example showing that the assumption is necessary for the coproduct operator as well.

Example 10.2.5 $[S_{E \# F} \not\simeq S_E \# S_F]$ Let m^ω abbreviate the term $\mathbf{rec}\, x.\, m \cdot x$ as before. We have seen already that $m^\omega \prec_S m^\omega$. For simplicity we take $E = F = m^\omega$, but the example can be modified easily so that only for one, E say, one has $E \prec_S E$. In our case we obtain the following non-isomorphic labelled skeletal systems.



Above, the edges a and b are labelled $\mathtt{inl}\, m$ and $\mathtt{inr}\, m$, respectively.
This time $m^\omega \prec_S m^\omega$.

Above, both assumptions of Lemma 10.1.7 are violated at the same time. But a small change in the above will result in an example that neither of them cannot be violated. For instance it is enough to replace one of m^ω-s with something as simple as $m \cdot \Omega$. $\diamond$

These examples show also that to compare our operational construction has to be adjusted if we are to obtain an agreement with the denotational.

10.2.3 Syntactic Unfolding

Our solution is the following. A translation $\{\!|_|\!\}: \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var}) \to \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$ on terms of $\mathcal{L}$ is first defined. Then, the operational construction is redefined as: $S(E) = S_{\{\!|E|\!\}}$, for $E \in \mathsf{Rec}_{\mathcal{L}}$.

Later, we show that this translation enjoys many nice properties. For instance it commutes with substitution. Also important, it is an invariant of the denotational semantics.

But let us start from the formal definition.

Definition 10.2.6 Let $x \in \mathsf{Var}$, $\sigma \in \Sigma^{\mathcal{L}}_n$, and $E, F_1, \ldots, F_n \in \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$. The map $\{\!|_|\!\}: \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var}) \to \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$ is defined inductively as follows

(i) $\{\!|x|\!\} = x$;

(ii) $\{\!|\sigma(F_1, \ldots, F_n)|\!\} = \sigma(\{\!|F_1|\!\}, \ldots, \{\!|F_n|\!\})$, for $\sigma \neq m \cdot _$;

(iii) $\{\!|m \cdot E|\!\} = ((m \cdot \{\!|E|\!\}) \# \Omega)[\mathrm{R}_+]$;

(iv) $\{\!|\mathtt{rec}\, x.\, E|\!\} = \mathtt{rec}\, x.\, \{\!|E|\!\}$.

Above, $\mathrm{R}_+: \mathcal{M} \to \mathcal{M}$ is defined as in the interpretation of CCS as a sublanguage of $\mathcal{L}$, i.e., we put:

$$\mathrm{R}_+\, \mu = \nu \quad \text{if } \mu = \mathtt{inl}\, \nu \text{ or } \mu = \mathtt{inr}\, \nu, \quad \text{for } \mu \in \mathcal{M}.$$

From the discussion in Chapter 7 it follows that the only activity that $\{\!|m\cdot E|\!\}$ can exhibit is captured by the following derived rule:

$$\frac{}{\{\!|m\cdot E|\!\} \xrightarrow{m} \{\!|E|\!\}\,\mathtt{[inl]}\,[\mathrm{R}_+]}.$$

It demonstrates an important feature of the translation: intuitively, if $\mathcal{L} \vdash \{\!|E|\!\} \xrightarrow{\mu} \{\!|F|\!\}$ then F has more subterms of the form $F'\mathtt{[inl]}$ than E. This property will be used later.

Now, we can prove that $\{\!|_|\!\}$ satisfies the first among the promised properties. First, let us extend the translation onto substitutions by taking the composition, i.e.: $\{\!|\varrho|\!\}\,x = \{\!|\varrho\,x|\!\}$.

Lemma 10.2.7 *Let* $E, F \in \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$, $\varrho\colon \mathsf{Var} \to \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$. *Then*

(i) $\mathsf{FV}(E) = \mathsf{FV}(\{\!|E|\!\})$;

(ii) $\{\!|E|\!\} \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)$ *iff* $E \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)$;

(iii) $\{\!|E\,\varrho|\!\} = \{\!|E|\!\}\,\{\!|\varrho|\!\}$;

(iv) $\{\!|E|\!\} \in g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$ *iff* $E \in g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$;

Proof. (i) By induction on the structure of E.

(ii) Is a consequence of (i).

(iii) By induction on the structure of E.

$\boxed{E = x}$ By definition of $\{\!|\varrho|\!\}$.

$\boxed{E = \sigma(E_1, \ldots, E_n)}$ By induction. Only the case $E = m\cdot F$ is considered, as an example.

$$\begin{aligned}
\{\!|E\varrho|\!\} &= \{\!|m\cdot F\varrho|\!\} \\
&= ((m\cdot \{\!|F\varrho|\!\}) \,\#\, \Omega)\,[\mathrm{R}_+] \\
&= ((m\cdot \{\!|F|\!\}) \,\#\, \Omega)\,[\mathrm{R}_+]\,\{\!|\varrho|\!\} \\
&= \{\!|E|\!\}\{\!|\varrho|\!\}
\end{aligned}$$

$\boxed{E = \mathtt{rec}\,x.\,F}$

$$\begin{aligned}
\{\!|E|\!\}\{\!|\varrho|\!\} &= (\mathtt{rec}\,x.\,\{\!|F|\!\})\{\!|\varrho|\!\} \\
&= \mathtt{rec}\,y.\,\{\!|F|\!\}(\{\!|\varrho|\!\}[x \mapsto y]) \\
&= \mathtt{rec}\,y.\,\{\!|F|\!\}(\{\!|\varrho[x \mapsto y]|\!\}) \\
&= \{\!|\mathtt{rec}\,y.\,F(\{\!|\varrho|\!\}[x \mapsto y])|\!\} \\
&= \{\!|(\mathtt{rec}\,x.\,F)\varrho|\!\} \qquad \text{from (i) and def., of substitution} \\
&= \{\!|E\varrho|\!\}
\end{aligned}$$

(iv) The translation $\{\!|_|\!\}$ can be extended to $\{\!|_|\!\}\colon \mathsf{Context} \to \mathsf{Context}$ by adding $\{\!|[\,]|\!\} = [\,]$, and letting E, F, etc., to range over Context rather than $\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$ in the Definition 10.2.6. Then, by structural induction on C, one shows that $\{\!|C[C']|\!\} = \{\!|C|\!\}[\{\!|C'|\!\}]$. The result follows. □

Our next goal is to show that the assumptions of Lemma 10.2.2, and of other technical results are satisfied by terms in $\{\![\mathsf{Rec}_{\mathcal{L}}(\emptyset)]\!\} = \{\{\![E]\!\} \mid E \in \mathsf{Rec}_{\mathcal{L}}\}$. That is we need to know that the relation $\prec_S$ restricted to $\{\![\mathsf{Rec}_{\mathcal{L}}(\emptyset)]\!\}$ is irreflexive.

To this end, as in the previous section, a measure of complexity is introduced. Next, we show that atomic moves that originate from a term in $\{\![\mathsf{Rec}_{\mathcal{L}}(\emptyset)]\!\}$ end up in terms with higher complexity.

Formally, function $c'\colon \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var}) \to \omega$ is defined by structural induction:

$$c'(\Omega) = c'(x) = 0;$$

$$c'(m\cdot F) = c'(F \setminus L) = c'(\mathtt{rec}\, x.\, F) = c'(F);$$

$$c'(F[R]) = \begin{cases} c'(F)+1 & R = \mathtt{inl} \\ c'(F) & \textit{otherwise} \end{cases};$$

$$c'(F \# F') = c'(F \,\&\, F') = c'(F) + c'(F').$$

As we see the measure of complexity is based on the number of subterms of the form $F[\mathtt{inl}]$ in a given term.

Fact 10.2.8 *Let $E, F \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)$. Then*

(i) If $c'(E\varrho) \geq c'(E)$.

(ii) If $E \in \{\![\mathsf{Rec}_{\mathcal{L}}(\emptyset)]\!\}$ and $\mathcal{L} \vdash E \xrightarrow{\mu} F$ then $\exists F' \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)\,.\, F = \{\![F']\!\}$ and $c'(E) < c'(F)$.

(iii) If $E \in \{\![\mathsf{Rec}_{\mathcal{L}}(\emptyset)]\!\}$ and $E \prec_S F$ then $E \neq F$.

Proof. (i) By induction on the structure of E.

$\boxed{E = x}$ By induction it follows easily that $c'(F) \geq 0$, for $F \in \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$. Thus,

$$c'(E\varrho) \geq 0 = c'(x).$$

$\boxed{E = \sigma(E_1, \ldots, E_n)}$ Follows simply by induction. As an example the case $E = F[R]$ is considered. Then $E\varrho = F\varrho[R]$, and so

$$\begin{array}{rcll} c'(E\varrho) & = & c'(F\varrho[R]) & \\ & = & \begin{cases} c'(F\varrho)+1 & R = \mathtt{inl} \\ c'(F\varrho) & \textit{otherwise} \end{cases} & \\ & \geq & \begin{cases} c'(F)+1 & R = \mathtt{inl} \\ c'(F) & \textit{otherwise} \end{cases} & \text{induction} \\ & = & c'(F[R]) & \end{array}$$

$\boxed{E = \mathtt{rec}\, x.\, F}$ Then $(\mathtt{rec}\, x.\, F)\varrho = \mathtt{rec}\, y.\, (F\varrho[x \mapsto y])$, for some $y \in \mathsf{Var}$. Hence,

$$\begin{array}{rcll} c'(E\varrho) & = & c'(\mathtt{rec}\, y.\, (F\varrho[x \mapsto y]) & \\ & = & c'(F\varrho[x \mapsto y]) & \\ & \geq & c'(F) & \text{induction} \\ & = & c'(\mathtt{rec}\, x.\, F) & \end{array}$$

(ii) Let $E = \{\![E']\!\}$ and $p \vdash E \to F$. The proves goes by induction on the structure of p. Let us consider cases of E'.

$\boxed{E' = \Omega}$ Voidly satisfied, since $p \vdash \Omega \rightarrow$ for no $p \in \mathsf{Proof}_{\mathcal{L}}$.

$\boxed{E' = \sigma(E_1, \ldots, E_n)}$ Only the subcase $E' = m \cdot E''$ is considered. This is basic case of the induction. The others are left ot the reader.

We have $E = ((m \cdot \{\!|E''|\!\}) \# \Omega)[\mathrm{R}_+]$. By the usual argument one shows that the only proof that originates from E is of the form

$$p = \dfrac{\dfrac{\overline{m \cdot \{\!|E''|\!\} \xrightarrow{m} \{\!|E''|\!\}}}{(m \cdot \{\!|E''|\!\}) \# \Omega \xrightarrow{\mathtt{inl}\, m} \{\!|E''|\!\}[\mathtt{inl}]}}{E \xrightarrow{\mathrm{R}_+ \circ \mathtt{inl}\, m} (\{\!|E''|\!\}[\mathtt{inl}])[\mathrm{R}_+]}$$

Thus, $F = (\{\!|E''|\!\}[\mathtt{inl}])[\mathrm{R}_+]$, and so we can take $F' = (E''[\mathtt{inl}])[\mathrm{R}_+]$. Also, $c'(E) = c'(E'')$, while $c'(F) = c'(E'') + 1$, as required.

$\boxed{E' = \mathtt{rec}\, x.\, E''}$ Then $E = \mathtt{rec}\, x.\, \{\!|E''|\!\}$. Therefore, there exists $q \in \mathsf{Proof}_{\mathcal{L}}$ s.t.,

$$p = \dfrac{q \vdash \{\!|E''|\!\}[x \mapsto E] \xrightarrow{\mu} F}{\mathtt{rec}\, x.\, \{\!|E''|\!\} \xrightarrow{\mu} F}.$$

From Lemma 10.2.7(iii) we obtain $\{\!|E''|\!\}[x \mapsto E] = \{\!|E''[x \mapsto E']|\!\}$.

Now, by induction it follows that $F = \{\!|F'|\!\}$, for some $F' \in \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$. Also,

$$\begin{array}{lcll} c'(F) & > & c'(\{\!|E''|\!\}[x \mapsto E]) & \text{by induction} \\ & \geq & c'(\{\!|E''|\!\}) & \text{by (i)} \\ & = & c'(\mathtt{rec}\, x.\, \{\!|E''|\!\}). & \end{array}$$

(iii) Follows by induction from (ii). □

An obvious, and very important consequence of the last result is that the assumptions of Lemma 10.1.3, Lemma 10.1.7, and Lemma 10.2.2 are automatically satisfied for term of the form $\{\!|E|\!\}$, for some $E \in \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$.

Corollary 10.2.9 $S_{E \# F} \simeq S_E \# S_F$, *for* $E, F \in \{\!|\mathsf{Rec}_{\mathcal{L}}(\emptyset)|\!\}$.

Proof. Immediately, from Lemma 10.1.7, by Fact 10.2.8(iii). □

Corollary 10.2.10 $S_{\{\!|m E|\!\}} \simeq m \cdot S_{\{\!|E|\!\}}$, *for* $E \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)$.

Proof. By definition, $\{\!|m \cdot E|\!\} = ((m \cdot \{\!|E|\!\}) \# \Omega)[\mathrm{R}_+]$.

Fact 10.2.8(iii) implies that $m \cdot \{\!|E|\!\} \not\sim_S m \cdot \{\!|E|\!\}$. Thus, $S_{m \{\!|E|\!\}} \simeq m \cdot S_{\{\!|E|\!\}}$ follows from Lemma 10.1.3.

From Corollary 10.2.9 we obtain $S_{m \{\!|E|\!\} \# \Omega} \simeq S_{m \{\!|E|\!\}} \# S_\Omega$. And by Lemma 10.1.5, we obtain $S_{\{\!|m E|\!\}} \simeq (m \cdot S_{\{\!|E|\!\}} \# \Omega)[\mathrm{R}_+]$.

The reader should verify that $S \# S_\Omega \simeq S[\mathtt{inl}]$, for any lss S. Hence, the result follows from the observation that $(S[\mathtt{inl}])[\mathrm{R}_+] = S[id_{\mathcal{M}}] = S$, for any lss S. □

And finally, recursion.

Corollary 10.2.11 $S_{\mathbf{rec}\, x.\, E} \simeq S_{E[x \mapsto \mathbf{rec}\, x.\, E]}$, *for* $x \in \mathsf{Var}$ *and* $E \in \{\!|\mathsf{Rec}_{\mathcal{L}}(\emptyset)|\!\}$.

Proof. Immediately, from Lemma 10.2.2, by Fact 10.2.8(iii). □

The translation $\{\!|_|\!\}$ is an invariant of the denotational semantics.

Lemma 10.2.12 $\mathcal{A}_{\mathcal{M}}[\![E]\!] \cong \mathcal{A}_{\mathcal{M}}[\![\{\!|E|\!\}]\!]$, *for* $E \in \mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$.

Proof. The remark in the proof of Corollary 10.2.10 can be expanded. One can show first that $_ \# \Omega \cong _[\mathtt{inl}]$, and so $_ \# \Omega[\mathrm{R}_+] \cong (_[\mathtt{inl}])[\mathrm{R}_+] = id_{\mathcal{A}_{\mathcal{M}}}$.

Therefore, $\mathcal{A}_{\mathcal{M}}[\![\{\!|m \cdot E|\!\}]\!] \cong \mathcal{A}_{\mathcal{M}}[\![\{\!|E|\!\}]\!]$. The result follows now easily by induction. □

Let us finish with one more remark, before we turn to the proof of the equivalence. The reader may wonder what are the origins of the title of this subsection. It seems, though we will not expand on that matter, that the translation $\{\!|_|\!\}$ is in fact functorial. If $\mathcal{B}$ denotes the full subcategory of $\mathbf{LSS}^{\mathcal{M}}$ with systems S_E, for $E \in \mathsf{Rec}_{\mathcal{L}}(\emptyset)$, as objects, then $\{\!|_|\!\}: \mathcal{B} \to \mathcal{B}$. This functor determines coreflection, i.e., $\{\!|\mathcal{B}|\!\}$ is coreflective in $\mathcal{B}$.

More concretely, let us see what the unfolding does as far as the syntactic construction is concerned.

Example 10.2.13 Let us consider the process $m^\omega = \mathbf{rec}\, x.\, m \cdot x.$ discussed already. The system S_{m^ω} can be found in Example 10.2.3. We want to know how $S_{\{\!|m^\omega|\!\}}$ looks like.

First, one checks that $\{\!|m^\omega|\!\} = \mathbf{rec}\, x.\, ((m \cdot x \# \Omega)[\mathrm{R}_+])$. It is easy to verify that the following is the only evolution from $\{\!|m^\omega|\!\}$ available:

$\{\!|m^\omega|\!\} \xrightarrow{m} (\{\!|m^\omega|\!\} \# \Omega)[\mathrm{R}_+]$.

In what follows $_\mathtt{idl}$ is used to abbreviate $(_ \# \Omega)[\mathrm{R}_+]$, as in Chapter 7. Then we obtain the following evolution taking place from $\{\!|m^\omega|\!\}\mathtt{idl}$:

$\{\!|m^\omega|\!\}\mathtt{idl} \xrightarrow{m} (\{\!|m^\omega|\!\}\mathtt{idl})\mathtt{idl}$.

In general $S_{\{\!|m^\omega|\!\}}$ looks as follows:

$$\{\!|m^\omega|\!\} = \star \longrightarrow \{\!|m^\omega|\!\}\mathtt{idl} \longrightarrow (\{\!|m^\omega|\!\}\mathtt{idl})\mathtt{idl} \longrightarrow \cdots$$

The edges/proofs in the above picture correspond to the evolutions discussed above. ◇

10.2.4 Equivalence

n-isos

To cope with recursion successfully we need to introduce a notion of *approximation*. This is achieved as follows. First, the process of taking a c^*-generated subsystem is stratified. Let S be a lss. There is a natural measure of *sequential distance*, or *sequential depth*, of configurations and edges reachable in S from c^*. This depth is expressed as a natural number. A family of procedures parameterised with natural numbers is introduced. The

n-th procedure takes the n-th *cut* of a lss. It means, roughly, a full subsystem of a given lss with configurations and edges of the sequential depth of at most n.

Formally, let S be a labelled skeletal system.

First, a *sequential length* $\|x\|$ of a sequence of actions of S is defined to be the length of the longest subsequence $y = a_1 \ldots a_n$ of x that satisfies: $\neg a_i \; I_S \; a_{i+1}$, for $i = 1, \ldots, n-1$.

The reader may verify that we could define it for entire traces in $A^*_{I_S}$ as follows:

$$\|t\| = \begin{cases} 0 & t = \varepsilon \\ \|\underline{t}\| + 1 & t \text{ -prime} \\ \max(\|t'\|, \|t''\|) & \exists a, b \in A_S \,.\, a \; I_S \; b \; \wedge \; t'a = t = t''b \end{cases}$$

The important thing in the above definition to notice is that given a sequence xab of S s.t., $a \; I_S \; b$ we have: $\|tab\| = \|xa\| = \|xb\| = \|xba\|$.

Next, we define:

$$C_S^{[n]} = \{c \in C_S \mid \exists x \in A_S^* \,.\, \|x\| \leq n \; \wedge \; c_S^* \xrightarrow{x} c\}, \quad \text{and}$$

$$A_S^{[n]} = \{a \in A_S \mid \partial_S^0(a), \partial_S^1(a) \in C_S^{[n]}\}.$$

We think of elements of $C_S^{[n]}$ and $A_S^{[n]}$ as having sequential depth n. For a configuration this means being at the end of a path of edges that starts from c_S^* and is of the sequential length at most n. For an edge this means occurrence somewhere on such a path.

Among simple consequences of the definition we obtain

$$C_S^{[0]} \subseteq C_S^{[1]} \subseteq \ldots \quad \text{and} \quad A_S^{[0]} \subseteq A_S^{[1]} \subseteq \ldots.$$

We can now define formally what it means to take an n-th cut of a lss. In fact the n-th cut is defined not only on lss-s, but on their morphisms as well.

Definition 10.2.14 For a lss S its *n-th cut*, denoted $S^{[n]}$, is defined as the full subsystem of S generated by $C_S^{[n]}$ and $A_S^{[n]}$.

Given $f: S \to S'$, a morphism of lss-s, its *n-th cut*, denoted $f^{[n]}$, is defined by $S^{[n]} = \langle f_C^{[n]}, f_A^{[n]} \rangle$ where $f_C^{[n]}: C_S^{[n]} \to C_{S'}^{[n]}$ and $f_A^{[n]}: A_S^{[n]} \to A_{S'}^{[n]}$ are restrictions of f_C and f_A, respectively.

Our first task now is to show that the procedure of taking an n-th cut is indeed a functor. Actually, a continuous functor, but this is of no real interest to us.

Proposition 10.2.15 *(i)* $_^{[n]}: \mathbf{LSS}^{\mathcal{M}} \to \mathbf{LSS}^{\mathcal{M}}$ *is a functor.*

(ii) $S^{[n]}$ *is c^*-generated, for any lss S, and* $S^{[n]} \in \mathcal{A}_{\mathcal{M}}$, *for* $S \in \mathcal{A}_{\mathcal{M}}$.

(iii) $(_^{[i]}) \circ (_^{[j]}) = _^{[n]}$ *where* $n = \min(i, j)$.

Proof. (i) Assumptions of Fact 8.1.4 and Fact 8.1.5 are satisfied, so n-th cut is well-defined on lss-s and their morphisms. Evidently, it preserves the identity morphisms, and composition. Hence, it is a functor.

(ii) and (iii) Left to the reader as a simple exercise. □

As a consequence of the above Proposition $_^{[n]}$ cuts down to $_^{[n]}: \mathcal{A}_{\mathcal{M}} \to_\omega \mathcal{A}_{\mathcal{M}}$. It is mostly the latter that we use.

Now, the notion of n-isomorphism, or simply n-iso, is defined.

Definition 10.2.16 Morphism $f: S \to S'$ in $\mathcal{A}_{\mathcal{M}}$ is called *n-iso* iff $f^{[n]}$ is iso.

This categorical definition enables to prove several important properties of n-isos quite easily.

Proposition 10.2.17 *Let $f: S \to S'$, $g: S' \to S''$ be morphisms in $\mathcal{A}_{\mathcal{M}}$. Then*

(i) f is iso iff f is n-iso, for $n \in \omega$.

(ii) f is $n+1$-iso implies f is n-iso.

(iii) f, g are n-isos implies $g \circ f$ is n-iso.

(iv) $g \circ f, f$ are n-isos implies g is n-iso.

(v) $g \circ f, g$ are n-isos implies f is n-iso.

Proof. (i) Let $f: S \to S'$ be a morphism in $\mathcal{A}_{\mathcal{M}}$. From the definition it follows readily that for a c^*-generated system S we have

$$C_S = \bigcup_{n \in \omega} C_S^{[n]} \quad \text{and} \quad A_S = \bigcup_{n \in \omega} A_S^{[n]}.$$

This is the case for S and S' in particular. Since for any $n \in \omega$ both f_C and f_A cut down to bijections $f_C^{[n]}: C_S^{[n]} \to C_{S'}^{[n]}$ and $f_A^{[n]}: A_S^{[n]} \to A_{S'}^{[n]}$, it follows that f_C and f_A are bijections themselves.

Thus it remains to verify that $f^{-1} = \langle f_C^{-1}, f_A^{-1} \rangle$ is a morphism in $\mathcal{A}_{\mathcal{M}}$. But problem of every condition required by Definition 8.1.2 being true can be reduced to the satisfaction of the same condition by one of the $f^{[n]}$, for some $n \in \omega$. E.g., take $a', b' \in A_{S'}$ s.t., $a' \asymp_{S'} b'$. Since the sequence of $A_{S'}^{[n]}$-s is increasing it follows that for some big enough $n \in \omega$ we have $a', b' \in A_{S'}^{[n]}$. Therefore, $a = f_A^{-1}(a'), b = f_A^{-1}(b') \in A_S^{[n]}$, since $f^{[n]}$ is iso, and $a' \asymp_{S'^{[n]}} b'$, by definition. Thus, $a \asymp_{S^{[n]}} b$, since $f^{[n]^{-1}}$ satisfies condition 8.1.2(iii). But it just means that $a \asymp_S b$. Hence, f^{-1} satisfies condition 8.1.2(iii) as well.

(ii) Since $f^{[n+1]}$ is iso we obtain $f^{[n]} = (f^{[n+1]})^{[n]}$, by Proposition 10.2.15(iii). And the latter is iso, since functors preserve them. Hence, f is n-iso.

(iii) (iv) (v) Follow easily from the properties of isos. E.g., $g \circ f, g$ are n-isos iff $g \circ f^{[n]} = g^{[n]} \circ f^{[n]}$ and $g^{[n]}$ are isos. But this implies that $g^{[n]} = g^{[n]} \circ f^{[n]} \circ (f^{[n]})^{-1}$ is iso. Thus, g is n-iso. □

Different set of properties relates the n-th cut functors to those that give interpretation to the constructors of $\mathcal{L}$. These have an important consequence when the operation of these functors on n-isos is considered. In what follows we apply our convention to use the names of constructors from $\mathcal{L}$ also to denote the continuous functors from Chapter 8 which interpret them.

Proposition 10.2.18 *(i) If $\sigma \in \Sigma_i^f$ and $\sigma \neq m\cdot_$ then $\sigma \circ [\underbrace{_^{[n]}, \ldots, _^{[n]}}_{i}] = _^{[n]} \circ \sigma$.*

(ii) $(m\cdot_) \circ (_^{[n]}) = (_^{[n+1]}) \circ (m\cdot_)$.

Proof. (i) Because of the nature of the functor all we need is to show that for any lss $S_1, \ldots, S_i$ from $\mathcal{A}_\mathcal{M}$ we have $\sigma^{[n]}(S_1, \ldots, S_i) = \sigma(S_1^{[n]}, \ldots, S_i^{[n]})$. Let us consider the possible cases of σ. The details are omitted.

$\boxed{\sigma = \Omega}$ This case is trivial, since $\mathbf{1}_{\mathcal{A}_\mathcal{M}} = \Omega() = \mathbf{1}_{\mathcal{A}_\mathcal{M}}{}^{[n]}$, for all $n \in \omega$.

$\boxed{\sigma = _ \setminus L}$ Both $S^{[n]} \setminus L$ and $S \setminus L^{[n]}$ are full subsystems of S. Hence, the result follows if we show that both systems have the same configurations and edges. One verifies that the sets are equal to

$$C = \{c \in C_S \mid \exists x \in A_S^* \,.\, \|x\| \leq n \wedge l_S^*(x) \in (\mathcal{M} - L)^* \wedge c_S^* \xrightarrow{x} c\}, \quad \text{and}$$
$$A = \{a \in A_S \mid \partial_S^0(a), \partial_S^1(a) \in C \wedge (l_S(a) \notin \mathcal{M} \vee l_S(a)\uparrow)\},$$

respectively.

$\boxed{\sigma = _[R]}$ Follows easily since $_^{[n]}$ and $_[R]$ operate on separate parts of the structure of a lss.

$\boxed{\sigma = _ \# _}$ Follows from the observation that $C_{S\#S'}^{[n]}$ is obtained from $C_S^{[n]} + C_{S'}^{[n]}$ by identification of c_S^* with $c_{S'}^*$, and $A_{S\#S'}^{[n]} = A_S^{[n]} + A_{S'}^{[n]}$.

$\boxed{\sigma = _ \,\&\, _}$ This time the result follows from the observation that $C_{S\&S'}^{[n]} = C_S^{[n]} \times C_{S'}^{[n]}$, and $A_{S\&S'}^{[n]} = A_S^{[n]} \times C_{S'}^{[n]} + A_S^{[n]} \times A_{S'}^{[n]} + C_S^{[n]} \times A_{S'}^{[n]}$.

(ii) This time we see that $C_{mS}^{[n+1]} = \mathrm{new}_C(S) + C_S^{[n]}$ and $A_{mS}^{[n+1]} = \mathrm{new}_A(S) + A_S^{[n]}$. □

To be quite precise we do not obtain equality in Proposition 10.2.18(ii), but rather a natural equivalence, but we shall not expand on this technicality. After all it has no adverse effect on the proof of the result stated below.

Corollary 10.2.19 *If $\sigma \in \Sigma_i^f$ and $f_1, \ldots, f_i$ are n-isos then $\sigma(f_1, \ldots, f_i)$ is n-iso. Moreover, $m\cdot f_1$ is $n+1$-iso.*

Proof. For $\sigma \neq m\cdot_$ the Proposition 10.2.18(i) gives $\sigma(f_1, \ldots, f_i)^{[n]} = \sigma(f_1{}^{[n]}, \ldots, f_i{}^{[n]})$ is iso, since functors, here σ, preserve them. Thus, $\sigma(f_1, \ldots, f_i)$ is n-iso.

Similarly, $m\cdot f$ is $n+1$-iso, from Proposition 10.2.18(ii), and so an n-iso, by Proposition 10.2.15(iii). □

Lemma n-iso

The next lengthy result is a necessary technical tool. With it we intend to show that certain endofunctors determined by terms with guarded recursion have *unique fixed point property*. This in turn is crucial when it comes to utilise Corollary 10.2.11.

Lemma 10.2.20 *Let $\tau: \rho \to \rho'$ be a natural iso of environments $\rho, \rho' \in \mathbf{Env}_{\mathcal{A}_\mathcal{M}}$, $f: S \to S'$ an n-iso, $E \in g\mathsf{Rec}_\mathcal{L}(\mathsf{Var})$ and $x \in \mathsf{Var}$. Then*

(i) $\mathcal{A}_\mathcal{M}[\![E]\!]\tau[x \mapsto f]$ *is n-iso.*

(ii) If all free occurrences of x in E are guarded then $\mathcal{A}_\mathcal{M}[\![E]\!]\tau[x \mapsto f]$ *is $n+1$-iso.*

Proof. We prove (i) and (ii) simultaneously by induction on the structure of E.

$\boxed{E = y}$ Two sub-cases need to be considered.

$\underline{x = y}$ Then (ii) follows trivially. As for (i) we have $\mathcal{A}_\mathcal{M}[\![E]\!]\tau[x \mapsto f] = (\tau[x \mapsto f])_x = f$, is n-iso by assumption.

$\underline{x \neq y}$ Therefore, we obtain $\mathcal{A}_\mathcal{M}[\![E]\!]\tau[x \mapsto f] = (\tau[x \mapsto f])_y = \tau_y$ is iso, and hence n-iso for any $n \in \omega$, by Proposition 10.2.17(i).

$\boxed{E = \sigma(F_1, \ldots, F_i)}$ Let $g = \mathcal{A}_\mathcal{M}[\![E]\!]\tau[x \mapsto f]$. Then, by the definition of $\mathcal{A}_\mathcal{M}[\![E]\!]$ we obtain $g = \sigma(\mathcal{A}_\mathcal{M}[\![F_1]\!]\tau[x \mapsto f], \ldots, \mathcal{A}_\mathcal{M}[\![F_i]\!]\tau[x \mapsto f])$.

By induction it follows that $\mathcal{A}_\mathcal{M}[\![F_1]\!]\tau[x \mapsto f]$, $\ldots$, $\mathcal{A}_\mathcal{M}[\![F_i]\!]\tau[x \mapsto f]$ are n-isos, and so (i) follows from Corollary 10.2.19.

If $\sigma = m \cdot _$ then Corollary 10.2.19 shows that g is an $n+1$-iso as well. Otherwise, the assumption that all occurrences of x in E are guarded implies that all occurrences of x in any of $F_1, \ldots, F_i$ are guarded. Thus, $\mathcal{A}_\mathcal{M}[\![F_1]\!]\tau[x \mapsto f], \ldots, \mathcal{A}_\mathcal{M}[\![F_i]\!]\tau[x \mapsto f]$ are $n+1$-isos, and so this time from Corollary 10.2.19 it follows that g is $n+1$-iso.

$\boxed{E = \mathrm{rec}\, y.\, F}$ By definition we have $\mathcal{A}_\mathcal{M}[\![E]\!] = \Upsilon_{\mathcal{A}_\mathcal{M}} \circ curry(\mathcal{A}_\mathcal{M}[\![F]\!] \circ \Phi_y)$. There are two sub-cases to consider.

$\underline{x = y}$ Here, for any $g \in \mathbf{Hom}_{\mathcal{A}_\mathcal{M}}$

$$\begin{aligned} curry(\mathcal{A}_\mathcal{M}[\![F]\!] \circ \Phi_x)\,\tau\, g &= \mathcal{A}_\mathcal{M}[\![F]\!]\tau[x \mapsto g] \\ &= \mathcal{A}_\mathcal{M}[\![F]\!]\tau[x \mapsto f][x \mapsto g] \\ &= curry(\mathcal{A}_\mathcal{M}[\![F]\!] \circ \Phi_x)\,\tau[x \mapsto f]\, g. \end{aligned}$$

Thus, $curry(\mathcal{A}_\mathcal{M}[\![F]\!] \circ \Phi_x)\,\tau[x \mapsto f] = curry(\mathcal{A}_\mathcal{M}[\![F]\!] \circ \Phi_x)\,\tau$. But τ is natural iso, by assumption, so $curry(\mathcal{A}_\mathcal{M}[\![F]\!] \circ \Phi_x)\,\tau$ is a natural iso, and so $\mathcal{A}_\mathcal{M}[\![E]\!]\tau = \Upsilon_{\mathcal{A}_\mathcal{M}} \circ curry(\mathcal{A}_\mathcal{M}[\![F]\!] \circ \Phi_x)\,\tau$ is an n-iso for any $n \in \omega$, by Proposition 10.2.17(i).

$\underline{x \neq y}$ Let $G: \mathbf{Env}_{\mathcal{A}_\mathcal{M}} \to_\omega [\mathcal{A}_\mathcal{M} \to_\omega \mathcal{A}_\mathcal{M}]$ stand for $curry(\mathcal{A}_\mathcal{M}[\![F]\!] \circ \Phi_y)$. Then from the definition of the functor *curry* we obtain for any $\rho_0, \rho_1 \in \mathbf{Env}_{\mathcal{A}_\mathcal{M}}$, $\tau_0: \rho_0 \to \rho_1$ and $S \in \mathcal{A}_\mathcal{M}$:

$G\,\rho_0: \mathcal{A}_\mathcal{M} \to_\omega \mathcal{A}_\mathcal{M}$ where

$G\,\rho_0\,S = \mathcal{A}_\mathcal{M}[\![F]\!]\rho_0[y \mapsto S]$, and

$G\,\rho_0\,f = \mathcal{A}_\mathcal{M}[\![F]\!]id_{\rho_0}[y \mapsto f]$.

Moreover, $G\,\tau_0: G\,\rho_0 \to G\,\rho_1$ where

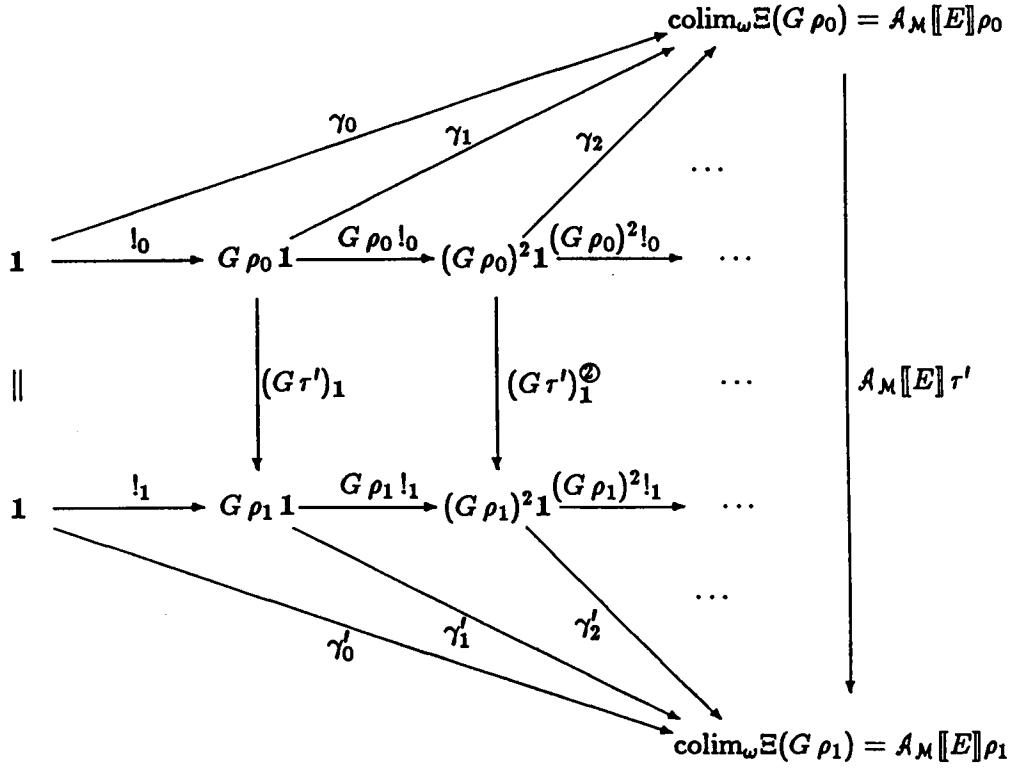
$(G\,\tau_0)_S = \mathcal{A}_\mathcal{M}[\![F]\!]\tau_0[y \mapsto id_S]$.

Also by definition we have $\mathcal{A}_\mathcal{M}[\![E]\!] = \Upsilon_{\mathcal{A}_\mathcal{M}} \circ G = \mathrm{colim}_\omega \circ \Xi \circ G$ where

$\Xi(G\rho) = \mathbf{1} \xrightarrow{!} G\rho\,\mathbf{1} \xrightarrow{G\rho\,!} (G\rho)^2\mathbf{1} \xrightarrow{(G\rho)^2!} \ldots$, where $!: \mathbf{1} \to G\rho\,\mathbf{1}$ is the unique morphism from initial object $\mathbf{1} = \mathbf{1}_{\mathcal{A}_\mathcal{M}}$ of $\mathcal{A}_\mathcal{M}$, and

$(\Xi \circ G\tau)_n = (G\tau)^{@}_{\mathbf{1}}$, for $n \in \omega$, is the $\mathbf{1}$-th component of natural transformation obtained by composing $G\tau$ horizontally n times with itself.

Altogether, we obtain the following commuting diagram, where $\rho_0 = \rho[x \mapsto S]$, $\rho_1 = \rho'[x \mapsto S']$ and $\tau' = \tau[x \mapsto f]$.



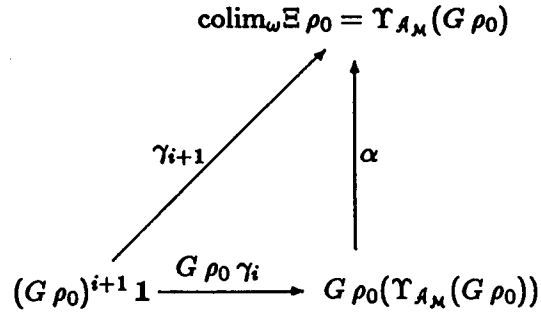
Above $\gamma: \Xi(G\rho_0) \to \mathcal{A}_\mathcal{M}[\![E]\!]\rho_0$, and $\gamma': \Xi(G\rho_1) \to \mathcal{A}_\mathcal{M}[\![E]\!]\rho_1$ are colimiting cocones of the diagrams $\Xi(G\rho_0)$, and $\Xi(G\rho_0)$, respectively. $\mathcal{A}_\mathcal{M}[\![E]\!]\tau'$ is by definition the unique morphism mediating between γ and $\gamma' \circ \Xi(G\tau') = \{\gamma'_n \circ (G\tau')^{@} \mid n \in \omega\}$.

First, by induction on i we show that $(G\rho_0)^i\,!_0$ and γ_i are i-isos. The same argument applied would give $(G\rho_1)^i\,!_1$ and γ'_i are i-isos as well.

For $i = 0$ the result holds trivially, since every morphism is 0-iso.

Thus, let us assume that we have shown the result for i. Since $E = \mathbf{rec}\, x.\, F \in g\mathrm{Rec}_\mathcal{L}(\mathrm{Var})$ it follows that all free occurrences of y in F are guarded, and $F \in g\mathrm{Rec}_\mathcal{L}(\mathrm{Var})$. Moreover, $(G\rho_0)^{i+1}\,!_0 = G\rho_0((G\rho_0)^i\,!_0) = \mathcal{A}_\mathcal{M}[\![F]\!]id_{\rho_0}[y \mapsto (G\rho_0)^i\,!_0]$ is $i+1$-iso, by main induction w.r.t., structure of terms.

As far as γ-s are concerned we have to rely on continuity of the functor $G\rho_0 = curry(\mathcal{A}_\mathcal{M}[\![F]\!] \circ \Phi_y)\rho_0$. This means that there exists an isomorphism $\alpha: G\rho_0(\Upsilon_{\mathcal{A}_\mathcal{M}}(G\rho_0)) \to \Upsilon_{\mathcal{A}_\mathcal{M}}(G\rho_0)$ that mediates between appropriate cocones. In particular the following diagram commutes:



Since γ_i is i-iso we have $G\,\rho_0\,\gamma_i$ is $i+1$-iso, as before, and α j-iso, for any $j \in \omega$, by Proposition 10.2.17(i). Thus, γ_{i+1} is $i+1$-iso by Proposition 10.2.17(iii).

Similarly, for all $i \in \omega$ we have $((G\,\tau')^{\circledR})_1$ is j-iso, where $j = \min(i, n+1)$. The proof proceeds by induction.

For $i = 0$ it is trivial. For $k = i+1$ we have

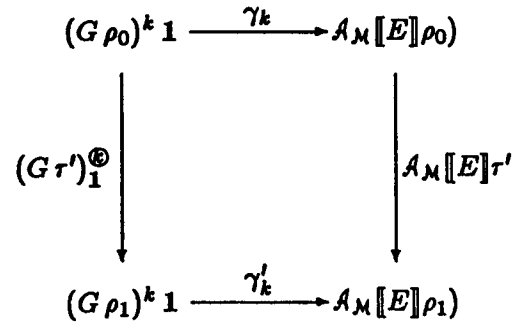
$$((G\,\tau')^{\circledR})_1 = (G\,\tau')_{(G\,\rho_1)^i\,1} \circ G\,\rho_0(G\,\tau')^{\circledR}_1.$$

The second morphism is $\min(i, n+1)+1$-iso as an image via $G\,\rho_0$ of an $\min(i, n+1)$-iso $(G\,\tau')^{\circledR}_1$, the latter by induction hypothesis. But we have also

$$\begin{aligned}(G\,\tau')_{(G\,\rho_1)^i\,1} &= \mathcal{A}_\mathcal{M}[\![F]\!]\tau'[y \mapsto id_{(G\,\rho_1)^i\,1}] \\ &= \mathcal{A}_\mathcal{M}[\![F]\!](\tau[x \mapsto f])[y \mapsto id_{(G\,\rho_1)^i\,1}] \\ &= \mathcal{A}_\mathcal{M}[\![F]\!](\tau[y \mapsto id_{(G\,\rho_1)^i\,1}])[x \mapsto f] \qquad \text{since } x \neq y.\end{aligned}$$

Clearly, $\tau[y \mapsto id_{(G\,\rho_1)^i\,1}]$ is a natural iso. Also, F has all free occurrences of x guarded, since this was true for E. Therefore, $(G\,\tau')_{(G\,\rho_1)^i\,1}$ is $n+1$-iso. By proposition 10.2.15(iii) it follows therefore that $(G\,\tau')^{\circledR})_1$ is $\min(\min(i, n+1)+1, n+1) = \min(i+1, n+1)$-iso.

Finally, let us consider a fragment of our initial diagram. For $k = n+1$ we have



As a consequence of the and Proposition 10.2.17 we obtain that $\gamma_k \circ (G\,\tau')^{\circledR}$ and γ_k are $n+1$-isos. Hence, $\mathcal{A}_\mathcal{M}[\![E]\!]\tau'$ is $n+1$-iso, by Proposition 10.2.17(iv). As required. □

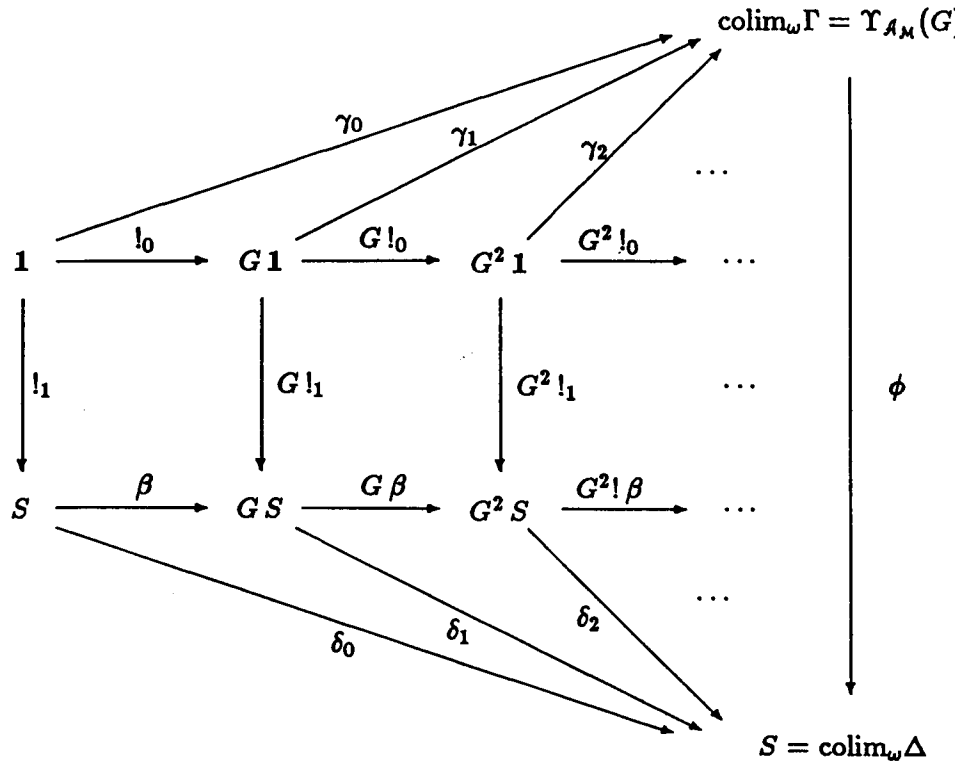
Lemma Unique Fixed Point

In the proof of the Equivalence Theorem one of the properties of the continuous functors of the form $curry(\mathcal{A}_\mathcal{M}[\![E]\!] \circ \phi_x)\rho : \mathcal{A}_\mathcal{M} \to_\omega \mathcal{A}_\mathcal{M}$ is crucial. An endofunctor $G : \mathbf{A} \to_\omega \mathbf{A}$ is said to have ***unique fixed point property*** iff $G(A) \simeq A$ implies $A \simeq \Upsilon_\mathbf{A}(G)$, for $A \in \mathbf{A}$. As it happens

the functors of our interest do have the unique fixed point property, provided rec $x.E$ comes from $g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$.

Lemma 10.2.21 *If $E \in g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$ has all free occurrences of x guarded then for any $\rho \in \mathbf{Env}_{\mathcal{A}_\mathcal{M}}$ functor $curry(\mathcal{A}_\mathcal{M}[\![E]\!] \circ \Phi_x)\rho : \mathcal{A}_\mathcal{M} \to_\omega \mathcal{A}_\mathcal{M}$ has the unique fixed point property.*

Proof. Let $G = curry(\mathcal{A}_\mathcal{M}[\![E]\!] \circ \Phi_x)\rho$. Let us assume that $\langle S, \beta \rangle$ is fixed point of G, i.e., $\beta : G\,S \to S$ is an iso, and let us consider the diagram



where $\Gamma = \mathbf{1} \xrightarrow{!_0} G\,\mathbf{1} \xrightarrow{G!_0} \ldots$ and $\Delta = S \xrightarrow{\beta} G\,S \xrightarrow{G\beta} \ldots$, and $\delta_0 = id_S$, and $\delta_{n+1} = G^n\beta^{-1} \circ \ldots \circ \beta^{-1}$.

The first square commutes by initiality of $\mathbf{1} = \mathbf{1}_{\mathcal{A}_\mathcal{M}}$ in $\mathcal{A}_\mathcal{M}$. The n-th is an image of the first via functor G^n, and so commutes as well. The triangles commute by definition. For our purposes it is enough to see that $\delta : \Delta \to S$ is a cocone. But it is a colimiting one as well. Because $\delta : \Delta \to S$ is a cocone we can define the morphism ϕ above to be the mediating morphism between cocones γ and $\delta' = \{\delta_n \circ G^n\,!_1 \mid n \in \omega\} : \Gamma \to S$.

Therefore, the diagram commutes.

Our task will be accomplished if we show that ϕ is an iso.

First, we note that γ_n is n-iso, for $n \in \omega$. The proof is exactly the same as in the proof of Lemma 10.2.20, and is founded on the continuity of the functor G.

Second, $G^n\,!_1$ is n-iso, for $n \in \omega$. This follows from Lemma 10.2.20 by induction on n. The basic case is trivial. For $n = i + 1$ we have $G^n\,!_1 = G(G^i\,!_1) = curry(\mathcal{A}_\mathcal{M}[\![E]\!] \circ$

$\Phi_x)\rho\,(G^i\,!_1) = \mathcal{A}_{\mathcal{M}}[\![E]\!]id_\rho[x \mapsto G^i\,!_1]$, is n-iso, since $G^i\,!_1$ is, by induction hypothesis an i-iso.

Third, every δ_n is an iso, hence n-iso, for all $n \in \omega$, by Proposition 10.2.17(i).

Finally, for any $n \in \omega$ it follow by Proposition 10.2.17(iii) and Proposition 10.2.17(iv) that ϕ is an n-iso. The result then follows by application of Proposition 10.2.17(i) again. □

Equivalence

For convenience the technical results established so far are collected in a single statement.

Lemma 10.2.22 *For every $\sigma \in \Sigma^{\mathcal{L}}$ with* arity $\sigma = n$ *we have*

$$S_{\{\!|\sigma(F_1,\ldots,F_n)|\!\}} \simeq \sigma(S_{\{\!|F_1|\!\}},\ldots,S_{\{\!|F_n|\!\}}), \quad \text{for } F_1,\ldots,F_n \in \mathsf{Rec}_{\mathcal{L}}(\emptyset).$$

Proof. Follows from Lemma 10.1.2, Corollary 10.2.10, Lemma 10.1.6, Lemma 10.1.5, Corollary 10.2.9, and Lemma 10.1.9. □

And now comes the main result of this section. This time for a closed substitution ϱ: Var $\to g\mathsf{Rec}_{\mathcal{L}}(\emptyset)$ by $\widetilde{\varrho}$ we denote the environment λx:Var. $S_{\{\!|\varrho\, x|\!\}}$.

Theorem 10.2.23 (Equivalence) *For any $E \in g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$, and ϱ: Var $\to g\mathsf{Rec}_{\mathcal{L}}(\emptyset)$*

$$S_{\{\!|E\varrho|\!\}} \simeq \mathcal{A}_{\mathcal{M}}[\![E]\!]\,\widetilde{\varrho}.$$

Proof. By induction on the structure of E.

$\boxed{E = x}$ Then

$$S_{\{\!|E\varrho|\!\}} = S_{\{\!|\varrho|\!\}(x)} = \widetilde{\varrho}(x) = \mathcal{A}_{\mathcal{M}}[\![x]\!]\,\widetilde{\varrho} = \mathcal{A}_{\mathcal{M}}[\![E]\!]\,\widetilde{\varrho}.$$

$\boxed{E = \sigma(F_1,\ldots,F_n)}$ Follows easily by induction.

$$\begin{aligned}
\mathcal{A}_{\mathcal{M}}[\![E]\!]\,\widetilde{\varrho} &= \sigma(\mathcal{A}_{\mathcal{M}}[\![F_1]\!]\,\widetilde{\varrho},\ldots,\mathcal{A}_{\mathcal{M}}[\![F_n]\!]\,\widetilde{\varrho}) \\
&\simeq \sigma(S_{\{\!|F_1\varrho|\!\}},\ldots,S_{\{\!|F_n\varrho|\!\}}) \qquad \text{by induction } n\text{-times} \\
&\simeq S_{\sigma(\{\!|F_1\varrho|\!\},\ldots,\{\!|F_n\varrho|\!\})} \qquad \text{by Lemma 10.2.22} \\
&= S_{\{\!|\sigma(F_1,\ldots,F_n)\varrho|\!\}}
\end{aligned}$$

$\boxed{E = \mathtt{rec}\, x.\, F}$ First, we note that the following takes place.

$$\begin{aligned}
S_{\{\!|E\varrho|\!\}} &= S_{\{\!|\mathtt{rec}\, y.\,(F\varrho[x\mapsto y])|\!\}} \qquad \text{for some } y \in \mathsf{Var}, \text{ by def., of substitution} \\
&\simeq S_{\{\!|(F\varrho[x\mapsto y])[y\mapsto \mathtt{rec}\, y.\,(F\varrho[x\mapsto y])]|\!\}} \qquad \text{by Corollary 10.2.11} \\
&= S_{\{\!|F(\varrho[x\mapsto E\varrho])|\!\}} \qquad \text{by associativity of substitution} \\
&\simeq \mathcal{A}_{\mathcal{M}}[\![F]\!]\,(\varrho[x \mapsto \widetilde{E\varrho}]) \qquad \text{by induction} \\
&= \mathcal{A}_{\mathcal{M}}[\![F]\!]\,\widetilde{\varrho}[x \mapsto S_{\{\!|E\varrho|\!\}}] \\
&= curry(\mathcal{A}_{\mathcal{M}}[\![F]\!] \circ \Phi_x)\,\widetilde{\varrho}\; S_{\{\!|E\varrho|\!\}}.
\end{aligned}$$

In other words $S_{\langle\!|E\varrho|\!\rangle}$ is a fixed point of a ω-functor $curry(\mathcal{A}_{\mathcal{M}}[\![F]\!] \circ \Phi_x)\,\widetilde{\varrho}$. But all free occurrences of x in F are guarded, since $E \in g\mathrm{Rec}_{\mathcal{L}}(\mathsf{Var})$, and so the functor $curry(\mathcal{A}_{\mathcal{M}}[\![F]\!] \circ \Phi_x)\,\widetilde{\varrho}$ has, by Lemma 10.2.21, the unique fixed point property. Thus,

$$\begin{aligned} S_{\langle\!|E\varrho|\!\rangle} &\simeq \Upsilon_{\mathcal{A}_{\mathcal{M}}}(curry(\mathcal{A}_{\mathcal{M}}[\![F]\!] \circ \Phi_x)\,\widetilde{\varrho}) \\ &= (\Upsilon_{\mathcal{A}_{\mathcal{M}}} \circ curry(\mathcal{A}_{\mathcal{M}}[\![F]\!] \circ \Phi_x))\,\widetilde{\varrho} \\ &= \mathcal{A}_{\mathcal{M}}[\![\mathrm{rec}\, x.\, F]\!]\,\widetilde{\varrho}, \end{aligned}$$

as required. □

The reader may have noticed that we did not have to use the Substitution Lemma in the proof of the Theorem. Associativity of composition of substitution was used instead.

We end the section with a statement that expresses our belief that the equivalence result holds for the full language $\mathcal{L}$.

Conjecture 10.2.24 $S_{\langle\!|E\varrho|\!\rangle} \simeq \mathcal{A}_{\mathcal{M}}[\![E]\!]\,\widetilde{\varrho}$, *for any* $E \in \mathrm{Rec}_{\mathcal{L}}(\mathsf{Var})$, *and* $\varrho\colon \mathsf{Var} \to g\mathrm{Rec}_{\mathcal{L}}(\emptyset)$.

Unfortunately, the natural method of proving the Conjecture does not work. That is the clause $\forall F \in \mathrm{Rec}_{\mathcal{L}}(\mathsf{Var})\, \exists E \in g\mathrm{Rec}_{\mathcal{L}}(\mathsf{Var})\,.\, \mathcal{A}_{\mathcal{M}}[\![F]\!] \cong \mathcal{A}_{\mathcal{M}}[\![E]\!]$ – does not hold. The problem can be exemplified with the term $(m\cdot\Omega)_\infty = \mathrm{rec}\, x.\, (m\cdot\Omega)\ \&\ x$. The lss $S_{m\Omega_\infty}$ is not *finitely branching*, i.e., it has an infinite number of edges leaving the initial configuration. Terms with guarded recursion, on the other hand, always yield finitely branching systems.

10.3 The Problem of Full Abstraction

In this last section the equivalence result just obtained is related to another accepted notion of equivalence between the operational and denotational semantics.

At least for a moment we abstract away from the syntactic details of the language $\mathcal{L}$, and carry on the discussion using general terminology. Thus, we assume that a signature Σ was given, together with a(n algebraic) Σ-domain $\mathbf{C}$. This determines the denotational semantics $\mathbf{C}[\![_]\!]\colon \mathrm{Rec}_\Sigma(\mathsf{Var}) \to [\mathbf{Env_C} \to \mathbf{C}]$.

One can define two equivalence relations on Σ-terms:

$$t \sim_{\mathbf{C}[\![_]\!]} t' \quad \text{iff} \quad \forall \rho \in \mathbf{Env_C}\, \mathbf{C}[\![t]\!]\,\rho = \mathbf{C}[\![t']\!]\,\rho \tag{10.4}$$

$$t \approx_{\mathbf{C}[\![_]\!]} t' \quad \text{iff} \quad \mathbf{C}[\![t]\!] = \mathbf{C}[\![t']\!]. \tag{10.5}$$

Obviously, we have $\approx_{\mathbf{C}[\![_]\!]} = \sim_{\mathbf{C}[\![_]\!]}$. Moreover, the relations are Σ-*congruences*. Let us remind the reader that a relation Q on a Σ-algebra $\mathcal{A}$ is a Σ-congruence whenever it is *substitutive*, i.e., $\sigma(a_1,\ldots,a_n)Q\sigma(a_1',\ldots,a_n')$ whenever a_iQa_i', for $a_i, a_i' \in \mathcal{A}$, $i = 1,\ldots,n$. The Σ-algebra we have in mind here has $\mathrm{Rec}_\Sigma(\mathsf{Var})$ as the carrier, and the constructors acting as operators.

Usually, operational semantics specifies a notion of *behaviour* $B(t)$, for every closed Σ-term t. This induces another equivalence on $\mathrm{Rec}_\Sigma(\mathsf{Var})$

$$t \sim_B t' \quad \text{iff} \quad \forall \varrho\colon \mathsf{Var} \to \mathrm{Rec}_\Sigma(\emptyset)\,.\, B(t\,\varrho) = B(t'\,\varrho). \tag{10.6}$$

This relation is not a Σ-congruence in general. There is however the greatest Σ-congruence $\approx_B$ contained in it, and defined by

$$t \approx_B t' \quad \text{iff} \quad \forall C \in \mathsf{Context}.\, C[t] \sim_B C[t']. \tag{10.7}$$

The connection between the operational and denotational semantics can be made by relating those congruences.

There is a minimal pragmatic requirement often made. If a denotational semantics is to be used as a reference to program manipulation we must be sure that the terms it equates exhibit the same behaviour, i.e., $\sim_{\mathbf{C}[\![_]\!]} \subseteq \approx_B$. Then one says that the denotational semantics is *adequate*, [Milner 75], or *correct*, [Plotkin 82], [Stoughton 86].

If we have $\sim_{\mathbf{C}[\![_]\!]} = \approx_B$, then the denotational semantics is called *fully abstract*.

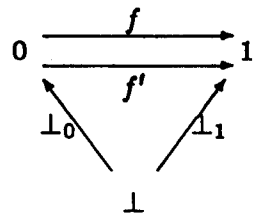
When one tries to adjust the above definitions to work for categorical denotational semantics a problem arises. The first step has to be a relaxation of the conditions (10.4) and (10.5), since equality is now too strong a requirement. Thus, we obtain:

$$t \sim_{\mathbf{C}[\![_]\!]} t' \quad \text{iff} \quad \forall \rho \in \mathbf{Env_C} \mathbf{C}[\![t]\!]\,\rho \simeq \mathbf{C}[\![t']\!]\,\rho \tag{10.8}$$

$$t \approx_{\mathbf{C}[\![_]\!]} t' \quad \text{iff} \quad \mathbf{C}[\![t]\!] \cong \mathbf{C}[\![t']\!]. \tag{10.9}$$

The problem is that these two congruences are now longer identical. Obviously, we have $\approx_{\mathbf{C}[\![_]\!]} \subseteq \sim_{\mathbf{C}[\![_]\!]}$, but the converse need not take place.

Example 10.3.1 Take $\Sigma = \{\sigma, \sigma'\}$ with $\mathsf{arity}(\sigma) = \mathsf{arity}(\sigma') = 1$. And consider category $\mathbf{A}$ that has only three objects: $\bot$, 0 and 1, and four non-identity morphisms: $f, f' : 0 \to 1$, and $\bot_i : \bot \to i$, for $i = 0, 1$, such that the following diagram commutes.



The reader may check that $\mathbf{A}$ is a ω-category. It can be made into a Σ-domain as well. To this end define two ω-functors $\sigma, \sigma' : \mathbf{A} \to \mathbf{A}$ by

$\sigma(A) = \sigma'(A) = A$, for $A \in \mathbf{A}$, and

$\sigma(\bot_i) = \sigma'(\bot_i) = \bot_i$, and $\sigma(f) = \sigma(f') = f$ and $\sigma'(f) = \sigma'(f') = f'$.

Then $\sigma(x) \sim_{\mathbf{A}[\![_]\!]} \sigma'(x)$, since for $\rho \in \mathbf{Env_A}$ we have $\mathbf{A}[\![\sigma(x)]\!]\,\rho = \mathbf{A}[\![\sigma'(x)]\!]\,\rho$. But, $\sigma(x) \not\approx_{\mathbf{A}[\![_]\!]} \sigma'(x)$, on the other hand. ◇

There remains a similar change to be made in the definition of the operational equivalence if, as we have it, the behaviours are objects of a category. Rather than (10.6) we should take

$$t \sim_B t' \quad \text{iff} \quad \forall \varrho : \mathsf{Var} \to \mathsf{Rec}_\Sigma(\emptyset).\, B(t\,\varrho) \simeq B(t'\,\varrho). \tag{10.10}$$

Now, let us return to $\mathcal{L}$, or rather to its sublanguage $g\mathcal{L}$ of terms with guarded recursion. Since the behaviour function was named S, $S: g\mathrm{Rec}_{\mathcal{L}}(\emptyset) \to \mathcal{A}_{\mathcal{M}}$, we denote the operational equivalences by sim_S and $\approx_S$, respectively.

First, we note that the Equivalence Theorem implies correctness of the denotational semantics w.r.t., the operational semantics.

Theorem 10.3.2 (Adequacy) $\approx_{\mathcal{A}_{\mathcal{M}}[\![-]\!]} \subseteq \sim_{\mathcal{A}_{\mathcal{M}}[\![-]\!]} \subseteq \approx_S = \sim_S$.

Proof. To show $\approx_S = \sim_S$ it is enough to check that the latter is a $\Sigma^{\mathcal{L}}$-congruence. But this is an easy consequence of the Lemma 10.2.22.

We have remarked already that the first inclusion holds, and thus we are left to prove the second. Even without the equality $\approx_S = \sim_S$ it would be enough to prove $\sim_{\mathcal{A}_{\mathcal{M}}[\![-]\!]} \subseteq \sim_S$, to conclude that $\sim_{\mathcal{A}_{\mathcal{M}}[\![-]\!]} \subseteq \approx_S$ also holds.

Let $E \sim_{\mathcal{A}_{\mathcal{M}}[\![-]\!]} E'$. Then for any closed substitution $\varrho: \mathrm{Var} \to g\mathrm{Rec}_{\mathcal{L}}(\emptyset)$ we have

$$\begin{array}{lcll} S(E\varrho) & = & S_{\{\![E\varrho]\!\}} & \\ & \simeq & \mathcal{A}_{\mathcal{M}}[\![E]\!]\,\widetilde{\varrho} & \text{Equivalence Theorem., Theorem 10.2.23} \\ & \simeq & \mathcal{A}_{\mathcal{M}}[\![E']\!]\,\widetilde{\varrho} & \text{assumption} \\ & \simeq & S_{\{\![E'\varrho]\!\}} & \text{Equivalence Theorem., Theorem 10.2.23} \\ & = & S(E'\varrho). & \end{array}$$

Thus, $E \sim_S E'$, as required. □

There are two difficulties with proving the converse result, i.e., showing full abstraction.

First, it is an open question whether the equality $\sim_{\mathcal{A}_{\mathcal{M}}[\![-]\!]} = \approx_{\mathcal{A}_{\mathcal{M}}[\![-]\!]}$ takes place for the denotational semantics for $g\mathcal{L}$ presented in Chapter 8.

Another problem is to obtain $\sim_{\mathcal{A}_{\mathcal{M}}[\![-]\!]} \supseteq \sim_S$. Here, our Equivalence Theorem appears useful as well, since it enables us to reduce this problem of *weak full abstraction*, to the question of *expressiveness* of $\mathcal{L}$.

Let us call $\mathcal{L}$ *expressive* with respect to $\mathcal{A}_{\mathcal{M}}$ iff for any $S \in \mathcal{A}_{\mathcal{M}}$ there exists a ω-diagram $\Xi = S(E_0) \xrightarrow{f_0} S(E_1) \xrightarrow{f_1} \ldots$ s.t., $S = \mathrm{colim}_\omega \Xi$, where $E_n \in \mathrm{T}_{\mathcal{L}}$, for $n \in \omega$ are ground terms of $\mathcal{L}$. Then we obtain the following result.

Theorem 10.3.3 *If $\mathcal{L}$ is expressive w.r.t, $\mathcal{A}_{\mathcal{M}}$ then $\sim_{\mathcal{A}_{\mathcal{M}}[\![-]\!]} = \sim_S$.*

Proof. From Theorem 10.3.2 it follows that only inclusion $\sim_{\mathcal{A}_{\mathcal{M}}[\![-]\!]} \supseteq \sim_S$ may ever be in question. So, let $E \sim_S E'$, i.e., $S(E\varrho) \simeq S(E'\varrho)$, for any closed substitution ϱ.

Now, let $\rho \in \mathbf{Env}_{\mathcal{A}_{\mathcal{M}}}$ be an environment. From the expressiveness of $\mathcal{L}$ we obtain for each $x \in \mathrm{Var}$ a ω-diagram $\Xi^x = S(E_0^x) \xrightarrow{f_0^x} S(E_1^x) \xrightarrow{f_1^x} \ldots$ s.t., $\rho(x) = \mathrm{colim}_\omega \Xi^x$. Let us put $\varrho_n = \lambda x{:}\mathrm{Var}.\, E_n^x$ and $(\tau_n)_x = f_n^x$. Then $\widetilde{\varrho_n}$ are environments, while $\tau_n: \rho_n \to \rho_{n+1}$ their morphism, and $\rho = \mathrm{colim}_\omega(\rho_0 \xrightarrow{\tau_0} \ldots)$. Therefore,

$$
\begin{array}{lcll}
\mathcal{A}_{\mathcal{M}}[\![E]\!]\,\rho & \simeq & \mathrm{colim}_{\omega}\mathcal{A}_{\mathcal{M}}[\![E]\!]\,\widetilde{\rho_n} & \mathcal{A}_{\mathcal{M}}[\![E]\!] \text{-continuous} \\
 & \simeq & \mathrm{colim}_{\omega}S(E\,\widetilde{\varrho_n}) & \text{Equivalence Theorem., Theorem 10.2.23} \\
 & \simeq & \mathrm{colim}_{\omega}S(E'\,\widetilde{\varrho_n}) & \text{assumption} \\
 & \simeq & \mathrm{colim}_{\omega}\mathcal{A}_{\mathcal{M}}[\![E']\!]\,\widetilde{\varrho_n} & \text{Equivalence Theorem., Theorem 10.2.23} \\
 & = & \mathcal{A}_{\mathcal{M}}[\![E']\!]\,\rho & \mathcal{A}_{\mathcal{M}}[\![E']\!] \text{-continuous.}
\end{array}
$$

As required. □

There is another possibility to established stronger version of the full abstraction, i.e., $\approx_{\mathcal{A}_{\mathcal{M}}[\![-]\!]} = \approx_{S}$. Since the behaviour are objects of a category it is possible to construct them categorically, i.e., as certain functors. Formally, one should take the full subcategory, call it $\mathbf{B}$, of $\mathcal{A}_{\mathcal{M}}$ given by $S(E)$, for $E \in \mathsf{Rec}_{(}\emptyset)$. Then, the semantics $\mathbf{B}[\![E]\!]{:}\mathbf{Env}_{\mathbf{B}} \to \mathbf{B}$ should be defined syntactically, so that for instance $\mathbf{B}[\![E]\!]\,\tilde{\varrho} = S(E\,\varrho)$. Then a stronger version of the Equivalence Theorem, Theorem 10.2.23, should be proved that asserts, roughly, that $\mathbf{B}[\![E]\!] \cong \mathcal{A}_{\mathcal{M}}[\![E]\!]$, for $E \in g\mathsf{Rec}_{\mathcal{L}}(\mathsf{Var})$. This seems feasible, at least for the language without recursion.

We do not feel strongly motivated to pursue the idea as there are more important, and unanswered question, e.g., expressiveness. Another point is that a stronger definition of expressiveness would be probably required to establish result corresponding to Theorem 10.3.3.

Chapter 11

Conclusions

In the preceding Chapters, categories of asynchronous systems were defined and studied.

In Part I they were compared with other models for concurrency. It was argued that they underly the others in the sense that e.g., prime event structures, trace languages are coreflective in **AS** — the category of asynchronous systems.

The important open problem remains whether asynchronous systems can themselves be regarded as a coreflective subcategory of safe Petri nets. It would be also interesting to know whether forward stable asynchronous systems are reflective in **AS**. A positive answer would nicely extend a reflection between ideal trace languages, and trace languages established in Chapter 4.

There is a whole world of models for concurrency more general than asynchronous systems. One can mention not only general Petri nets considered here, but also general event structures, see [APN'86/II], and others. We have deliberately circumscribed the generality of our model, and therefore the above do not fit into the picture very well.

It would be interesting to mark the limits of possible unified theory of models for concurrency. For example one can try to extend the main result in Chapter 3, and show that the functor SYS: **PNet** $\rightarrow$ **PSys** is not a (left) adjoint. Alternatively, one can attempt to find a model that would encompass more than our asynchronous systems managed to achieve here. For instance, if in asynchronous systems one allows the independency relation to vary from state to state then general event structures can be embedded by a functor extending AS $\circ$ TL from Chapter 5.

In Part II the category **LSS** of labelled skeletal systems was used in the studies of the semantics of *CCS*-like languages. A specific language $\mathcal{L}$ for synchronisation was given an axiomatic operation semantics à la Plotkin in Chapter 7. In Chapter 8 $\mathcal{L}$ was given a (categorical) denotational semantics in a subcategory $\mathcal{A}_M$ of **LSS**. The elements of the operational description were used to construct a labelled skeletal system for each closed term of $\mathcal{L}$ in Chapter 9. This construction was compared to the denotational in Chapter 10. We have been able to prove them equivalent for certain sublanguages of $\mathcal{L}$.

The above picture of the accomplishments of Part II has many holes in it.

To start with some basic questions, the problem of divergence was not even touched upon.

Also, no equational characterisation of the equivalence among terms determined by the denotational semantics was offered. In the author's opinion it might be impossible to achieve this goal. Perhaps, one should try to develop a more extensional notion of behaviour instead, for which a finite axiomatisation can be found more easily.

There are more specific problems that were left aside as not contributing directly to the main subject. Others, simply remain unsolved.

First, there is a question of the categorical properties of **LSS** and its subcategories. Specifically, it would be interesting to know if **LSS** is small cocomplete as well as small complete.

Second, it was shown that the restriction functors arise from certain functor being a fibration. The problem is that this functor does not arise as naturally as one would like. To remedy this defect one can try to consider a more general class of morphisms, as indicated in the construction of the fibration, and check that all the important categorical properties of **LSS** are not adversely affected. Also, at present the fibration is established for injective partial functions only. One should check if a general result also holds.

Finally, no attempt was made to relate the denotational semantics known from the literature with the one given in Chapter 8. For some of them a direct link is impossible at the moment. In case of the general event structure semantics presented in [Winskel 82] one would have to *unfold* both of them to the semantics in a common coreflective subcategory of prime event structures, and make the comparison there. It should be much easier task in case of the safe nets semantics suggested in [Winskel 84a], since we have a direct translation of safe nets to asynchronous systems. The author believes that in both cases a natural equivalence can be established between the translations of the interpreting functors.

In Chapter 9 we have seen that there are some limits to the adequacy of axiomatic operational semantics as a method of presentation of concurrent evolutions of systems. The negative result for *CCS* with its original axiomatic operational semantics presented there was obtained despite apparently mild and natural restrictions imposed on the construction. Recently, in [Olderog 87] and [Degano et al 87a], similar construction in safe nets were presented. It would be interesting to see how they differ from the construction given in Chapter 9. The immediate comparison is difficult to carry out for the axiomatic operational semantics used there differs, at least in the method of presentation. Nevertheless, we know how to translate safe nets into asynchronous systems, and so one can compare the results of these two constructions with ours.

In Chapter 10 the equivalence was established between the denotational and the syntactic constructions for two sublanguages of $\mathcal{L}$. It remains to extend the results to the full language.

The equivalence result for the sublanguage with guarded recursion involves a syntactic unfolding operation. This is necessary, for we have seen that the syntactic construction can produce finites systems with loops, which denotationally are modelled by infinite systems.

In principle at least, it might be possible to obtain a better correspondence. Especially interesting would be a possibility to produce finite systems denotationally.

The equivalence results established make the denotational semantics *adequate*, or *correct* w.r.t., the operational. We have seen that the notion of full abstraction comes in two different forms, if the categorical semantics is used.

A positive answer to the question of expressiveness of $\mathcal{L}$ w.r.t., $\mathcal{A}_{\mathcal{M}}$ would give a weak full abstraction result.

To obtain a stronger result, the syntactic construction would have to be extended to a syntactic definition of functors.

Bibliography

[Aalbersberg&Rozenberg 86] Aalbersberg, I. J. and G. Rozenberg. Theory of Traces. Technical Report 86–16, Leiden University, 1986.

[Abramsky 83] Abramsky, S. On semantic foundations for applicative multiprogramming. *Proc. 10th ICALP*, Lecture Notes in Computer Science, vol. 154, Springer-Verlag, 1983.

[ADJ 77] Goguen, J., J. Thatcher, E. Wagner and J. Wright. Initial algebra semantics and continuous algebras. *Journal of the ACM* 24, 1, 1977.

[APN'86/I] *Petri Nets: Central Models and Their Properties* Advances in Petri Nets 1986, Part I. Proceedings of an Advanced Course, Bad Honnef, September 1986. Lecture Notes in Computer Science, vol. 254, Springer-Verlag, 1987.

[APN'86/II] *Petri Nets: Applications and Relationship to Other Models of Concurrency*. Advances in Petri Nets 1986, Part II. Proceedings of an Advanced Course, Bad Honnef, September 1986. Lecture Notes in Computer Science, vol. 255, Springer-Verlag, 1987.

[APN'87] *Advances in Petri Nets 1987*, Lecture Notes in Computer Science, vol. 266, Springer-Verlag, 1987.

[Arbib&Manes 75] Arbib, M. A. and E. G. Manes. *Arrows, Structures, and Functors.* Academic Press, 1975.

[Bekič 69] Bekič, H. Definable Operators in General Algebras, and the Theory of Automata and Flowcharts. *Machine Intelligence* 5, Edinburgh, 1969.
also in
Bekič, H. *Programming Languages and Their Definition.* Selected papers edited by C. B. Jones. Lecture Notes in Computer Science, vol. 177, Springer-Verlag, 1984.

[Bergstra&Klop 84] Bergstra, J. A. and J. W. Klop. Process Algebra for Synchronous Communication. *Information and Control*, 60, 1–3, 1984.

[Berry et al 85] Berry, G, Currien, P.-L. and J.-J. Lévy. Full Abstraction for Sequential Languages: the State of the Art. In *Algebraic Methods in Semantics*, Cambridge University Press, 1985.

[Boudol&Castellani 87] Boudol, G. and I. Castellani. On the Semantics of Concurrency: Partial Orders and Transition Systems. *Proc TAPSOFT'87*, Lecture Notes in Computer Science 249, Springer-Verlag, 1987

[Brookes&Rounds 83] Brookes, S. D. and W. C. Rounds. Behavioural Equivalence Relations Induced by Programming Logics. *Proc. 10th ICALP*, Lecture Notes in Computer Science, vol. 154, Springer-Verlag, 1983.

[Burmeister 82] Burmeister, P. Partial algebras – survey of a unifying approach towards a two-valued model theory for partial algebras. *Algebra Universalis*, 15, pp. 306–358, 1982.

[Castellani&Hennessy 87] Castellani, I. and M. Hennessy. Distributed Bisimulations. Sussex University, Report 5/87, 1987.

[Degano et al 87] Degano, P. De Nicola, R. and U. Montanari. Observational Equivalence for Concurrency Models. In *Formal Description of Programming Concepts III*, Lecture Notes in Computer Science, Springer-Verlag, 1987.

[Degano et al 87a] Degano, P. De Nicola, R. and U. Montanari. A Distributed Operational Semantics for *CCS* Based on Condition/Event Systems. CNR IEI Pisa, Nota Interna B4-21, 1987.

[De Nicola&Hennessy 84] De Nicola, R. and M. Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, 34, 1984.

[Gentzen 34] Gentzen G. Investigations into natural deduction, in *The Collected Work of Gerhard Gentzen,* ed., M. Szabo. North-Holland, 1969.

[Goldblatt 79] Goldblatt, R. I. *Topoi.* North-Holland, 1979.

[Goltz&Mycroft 84] Goltz, U. and A. Mycroft. On the Relationship of CCS and Petri Nets. *Proc. 11th ICALP*, Lecture Notes in Computer Science, vol. 172, Springer-Verlag, 1984.

[Gordon 79] Gordon, M. J. C. *The Denotational Description of Programming Languages.* Springer-Verlag, 1979.

[Grätzer 79] Grätzer, G. *Universal Algebra.* Springer-Verlag, 1979.

[Guessarian 81] Guessarian, I. *Algebraic Semantics.* Lecture Notes in Computer Science, vol. 99, Springer-Verlag, 1981.

[Hennessy 83] Hennessy, M. Synchronous and Asynchronous Experiments on Processes. *Information and Control* 59, 1–3, 1983.

[Hennessy 85] Hennessy, M. Acceptance Trees. *Journal of the ACM*, 32, 4, 1985.

[Hennessy 85/86] Hennessy, M. An Algebraic Theory of Processes. Part One and Two. Dept. of Computer Sc. Lecture Notes, University of Århus, Denmark, DAIMI FN-19, 1981.
also to appear as a book in MIT Press.

[Herrlich&Strecker 73] Herrlich, H., and G. E. Strecker. *Category Theory*. Allyn and Bacon, 1973.

[Hoare 78] Hoare, C. A. R. Communicating Sequential Processes. *Communications of the ACM*, 21(8), 1978.

[Hoare 85] Hoare, C. A. R. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[Hoare et. al. 84] Hoare, C. A. R., Brookes, S. D., and A. W. Roscoe. A Theory of Communicating Processes. *Journal of the ACM*, 1984.

[Hyland et al 87] Hyland, J. M. E, Robinson, E. P. and G. Rosolini The Discrete Objects In the Effective Topos. To appear.

[Keller 76] Keller, R. Formal Verification of Parallel Programs. *Communications of the ACM*, vol.17, 7, 1976.

[Kunen 80] Kunen, K. *Set Theory*. North Holland, 1980.

[Lehmann 76] Lehmann, D. *Categories for fixpoint semantics*. Thesis. University of Warwick, Theory of Comp. Report 15, 1976.

[Lehmann 76a] Lehmann, D. Categories for fixed point semantics. *Proc. 17th IEEE Symposium on Foundations of Computer Science*, 1976.

[Lehmann&Smyth 81] Lehmann, D. and M. Smyth. Algebraic specification of data types: a synthetic approach. *Mathematical Systems Theory*, 14, 1981.

[Mac Lane 71] Mac Lane, S. *Categories for the Working Mathematician*. Springer-Verlag, 1971.

[Mazurkiewicz 77] Mazurkiewicz, A. Concurrent Program Schemes and Their Interpretations. Dept. of Computer Sc. Technical Rep., University of Århus, Denmark, DAIMI PB–78, 1977.

[Mazurkiewicz 84] Mazurkiewicz, A. Traces, Histories, Graphs: Instances of a Process Monoid. Lecture Notes in Computer Science, vol. 176, Springer-Verlag, 1984.

[Milner 75] Milner, R. Processes: a mathematical model of computing agents. *Proc. Logic Colloquium '73*, North Holland, 1975.

[Milner 76] Milner, R. Program Semantics and Mechanised Proof. Mathematical Centre Tracts No 82, Amsterdam, 1976, pp.3–44.

[Milner 80] Milner, R. *A Calculus of Communicating Systems.* Lecture Notes in Computer Science, vol. 92, Springer-Verlag, 1980.

[Milner 83] Milner, R. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science* 25, 1983.

[Nielsen et al 81] Nielsen, M, G. Plotkin and G. Winskel Petri Nets, Event Structures and Domains, Part I. *Theoretical Computer Science* 13, 1981.

[Olderog 87] Olderog, E.-R. Operational Petri Net Semantics for *CCSP*. In [APN'87].

[Park 81] Park, D. Concurrency and Automata on Infinite Sequences. *Proc. 5th GI Conf. on Theoret. Comp. Science*, Lecture Notes in Computer Science 104, Springer-Verlag, 1981.

[Petri 62] Petri, C. A. Fundamentals of a Theory of Asynchronous Information Flow. *Proc. IFIP'62*, North Holland, 1962.

[Plotkin 81] A structural approach to operational semantics. Dept. of Computer Sc. Lecture Notes, University of Århus, Denmark, DAIMI FN-19, 1981.

[Plotkin 82] A powerdomain for countable nondeterminism. *Proc. 9th ICALP*, Lecture Notes in Computer Science, vol. 140, Springer-Verlag, 1982.

[Prawitz 65] Prawitz, D. *Natural Deduction.* Almqvist & Wiksell, Stockholm, 1965.

[Reisig 85] Reisig, W. *Petri Nets.* EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1985.

[Scott 76] Scott D. Data types as lattices. *SIAM Journal on Computing* 5, 3, 1976.

[Shields 85] Shields, M. W. Concurrent Machines. *The Computer Journal* 28, 5, 1985.

[Smyth&Plotkin 82] Smyth, M. and G. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing* 11, 4, 1982.

[Stoughton 86] Stoughton, A. *Fully Abstract Models of Programming Languages.* Thesis, University of Edinburgh CST – 40 – 86, November 1986.

also, to appear in Pittman series Research Notes in Theoretical Computer Science.

[Stoughton ??] Stoughton, A. Substitution Revisited. *Theoretical Computer Science*, to appear.

[Stoy 77] Stoy, J. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* The MIT Press, 1977.

[Tarski 55] Tarski, A. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5, 1955.

[Winskel 82] Winskel, G. Event Structures Semantics for CCS and Related Languages. Lecture Notes in Computer Science 140, Springer-Verlag, 1982.
also
Dept. of Computer Sc. Report, University of Århus, Denmark, DAIMI PB-159, 1983.

[Winskel 83] Winskel, G. A representation of completely distributive algebraic lattices. Tech., Report, Carnegie-Mellon University, 1983.

[Winskel 84] Winskel, G. A new definition of morphism on Petri Nets. Lecture Notes in Computer Science 166, Springer-Verlag, 1984.

[Winskel 84a] Winskel, G. Categories of models for concurrency. *Proc. Workshop on the Semantics of Concurrency, Pittsburgh 1984,* Lecture Notes in Computer Science 197, Springer-Verlag, 1984;
also
Technical Report, Computer Laboratory, University of Cambridge, 1984.

[Winskel 84b] Winskel, G. Synchronisation Trees. *Theoretical Computer Science* 34, 1984.

[Winskel 87] Winskel, G. Petri Nets, Algebras, Morphisms and Compositionality. *Information and Computation*, 72, 1987.

[Yech 78] Yech, T. *Set Theory.* Academic Press, 1978.