ALGEBRAIC REASONING IN LAMBDA CALCULI

by

Albert

Edmund A Kazmierczak, BSc (Hons)

Department of Computer Science

Submitted in fulfilment of the requirements

for the Degree of

Doctor of Philosophy

UNIVERSITY OF TASMANIA

HOBART

June 1991

This thesis contains no material which has been accepted for the award of any other higher degree or graduate diploma in any tertiary institution. To the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference has been made in the text of the thesis.

E. Kazmierczak

Abstract

This dissertation examines some aspects of the relationship between λ calculus and universal algebra. Motivated by the desire to understand the implementation of abstract data types in functional languages we use λ algebras, or models of the pure $\lambda\beta$ calculus, as a universe in which models for specifications are to be constructed. Given an arbitrary λ algebra $\mathcal{M}$ we construct from it a cartesian closed category with sum objects which we call $\mathcal{C}(\mathcal{M})$. The objects of $\mathcal{C}(\mathcal{M})$ are interpreted as *types* while the arrows can be thought of as *effective functions.* Unlike set theory where the semantics of an abstract data type is often chosen to be the (isomorphism) class of initial models the simple model of types and functions presented here contains no initial object. This means more care is required when choosing the semantics of an abstract data type. In particular the category $\mathcal{C}(\mathcal{M})$, which is an extension of the *cartesian closed monoids* of [Koy82, LS86, Bar84] is cartesian closed (4.1.6) and so the basic properties of functional languages such as currying and higher order functions can be modelled.

Using the idea of a T-algebra [LS81, SP82] we construct algebras as the greatest fixed points of endofunctors over $\mathcal{C}(\mathcal{M})$ (4.2.5). While in general systems of equations do not always exhibit computational properties in many cases they do, especially if a set of Church-Rosser and strongly normalising rewrite rules can be extracted from the equations by a technique such as the Knuth-Bendix completion procedure [KB70]. It is known [KS81] that a Church-Rosser strongly normalising set of rewrite rules is associated with the word problem in the free algebra and if the word problem is solvable then there is a recursive function taking each term in the free algebra to its normal form. We construct models for systems of Church-Rosser, strongly normalising sets of rewrite rules based on this characterisation of the word problem (4.3.14).

Secondly a calculus, $\lambda^{\Sigma E}$, is given for reasoning about these models. $\lambda^{\Sigma E}$ is a simply typed λ calculus augmented with the operations and equations of the original specification. It is shown that this calculus is a conservative extension of the usual equational calculus if a strongly normalising and Church-Rosser set of rewrite rules can be generated from the original equations (5.2.10).

Thirdly, given a specification and an *arbitrary* model of that specification in $\mathcal{C}(\mathcal{M})$, the soundness of deduction in $\lambda^{\Sigma E}$ is proved (6.1.5). Finally, a theorem relating the equational theory to $\lambda^{\Sigma E}$, namely that for every model of the equational theory there exists a unique model of $\lambda^{\Sigma E}$.

Acknowledgements

There are many people to whom I owe thanks for both their help and their generosity of time and spirit.

Firstly to my supervisors, Phil Collier and Clem Baker-Finch. To Clem Baker-Finch for listening to and reading through many half baked ideas, setting me straight when I needed setting straight and for his patience with a sometimes "difficult" student, to Phil Collier for putting up with all my continuing requests of him even after I had moved to Edinburgh and to both for their encouragement and faith at all times. Also to the staff at the CS Dept. at the University of Tasmania for the many fruitful and enjoyable years spent there.

Secondly to the G.C.S.B, that is, Andrew Partridge, David Wright, Bernard Gunther, Ben Lian, Tony Dekker and James Coleman for their comradeship and for making my days at the University of Tasmania special. I would, however, especially like to thank David Wright for his assistance with the LaTeX'ing of parts of this dissertation, James Coleman for all his friendship and good humour (especially when I was without mine) and Tony Dekker for all the time we spent discussing our ideas, relevant or not.

Finally to my wife Thérèse for all her patience, understanding, encouragement and help, which perhaps went unthanked at the time but certainly not un-appreciated, and to my parents Helen and Walter for all their selfless help and support that they have given me throughout the many years.

Contents

List of Figures

Chapter 1

Introduction

The universal algebra approach to the design and specification of programs has led to a number of important and fundamental concepts in computer science. In this approach programs can be equated to algebras and program specifications with a set of axioms describing the behaviour of a program that is to solve a particular problem. Such a view allows for a rigorous treatment of both programs and specifications as well as the relation between them within one unified framework and this in turn has facilitated a number of other developments. Thus in software engineering one may begin the design of programs by first writing a specification of the various functions required to solve the problem, their domains and ranges and a set of axioms specifying their behaviour. Such a program specification may be explored for various logical properties such as consistency, or perhaps it may be shown to possess some property inherent to the problem domain and thus some measure of confidence in the specification is gained. From the specification one must now construct one particular algebra, that is, give a program meeting the specification.

Specification languages to aid the writing, exploring and understanding of large specifications have been one such development, for example, CLEAR [GB79], ACT ONE [EM85] and ACT TWO [EM90], as well as OBJ and OBJ3 [GW88]. A second development, which is incorporated into some specification languages, is that of formalising software engineering techniques within this algebraic framework, for example, one software engineering technique which has been formalised is *stepwise* *refinement.*

One aspect of the theory of algebraic specifications which remains unsatisfactory is the formal definition of *implementation* and how this should take into account an actual programming language. What is usual in the initial algebra approach [JAGW78, EM85] is to consider the actual language as a particular specification itself, say a collection of abstract data types as in [JAGW78]. With such an assumption one can define what it means for a program to meet its specification and so show that any program written in this (abstract) programming language will meet its specification. The final coding step, however, must still be done outside the algebraic framework.

Almost in parallel with the development of the theory of algebraic specifications came the development of higher order functional programming languages such as Miranda [Tur] and Standard ML [HMT90, MT90]. Functional programming languages offer a simpler alternative than imperative programming languages for studying the interactions between algebraic specifications and programs because their view of programs is much simpler. If we are to use a higher order functional language as a target language for the implementation of algebraic specifications and again consider the formalisations of software engineering ideas, such as stepwise refinement and *implementation*, then we require that the latter take into account the fact that a program may use higher order functions to realize a (potentially first order) specification.

Functional languages are still too complex[1] to deal with directly and so we choose to simplify the task by considering λ calculus as a simple functional programming language. Viewed another way, λ calculus is a simple abstraction for higher order functional programming languages and thus, we have the basis for studying in more detail the interaction between algebraic specifications and functional programming languages.

We wish to understand the logical relationships between equational specifications and functional programming languages better for two main reasons: (1) such

[1]Consider for example, type polymorphism, exceptions and the possibility of assignment in Standard ML [HMT90].

an understanding may allow us to improve the facilities offered by functional programming languages , for example, to allow mixtures of executable specification and the more usual style of higher order function definition within one language; (2) if the relation between implementation and language can be formalised then we may perform proofs of correctness of the program code against its specification within one unified framework. We are not yet at such a point and still must rely upon a suitable abstraction of programming languages . In the case of higher order functional languages this abstraction can be made more precise than the (simpler) assumptions usually found in definitions of implementations of equational specifications[2].

In this dissertation we examine the problem of constructing implementations of equational specifications[3] in λ calculus. In particular we will be concerned with two questions:

1. under what conditions can algebras in the λ calculus be constructed;
2. is there a sound and complete inference system for reasoning about algebras in the λ calculus.

An answer to the first question would enable two things: (1) it would allow us to devise a test to check if a specification is realizable (in the λ calculus) and (2) it would allow an extension of the usual function definition facilities in functional languages to include abstract data types. An answer to the second question would enable us to develop a theory of an implementation in a functional language. To answer these questions we view the λ calculus as a *universe* in which abstract implementations may be made.

Rewrite rules are a means of computing with equations [HO80, O'D77] and since we wish to construct algebras within the λ calculus their operations will, in some sense, be computable. For the λ calculus this means (β) reduction, that is, the value of an expression is computed by trying to reduce it to a (β) normal form. For rewrite rules the same ideas apply and we show how (β) reduction can

[2]see section 2.1
[3]For a more proof theoretic view of implementation between sets of rewrite rules see [KS85].

be used to "simulate" the result of rewriting a term to normal form. If (Σ, E) is a signature and a set of Σ equations then we show that under the assumptions that each sort in S is *non-void* and that a set of Church-Rosser rewrite rules can be obtained from the equations in E then an algebra may be constructed from (Σ, E).

If A is an algebra in the λ calculus satisfying a set of first order equations then the theory of A may well contain equations which equate terms involving higher order subterms with first order terms. We show that a simply typed λ calculus extending the algebraic theory together with the usual rules (α) and (β) as well as the rules of deduction (ζ) and (ξ) [Bar84] and some (δ) rules for surjective pairing is sound for such algebras. We also examine the possibility of some extensions to this calculus and show that it gives rise to a cartesian closed category, in the manner of [LS86], which is the free cartesian closed category on the algebraic theory presented by (Σ, E).

1.1 Background and Related Work

Since the papers of [JAGW78] the concepts of *data refinement* and *abstract implementations* have figured quite prominently in algebraic specifications. Given an abstract specification of a data type, or program, the problem was to choose a representation for the data and code for the functions in actual programming language so that all the abstract properties of these functions were preserved. Within each of the major schools of thought in the semantics of algebraic specifications these two concepts have been studied.

For those researchers using initial algebra semantics [TJWB77, JAGW78, Zil79, EM85, EM90] there are a number of definitions of *implementation.* For example, in the work of the *ADJ* group [JAGW78] (but see also [Nou80, Nou83]) an implementation is a *derivor.* It is assumed that a programming language or machine gives rise to a particular algebra in which all the implementations will be made and in this case a derivor is a map from the (abstract) specification[4] signature

[4]A brief word on nomenclature is required here. We say that a specification or data type is

into the set of derived operations of this (concrete) algebra. The implementation is correct if the image of the derivor is a isomorphic to the initial algebra of the abstract specification. In the work of [HEP80b, HEP80a, EK83, EM85, EM90] the concrete data types are characterised by an equational presentation with initial algebra semantics. An implementation is then an extension of the signature of the concrete specification by the sorts and operations of the abstract signature together with some equations relating the "new" operations back to the original ones (called an *implementation specification*). Using this framework the concept of program development by stepwise refinement can be formalised. Perhaps the distinguishing feature of this work, from that of [JAGW78] is that specifications are not considered separately from their semantics and so abstract implementations also have a semantics. This is expressed as the composition of two functors: *synthesis* and *restriction. Synthesis* takes each algebra of the concrete specification to a model of the implementation specification while *restriction* essentially forgets the sorts and operations of the concrete data type and restricts the resulting models to those reachable from the initial algebra of the abstract specification.

In [Ehr78, Ehr82] the focus is purely on the syntactic aspects of implementation. This is done purely in terms of theories, where the notion of implementation is defined by specific morphisms between theories. The implementation of an abstract specification in (the theory of) a concrete specification is an extension of the latter by the operations of the abstract specification together with equations which relate the new operations back to the operations of the concrete specification (as in [HEP80b]). The difference between the syntactic approach of Ehrich and that of [HEP80b] is that the conditions for correctness are also formulated in a purely syntactic manner and aim at defining a class of *permissible* implementations by ensuring that each of the theories, that of the abstract specification and the concrete specification, is preserved in the theory defining the implementation.

In [ST86] a different perspective on implementations is given. It is assumed that specifications are *loose* [SW81], that is, the semantics of a specification is not

concrete if it is a specification of a data type in our actual programming language and does not need to be refined further before it can be implemented, otherwise we say that it is *abstract*.

simply confined to the initial models as in the work above. The assumption is made that programs are to be modelled as algebras and under this assumption a specification implements another specification if each model of the former is also a model of the latter. How does this differ from the related notions above? In the work of [JAGW78, Nou80] an implementation takes the theory of an abstract specification into that of one more closely related to an actual programming language. In the framework of [SW81, ST86] abstract implementations make design choices by simply rejecting certain programs.

Other notions of implementation also exist, for example, [KA84, Hup80, BV85, Foo85, GM82] and particularly [PV85]. This last notion of implementation is of interest because of the view of programming language that is taken.

In [PV85] a programming language Λ is defined in which the basic type structure of the language may include solutions of recursive equations between the sorts which are, in effect, domain equations. Thus we may include solutions to recursive equations like:

$$\begin{aligned} List &\cong 1 + Nat \times List \\ Tree &\cong Bool + Tree \times Tree \end{aligned}$$

over the basics sorts *Nat* and *Bool*. The language Λ of [PV85] also includes higher order functions. If we are given an (abstract) specification the implementation in Λ is given by a pair of maps taking each sort of the specification to a *type* in the language and each operation in the specification to term of the appropriate arity in Λ which satisfies the equations. The language Λ of [PV85] is a simple functional programming language in which data type definitions may be defined by induction.

In all these approaches to implementation certain assumptions are made regarding the best way of representing the programming language. In [JAGW78] it is assumed that the programming language forms an algebra. The implementation thus becomes a choice of derived operations within this algebra for each of the operations demanded by the abstract specification. In the work of [HEP80b, HEP80a, EK83, EM85, EM90] a particular programming language is considered to be a collection of abstract data types, perhaps organised into a hierarchy. Implementations are consequently translations between operations specified in one abstract data type (the abstract specification) and those of another (the concrete data types of the language).

While these approaches can in theory be extended to higher order functional languages that has not been not explicitly. Higher order specifications have been investigated [PG81, Poi86] but it is not yet clear how the theory of initial algebra semantics that these two approaches rely on extends to the higher order case.

If the purely syntactic approach is adopted as in [Ehr78, Ehr82] then for higher order functional programming languages this requires a notion of higher order theories in which implementations may be made. This we feel, is a possible avenue and it is one of our goals in this dissertation to introduce such theories. Where we differ from a purely syntactic approach is perhaps the assumption that a higher order theory is actually itself a specification of a program while the program is a model of that theory.

The approach of [ST86] has the advantage of being applicable to algebras of a very general nature and so naturally includes higher order algebras within the general framework. Despite this there is still some benefit to be gained from studying the way in which programs relate back to their specifications. One benefit which stems from a precise formulation of the relation between (functional) programs and their specifications is that the ability to reason about specifications, programs and data refinement is possible within a single framework.

The work on λ calculus has unearthed many interesting paradigms for functional programming. Investigations into the semantics of λ calculus, for example [Sco71, Sco76, Sto77, Wad76, Mey82], has grown into the theory of domains which is now a well understood theory of the models of both the pure and typed λ calculi. Our interest in λ calculus stems from the fact that it can be used to investigate many properties of functional programming languages without the overhead of having to consider a lot of inessential operational detail. For example, in the sphere of functional programming domains are used to reason about the behaviour of higher order functions [Sto77, Bar84], define suitable notions of ***data type*** [Sco76, LS81, SP82] and understand type polymorphism [Mil77, BM84, CW85, MPS86, Rey85, Col87].

An early formalisation of data types in domain theory, and particularly in the λ calculus, can be found in [Sco76] further elaborated in [Sco80]. In this simple theory a given model of the pure λ calculus always contains a single "type" U which plays the role of the universe of all "types", that is U is a type and all other types are subtypes of U. A parallel may be seen in some set theories where each member of a set is drawn from a universe, but the analogy is only a weak one since the simple models of types that we consider here have only some of the properties of sets.

An important feature of the domains discussed above is that they contain solutions to recursive domain equations. Indeed there is an alternative characterisation of the algebras within a domain which arises from the category theoretic approach to algebra [LS81, SP82, MA86]. Algebras do not have to be specified axiomatically, as is done in universal algebra, but can be specified as the solutions to recursive domain equations involving functors. This necessitates the view that the data types which can be defined in a domain form a category preferably with a certain structure which allows the solution of recursive domain equations[5].

In a simple type theory two essential features thus co-exist: (1) the ability to define data by solving recursive domain equations (provided that the category of "types" has enough structure to do this) and (2) the denotations of recursive functions (because domains are simply models of the λ calculus in which the number theoretic recursive functions may be represented [JRHS72, Cut80, Bar84]). If there exists a domain in which types and functions can be constructed in a purely syntactic way, as in [Sco80, Koy82], then that domain would be a very suitable universe in which to study implementations. The reason is that programs would be actual syntactic entities which can be treated operationally as well as being the elements of an algebra satisfying the algebraic specification.

[5]In the context of this discussion it is perhaps better to think of these as recursive *type* equations

To construct an implementation in such a universe we need to do two things: (1) give a *type* for each sort which will be the carrier of that sort and (2) give a (λ definable) function for each (abstract) operation in the signature. Since λ calculus, and functional programming languages, are both languages for expressing computations we conjecture (and later show) that if the equations in an equational specification can be turned into a suitable set of rewrite rules then a special algebra may be constructed in models of the λ calculus (see [Bar84, Mey82]).

Since the implementation is essentially an algebra within the λ calculus we may have equations between functions definable in the λ calculus and those of the implementation which are not deducible using just equational reasoning, for example, in the theory of lists and natural numbers we may have the equation

$$\mathit{fold}\, 0 \,+\, [1,2,3] = 6$$

where "fold" for lists of natural numbers is defined by the usual recursion equations:

$$\begin{aligned} \mathit{fold}\; x\; f\; \mathit{nil} &= x \\ \mathit{fold}\; x\; f\; \mathit{cons}(y, ys) &= f(y, \mathit{fold}\; x\; f\; ys) \end{aligned}$$

It is important therefore to provide a system of reasoning about both the model and the specification in order to be able to prove facts about the algebras in the λ calculus and to verify that specifications are met. One such a system has been proposed in [PG81] based upon the equational calculus of [Bir35] (see also [GM81, EM85]). We adopt the approach of combining simply typed λ with algebraic operations because we feel that for λ calculus it is a more natural approach. The same approach has been taken in [BT88] where λ calculus is combined with equations between terms over some signature. We compare our results with [BT88] in more detail at the end of chapter 5 where such a comparison is perhaps more relevant.

1.2 Overview

This dissertation is organised as follows.

In chapter 2 we review some universal algebra and some of the theory of term rewriting. We also discuss the transition from the usual proof theoretic notions of term rewriting to the equivalent concept for algebras. The remainder of this chapter then goes on to deal with algebraic theories and the construction of algebras as *fixed points* of certain polynomial functors [Mac71]. The aim here is to lay the algebraic foundations for the work in the sequel. In chapter 3 we outline the correspondence between λ calculus and *cartesian closed categories* by reviewing firstly λ algebras and models, and secondly the construction of free cartesian closed categories over a simply typed λ calculus. λ algebras are to be the *universes* in which algebras are constructed while the use of cartesian closed categories will be to show two particular theorems about the calculus $\lambda^{\Sigma E}$ which we consider in chapter 5.

In chapter 4 we give a category which is our abstraction of a programming language and then go on to examine the structure of algebras within this category. In section 4.1 we define the category $\mathcal{C}(\mathcal{M})$ and examine some of its structure. In particular it is a simple extension of the categories of [Koy82], cartesian closed (theorem 4.1.6) but not bicartesian closed [LS86], but essentially there is enough structure to deal with the inductively defined data types which we are considering. In section 4.2 we look at the structure of anarchic algebras in $\mathcal{C}(\mathcal{M})$ where operations are given by the obvious injections (4.2.5). In section 4.3 we look at the construction of algebras based on confluent and terminating sets of rewrite rules. Given a set of such rewrite rules we can construct their solution as a recursive function (4.3.4 but see also [KS81]). The free algebra together with this solution constitutes an algebra in the sense of 2.3.1 (see 4.3.14). In section 4.4 we look at some examples while in 4.5 we summarise the chapter.

Chapter 5 deals with extending algebraic theories to cartesian closed categories, or when dealing with internal languages, embedding free algebras in higher order logic. Fix a presentation (Σ, E). The plan is to first define a λ calculus generated by the free Σ algebra on variables X, $\lambda_{\Sigma E}$. This is done in section 5.1. We explicitly include products of terms with surjective pairing and projections from the consideration of the *clone* of (Σ, E). The calculus which is thus derived can be considered as a weak axiomatisation of the models of (Σ, E) in $\mathcal{C}(\mathcal{M})$. In section 5.2 we look at some of the reduction properties of the calculus defined in section 1 and via reduction show that the extension in section 1 is conservative (theorem 5.2.10).

In chapter 6 we return to semantics again and show that the calculus $\lambda^{\Sigma E}$ of chapter 5 is *sound* with respect to any algebra in $\mathcal{C}(\mathcal{M})$ of a presentation (Σ, E). This is done by extending the proof that any cartesian closed category is a λ algebra given in [Koy82, Bar84]. We give a cartesian closed category $\mathcal{C}_{\Sigma,E}$ which is obtained by the correspondence between λ calculus and cartesian closed categories [LS86] and show that it is the *free* cartesian closed category on an algebraic theory $\mathcal{T}_{\Sigma,E}$. Also we show that the canonical $\mathcal{T}_{\Sigma,E}$ algebra in $\mathcal{C}(\mathcal{M})$ can be uniquely extended to a $\mathcal{C}_{\Sigma,E}$ algebra and that the construction is general enough so that any $\mathcal{T}_{\Sigma,E}$ can be extended in this way.

In chapter 7 we conclude and give some directions for future research.

Chapter 2

Algebraic Preliminaries

If the assumption regarding programs as algebras and program specifications as presentations of equational theories is adopted then for a program to satisfy its specification a program must provide at least those functions and types required by the signature of the specification. Naturally the operations making up the program must satisfy the axioms but if the axioms themselves can be used for computation, say as a set of rewrite rules, then the axioms may be used to specify computations as well. All that is needed is a means of choosing, or constructing, the data types over which we are to compute. This has been done in the case of rewrite rules [O'D77, HO82] where the value of a computation is simply the normal form of the term specifying that computation. Assume now that the axioms are simple equations. If the equations are in the form of recursion equations then it is easy to see what the "values" of computations should be, but if they are true equations then it is not so obvious.

The problem of finding a decision procedure to test the equivalence of two terms in an algebra is called the *word problem* for that algebra and the word problem for free algebras has well known connections with recursive function theory, see for example [KS81, JABW81]. In the first part of this chapter we review some results in this area before exploiting this connection further in chapter 4. We then give the category theoretic perspective on algebraic theories and their algebras which we use later in chapter 6.

2.1 Deduction, Computation and the Word Problem

We begin by a brief review of some universal algebra. The reader is referred to [Coh81, MB67, Bou86] for further details regarding universal algebra, general algebra and the set theory, and to [JAGW78, EM85, ST89] for the use of these in algebraic specifications.

Let S be a set of *sorts*, S^* be the set of strings formed from elements of S and Ω an $S^* \times S$ indexed family of *operation symbols* (called an *S sorted operator domain* in [JAGW78]). A *Signature* Σ is a pair (S, Ω) where Ω is an S sorted operator domain. In the sequel we will write $\Sigma_{\omega,s}$ for the set of operation symbols $\Omega_{\omega,s}$ in $\Sigma = (S, \Omega)$ and ε to denote the empty string.

Definition 2.1.1 ([GM87]) *A signature Σ is called* non-void *if for every $s \in S$ either $\Sigma_{\varepsilon,s}$ is not empty or there is a function symbol $\sigma : s_1 \times \ldots \times s_n \to s$ such that every s_i, $1 \leq i \leq n$, is non-void. Otherwise the sort is called* void.

A Σ algebra is an S indexed family of sets $\{A_s\}_{s \in S}$ together with an $S^* \times S$ indexed family of maps:

$$\gamma_{\omega s} : \Sigma_{\omega s} \to A_s^{A^\omega}$$

interpreting each operator symbol in Σ as an actual operation in A. We use the pair (A, γ) to denote the algebra with carrier A and operations defined by γ. Here and in the sequel we use A^ω where $\omega = s_1, \ldots, s_n$ to denote the product of sets $A_{s_1} \times \ldots \times A_{s_n}$. Also we write σ_A as a shorthand for $\gamma_{\omega s}(\sigma)$.
One algebra of particular interest is the free algebra generated by a set of variables X which is defined as follows.

Definition 2.1.2 ([JAGW78]) *The set of Σ terms $T_\Sigma(X)$ generated by a set of variables X is the least set inductively defined by the following rules:*

1. $X \subseteq T_\Sigma(X)$

2. $\Sigma_{\varepsilon s} \subseteq T_{\Sigma s}(X)$

$$\begin{tikzcd}X \arrow[r, "\eta"] \arrow[dr, "\rho"'] & T_\Sigma(X) \arrow[d, "\overline{\rho}"] \\ & A\end{tikzcd}$$

Figure 1: The Universal Property of $T_\Sigma(X)$

3. if $\omega = s_1 \dots s_n$, $\sigma \in \Sigma_{\omega s}$ and $t_i \in T_{\Sigma s_i}(X)$ for all $1 \leq i \leq n$, then

$$\sigma(t_1, \dots, t_n) \in T_\Sigma(X)$$

The Σ algebra structure on $T_\Sigma(X)$ is given by interpreting each $\sigma \in \Sigma_{\omega s}$ as the function given by

$$\sigma_T(\langle t_1, \dots, t_n \rangle) = \sigma(t_1, \dots, t_n)$$

If the set of generators X is empty then definition 2.1.2 defines another Σ algebra which, if it is non-trivial, is called the Σ *Word Algebra* and consists simply of the those terms built from constants and function symbols only. If Σ is nonvoid then $\mathrm{T}_\Sigma(\emptyset)_s$ is non-empty for each sort s.

Let A be a Σ algebra and X a set of variables. An *assignment* of values in A to variables in X is a map $\rho : X \rightarrow A$. ρ can be extended to a homomorphism $\overline{\rho}$ which interprets every term of $T_\Sigma(X)$ in A as in figure 1. Moreover this extension is unique [JAGW78, EM85].

A Σ equation is a triple (X, t_1, t_2), where X is a set of variables and $t_1, t_2 \in T_\Sigma(X)$ with $var(t_1) \subseteq X$ and $var(t_2) \subseteq X$. The idea is that each equation (X, t_1, t_2) is universally quantified over the variables in X. This is often written as $\forall X.t_1 = t_2$ to make this quantification explicit. An algebra A *satisfies* the equation $\forall X.t_1 = t_2$ iff for every assignment $\rho : X \rightarrow A$,

$$\overline{\rho}(t_1) = \overline{\rho}(t_2)$$

where again $\overline{\rho}$ is the unique homomorphism extending ρ. This is written as $A \models \forall X.t_1 = t_2$ and we say that $\forall X.t_1 = t_2$ is *Valid* in A. An algebra A satisfies a set of equations if and only if it satisfies every equation in the set.

Let E be a set of equations and $Mod[E]$ the set of all Σ algebras which satisfy E. Birkhoff's now familiar equational calculus [Bir35] was originally proposed to derive equations which will be valid in every algebra of $Mod[E]$. The original equational calculus of [Bir35] was intended for the single sorted case but when this calculus is naively generalised to the many sorted case the problem of unsound deduction occurs [GM81] (but see also [GM86, GM87]). Below we present a modified equational calculus due to [EM85].

Definition 2.1.3 ([EM85]) *Given a set of Σ equations E, a* Many Sorted Equational Calculus *is given by the following deduction rules:*
Substitution

$$\frac{\forall X,\ t_1 = t_2}{\forall X \bigcup Y,\ \overline{h}(t_1) = \overline{h}(t_2)}$$

where $h : X \to T_\Sigma(Y)$ and $\overline{h}$ is the homomorphic extension of h to $T_\Sigma(X)$.
Replacement

$$\frac{\forall X,\ t_1 = t_2}{\forall X \bigcup Y,\ \overline{h_1}(t) = \overline{h_2}(t)}$$

where $t \in T_\Sigma(X)$ and $h_i : X \to T_\Sigma(Y)$ such that for some $x_0 \in X$,

$$h_i(x) = \begin{cases} t_i & \textit{if } x = x_0 \\ x & \textit{otherwise} \end{cases}$$

Variables *if $X_0 \subseteq X$*

$$\frac{\forall X,\ t_1 = t_2}{\forall X_0,\ t_1 = t_2}$$

provided that

1. $t_1, t_2 \in T_\Sigma(X_0)$

2. $\forall s \in S, \quad X_s \neq \emptyset \quad$ *implies* $\quad T_\Sigma(X_0) \neq \emptyset$

and also Transitivity, Reflexivity *and* Symmetry[1].

[1] *There are alternatives to this particular set of rules which are also sound and complete e.g [GM81]*

Definition 2.1.4 *A* Derivation *of a theorem e from a set of Σ equations E is a sequence $e_1, \ldots, e_n$ such that $e_1 \in E$ and $e_n = e$ and each e_i is obtainable from e_{i-1} by application of one of the rules in 2.1.3 or $e_i \in E$. We use $E \vdash_E e$ to say that e follows from E by such a derivation and denote by E^*, the set of all equations e such that $E \vdash_E e$.*

Definition 2.1.5 ([EM85]) *Let E be a set of Σ equations. The Σ congruence on an algebra A generated by E, which we denote by $\equiv_E$, is defined as follows:*

1. *if $\forall X. L = R \in E$ is an equation then for all assignments $h : X \to A$,*

$$\overline{h}(L) \equiv_E \overline{h}(R)$$

   *(*Closure under substitution*).*

2. *$\forall a \in A.\ a \equiv_E a$*

3. *if $a \equiv_E b$ then $b \equiv_E a$*

4. *if $a \equiv_E b$ and $b \equiv_E c$ then $a \equiv_E c$*

5. *for each operation symbol $\sigma : s_1 \times \ldots \times s_n \to s$ of Σ, if a_i and b_i are elements of sort s_i and $a_i \equiv_E b_i$ for each $1 \leq i \leq n$ then*

$$\sigma(a_1, \ldots, a_n) \equiv \sigma(b_1, \ldots, b_n)$$

   *(*Compatibility*).*

A congruence relation on a Σ algebra A is an equivalence relation on A which is also compatible (as in 2.1.5(5)) with the Σ structure. Σ congruences also arise in another manner. The *Kernel* of a homomorphism $f : A \to B$ is the set $\{(a, a') \mid f(a) = f(a')\}$ which is given by the pullback diagram

$$\begin{tikzcd}
Ker\ f \arrow[r] \arrow[d] & A \arrow[d, "f"] \\
A \arrow[r, "f"'] & B
\end{tikzcd}$$

If A is a Σ algebra and $f : T_\Sigma(X) \rightarrow A$ then *ker* f will be a Σ congruence on $T_\Sigma(X)$ [Coh81]. The smallest such congruence on $T_\Sigma(X)$ is the Σ congruence generated by E in definition 2.1.5 [Coh81].

Definition 2.1.6 ([EM85, Coh81]) *Given an Σ algebra A and congruence relation q the* Quotient *of A by q, denoted by A/q, is defined as follows:*

1. *The carrier of sort s is $\{[a] \mid a \in A_s\}$ where $[a] = \{a' \mid (a, a') \in q\}$, that is, the q* equivalence *class of a.*

2. *For each operation symbol $\sigma \in \Sigma_{ws}$ the corresponding operation in A/q is given by*
$$\sigma_q(a_1, \ldots, a_n) = [\sigma_{A/q}(a_1, \ldots, a_n)]$$

The following theorem from [EM85] summarises the relationship between the equational theory, the congruence $\equiv_E$ and the free and quotient algebras.

Theorem 2.1.7 (Soundness and Completeness [EM85]) *Let (Σ, E) be a presentation, X an S indexed set of variables and t and t' two terms in $T_\Sigma(X)$. Then the following are equivalent;*

1. $E \vdash_E \forall X,\ t = t'$

2. $\forall X.\ t = t'$ *is valid in all algebras satisfying E.*

3. $\forall X.\ t = t'$ *is valid in $T_\Sigma(X)/ \equiv_E$*

4. $t \equiv_E t'$

Is there a way of deciding if an equation is valid in a particular algebra A? This question is often referred to as the *word problem* for A and can be restated as follows:

> Given terms t and t' and an algebra A, find an algorithm to decide if t and t' both denote the same value in A.

More formally:

Definition 2.1.8 ([Coh81], [HO80]) *The* word problem *for an algebra A is to find an algorithm to decide if $A \models e$ for some Σ equation e.*

In general the word problem is undecidable [Coh81], but from theorem 2.1.7 if there is an algorithm to solve the word problem for $T_\Sigma(X)$ then the relation $E \vdash_E e$ is also decidable. It can be shown [Coh81] that the problem of finding a solution to the word problem, for a given algebra A, can be reduced to that of finding a homomorphism from $T_\Sigma(X)$ to a special subalgebra of itself. This is essentially what we describe now.

Definition 2.1.9 *Let q be a Σ congruence on a Σ Algebra A, and $R \subseteq A$. If each $r \in R$ is in exactly one q congruence class such that $\bigcup_{r \in R}[r] = A$ then R is a* Transversal *for q in A.* [2]

If q is an equivalence relation and R a transversal for q in a Σ algebra A then there is a function

$$\tau : A \rightarrow R$$

which takes any element in A to its representative in R. Moreover R can be supplied with a Σ algebra structure as follows:

$$\sigma_R(r_1, \ldots, r_n) = \tau(\sigma_A(r_1, \ldots, r_n))$$

where $\sigma \in \Sigma_{ws}$ and each r_i is an element of A_{s_i}. If τ is recursive then this gives us a way of computing in A. We consider this for the case when A is the free Σ algebra $T_\Sigma(X)$.

The following can be found in [O'D77, HO80, Klo80].

Definition 2.1.10 *A set of rewrite rules over a signature Σ is a set of ordered pairs of terms*

$$\mathcal{B} = \{(t, t') \mid t, t' \in T_\Sigma(X) \; and \; var(t') \subseteq var(t)\}$$

where again $var(t)$ denotes the set of variables of a term t. The Reduction Relation *$\rightarrow_{\mathcal{B}}$ is the least set containing B and closed under the following two rules:*

[2] *If we let $\triangle$ denote the diagonal of A, i.e $\{(a,a) \mid a \in A\}$ then another characterisation can be given as follows : R is a transversal for q in A iff $q \bigcap R \times R = \triangle$*

1. Substitution*: if $t, t' \in T_\Sigma(X)$, $t \rightarrow_{\mathcal{B}} t'$ and $\varphi : X \rightarrow T_\Sigma(Y)$ is any substitution then $\overline{\varphi}(t) \rightarrow_{\mathcal{B}} \overline{\varphi}(t')$;*

2. Replacement*: if $t_1, t_2 \in T_\Sigma(X)$, $t_1 \rightarrow_{\mathcal{B}} t_2$, $t \in T_\Sigma(Y)$ and φ_1, φ_2 are two substitions such that for some $y_0 \in Y$, $\varphi_1(y_0) = t_1$ and $\varphi_2(y_0) = t_2$ and for all $y \neq y_0 \in Y$ $\varphi_1(y) = \varphi_2(y) = y$ then $\overline{\varphi_1}(t) \rightarrow_{\mathcal{B}} \overline{\varphi_2}(t)$.*

The transitive and reflexive closure of $\rightarrow_{\mathcal{B}}$ is denoted by $\rightarrow^*_{\mathcal{B}}$.

A *Redex* in t is a subterm t' such that for some rule $(t_1, t_2) \in \mathcal{B}$, and some substitution φ, $\varphi(t_1) = t'$. A term t is said to be in *Normal Form* if it contains no redexes. The reduction relation also gives rise to a pre-order relation on the set of Σ terms. Let $t \leq_{\mathcal{B}} t'$ if $t' \rightarrow^*_{\mathcal{B}} t$. By the substitution and replacement conditions it follows that if $t_1 \leq_{\mathcal{B}} t'_1$ and $t_2 \leq_{\mathcal{B}} t'_2$ then $t_1[t_2/x] \leq_{\mathcal{B}} t'_1[t'_2/x]$. The symmetric, transitive and reflexive closure of $\rightarrow_{\mathcal{B}}$ is called the *Equivalence Relation* generated by $\mathcal{B}$.

Definition 2.1.11 *1. $\mathcal{B}$ is* Confluent *if for any Σ term t such that $t \rightarrow^*_{\mathcal{B}} t_1$ and $t \rightarrow^*_{\mathcal{B}} t_2$ then there is a t' such that $t_1 \rightarrow^*_{\mathcal{B}} t'$ and $t_2 \rightarrow^*_{\mathcal{B}} t'$.*

2. $\mathcal{B}$ is Strongly Normalising *if for all terms in $T_\Sigma(X)$, there is no infinite sequence of reduction steps starting at t, i.e all terms reduce to normal form along any reduction path.*

It is a well known fact [HO80] that if a set of rewrite rules is Church-Rosser then it is also confluent.

We can now state a theorem, due to Newman [New42], which gives sufficient conditions for a set of rewrite rules to be confluent. Say that a set of rewrite rules is *Locally Confluent* if $t \rightarrow_{\mathcal{B}} t_1$ and $t \rightarrow_{\mathcal{B}} t_2$ then there is a t such that $t_1 \rightarrow^*_{\mathcal{B}} t'$ and $t_2 \rightarrow^*_{\mathcal{B}} t'$.

Theorem 2.1.12 (Newman) *If $\mathcal{B}$ is locally confluent and strongly normalising then $\mathcal{B}$ is also confluent.*

A fundamental result given in [KB70] for testing a set of rewrite rules for local confluence is the Knuth-Bendix theorem. From the point of view of computing with sets of rewrite rules confluence is important for the reason that normal forms constitute the results of computation and confluence implies that these results are unique.

For the sequel we let $t[u \leftarrow t']$ denote the term which results from t by replacing the subterm u by t'.

Definition 2.1.13 *Let $\lambda_1 \rightarrow \rho_1$ and $\lambda_2 \rightarrow \rho_2$ be two rewrite rules. Let u be a subterm of λ_1 and φ a substitution such that $u = \overline{\varphi(\lambda_2)}$. Then the* derived pair *resulting from the superposition of $\lambda_2 \rightarrow \rho_2$ on $\lambda_1 \rightarrow \rho_1$ is the pair*

$$\langle \overline{\varphi}(\lambda_1)[\overline{\varphi}(u) \leftarrow \overline{\varphi}(\rho_2)], \overline{\varphi}(\rho_1)\rangle$$

The theorem below is due to Knuth and Bendix [KB70] and coupled with the Newman theorem gives a test for the confluence of a set of rewrite rules.

Theorem 2.1.14 (Knuth - Bendix Theorem) *The set of rewrite rules $\mathcal{B}$ is locally confluent iff for every critical pair $\langle P, Q\rangle$ resulting from superpositions in $\mathcal{B}$, there exists a term R such that $P \rightarrow^*_{\mathcal{B}} R$ and $Q \rightarrow^*_{\mathcal{B}} R$.*

The Knuth Bendix algorithm derives a set of rewrite rules from a set of equations which solve the word problem for the quotient of the algebra $T_\Sigma(X)/\equiv_E$. To decide if an equation $t = t'$ is true in this algebra, both t and t' are rewritten to normal form and compared. Normal forms can also be considered as the results of computations performed using the rewrite rules and this leads us finally to the theorem of [KS81] linking properties of reduction to those of recursive functions:

Theorem 2.1.15 ([KS81]) *If E is a set of equations such that the word problem for $T_\Sigma(X)/\equiv_E$ is solvable, then the retraction τ is a partial recursive function.*

2.2 S Sorted Theories

In this section we review some results about algebraic theories. Algebraic theories were first proposed by F. W. Lawvere [Law63] to study classes of equational theories, just as we may use the presentation of equational theories to study classes of algebras. While our ultimate aim in chapters 4 and 5 is to study algebras in a particular cartesian closed category we start here by looking at the general theory. The bulk of the material of this section may be found in [LG85, JAG75, KR89, Law63] and is not original. We assume the definitions of category, functor, natural transformation, limit and colimit which may be found in standard texts like [Mac71, AM75, HS79].

Let S be a set of sorts. S may be viewed as a *discrete category* by letting the objects of S be the elements of S and the morphisms of S be defined as follows:

1. for any $s \in S$, $Hom(s,s) = id_s$
2. for any s, s' such that $s \neq s'$, $Hom(s,s') = \emptyset$.

The category of all (small) sets and functions is here called $\mathcal{SET}$ and so the functor category $\mathcal{SET}^S$ is the category of all S sorted sets and S indexed families of maps between them.

More precisely, if A and B are two functors in $\mathcal{SET}^S$ then a map between them is a natural transformation $h : A \dot{\rightarrow} B$ with components $h(s) : A(s) \dot{\rightarrow} B(s)$ for every $s \in S$. Note also that we use the notation $\dot{\rightarrow}$ for natural transformations. Since S is small then $\mathcal{SET}^S$ is a topos and so is both complete and cocomplete (see [Mac71, LS86]).

Definition 2.2.1 ([LG85]) *Let S be a set of sorts. An S sorted theory $\mathcal{T}$ is a category defined by the following data:*

1. *The objects of $\mathcal{T}$ are the strings in S^*.*
2. *Let $u = u_1 \ldots u_n$ and $w = w_1 \ldots w_m$ be any two objects of $\mathcal{T}$, that is, u and w are strings in S^*. An S sorted* operation *in $\mathcal{T}$ is a map $\sigma : s \rightarrow w$ for some $s \in S$. A tuple of S sorted operators is a map $\langle \sigma_1, \ldots, \sigma_n \rangle : u \rightarrow w$ in which each σ_i is a u_i sorted operation in $\mathcal{T}$ with codomain w. Thus the elements of $Hom(u,w)$ are tuples of S sorted operators with codomain w. For each w there are n distinguished morphisms $\pi^w_{w_i} : w_i \rightarrow w$ such that*

$$\langle \sigma_1, \ldots, \sigma_m \rangle \circ \pi^u_{u_i} = \sigma_i$$

We call the maps $\pi^w_{w_i}$ w-ary variables.

Each of the operations $\sigma : s \rightarrow w$ can be thought of as an operation of sort s whose arguments are of the sorts in w. It then follows that the maps in $Hom(s, \varepsilon)$ are the constants of sort s.

Example 2.2.2 ([LG85]) *Let S be any non-empty set. The theory T_0 consists simply of the w-ary variables closed under composition and tupling. If $u = u_1 \ldots u_n$ and $w = w_1 \ldots w_m$ then there is a morphism $u \rightarrow w$ if and only if every u_i occurring in u also occurs in w. T_0 also contains coproducts. If u and w are two objects of T_0 (strings in S^*) then their coproduct is given by the concatenation uw with injections $in_u = \langle \pi^{uw}_{u_1}, \ldots, \pi^{uw}_{u_n} \rangle$ and $in_w = \langle \pi^{uw}_{w_1}, \ldots, \pi^{uw}_{w_m} \rangle$. Every algebraic theory contains a copy of T_0 (see [LG85]).*

The algebraic theory presented by (Σ, E) is defined to be the free algebraic theory generated by an $S^* \times S$ indexed collection of sets. To make this construction explicit we first need the notion of S sorted *theory morphism.*

Definition 2.2.3 *Let $\mathcal{T}$ and $\mathcal{T}'$ be two S sorted algebraic theories. A theory morphism $\mathcal{F} : \mathcal{T} \rightarrow \mathcal{T}'$ is a functor which preserves objects and w-ary variables, that is, for any $w \in Obj(\mathcal{T})$, $\mathcal{F}(w) = w$ and $\mathcal{F}(\pi^w_{w_i}) = \pi^w_{w_i}$ in $\mathcal{T}'$*

The category of S sorted theories, $\mathcal{TH}_S$, can now be defined. The objects of $\mathcal{TH}_S$ are S sorted theories and the morphisms of $\mathcal{TH}_S$ are S sorted theory morphisms. If $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are S sorted theories and $\mathcal{F} : \mathcal{A} \rightarrow \mathcal{B}$ and $\mathcal{G} : \mathcal{B} \rightarrow \mathcal{C}$ are two S sorted theory morphisms then their composition is defined such that $\mathcal{G} \circ \mathcal{F}(w) = w$ and $\mathcal{G} \circ \mathcal{F}(\pi^w_{w_i}) = \pi^w_{w_i}$.

Definition 2.2.4 *A functor*

$$T : \mathcal{TH}_S \rightarrow \mathcal{SET}$$

may now be defined as follows: every S sorted theory is mapped to an $S^* \times S^*$ indexed family of sets $\mathcal{T}_{u,w}$ where $\mathcal{T}_{u,w} = \mathcal{T}(u, w)$. If $G : \mathcal{T} \rightarrow \mathcal{T}'$ is an S sorted theory morphism then $T(G)$ is an $S^* \times S^*$ indexed family of maps $G_{u,w}$ such that for every $\theta \in \mathcal{T}_{u,w}$, $G(\theta) \in \mathcal{T}'_{u,w}$

The free algebraic theory can now be defined as follows[3].

Definition 2.2.5 ([Law63, LG85]) *Let Σ be a signature, that is, an $S^* \times S$ sorted set of operator symbols. The* free category *on Σ, $\mathcal{F}_\Sigma$, is defined as follows: (1) the objects of $\mathcal{F}_\Sigma$ are the strings in S^* and (2) the morphisms of $\mathcal{F}_\Sigma(u,v)$ are the morphisms of $T_0(u,v)$ (of example 2.2.2) together with an arrow*

$$s \xrightarrow{\sigma} w$$

for every $\sigma \in \Sigma_{w,s}$. The arrows in $\mathcal{F}_\Sigma$ are closed under tupling, that is, if $\varphi_i : u_i \to w$, $1 \leq i \leq n$ are arrows in $\mathcal{F}_\Sigma$ then so is $\langle \varphi_1, \ldots, \varphi_n \rangle : u \to w$, and composition. The arrows in $\mathcal{F}_\Sigma$ are also subject to the following three axioms [Law63]:

Projection *if $\langle \Phi_1, \ldots, \Phi_n \rangle : w \to u$ then $\langle \Phi_1, \ldots, \Phi_n \rangle \circ \pi^w_{s_i} = \Phi_i$;*

Identity *if $\Phi : u \to w$, $u = u_1 \ldots u_n$ and $w = w_1 \ldots w_m$ then $\Phi \circ \langle \pi^u_{u_1}, \ldots, \pi^u_{u_n} \rangle = \Phi$ and $\langle \pi^w_{w_1}, \ldots, \pi^w_{w_n} \rangle \circ \Phi = \Phi$;*

Uniqueness *if $\Phi : u \to w$ then $\langle \Phi \circ \pi^u_{u_1}, \ldots, \Phi \circ \pi^u_{u_n} \rangle = \Phi$.*

We then have the following theorem from [LG85]:

Theorem 2.2.6 *$\mathcal{F}_\Sigma$ has the following universal property: if $\mathcal{T}$ is any S sorted theory, η the inclusion of the $S^* \times S$ sorted set into $T(\mathcal{F}_\Sigma)$ and $f : \Sigma \to T(\mathcal{T})$ then there is a unique f^* which make the following diagram commute,*

$$\begin{tikzcd}
\Sigma \arrow[r, "\eta"] \arrow[dr, "f"'] & T(\mathcal{F}_\Sigma) \arrow[d, "f^*"] \\
 & T(\mathcal{T})
\end{tikzcd}$$

Theorem 2.2.6 expresses the fact that $\mathcal{F}_\Sigma$ is the *free* algebraic theory over a given signature.

[3]The *Free* algebraic theory is the left adjoint to the functor T in definition 2.2.4 [AM75, Law63, KR89]. We choose a simpler definition due to [BW85] because it avoids the necessity to introduce adjunctions before defining the algebraic theory defined by a presentation.

Example 2.2.7 *Consider the signature for groups given by a single sort G, and operations $e : \to G$, $+ : G \times G \to G$ and $- : G \to G$. The objects of $\mathcal{F}_{Grp}$ are elements of the free monoid generated by the symbol $\{G\}$. Some typical operations of $\mathcal{F}_{Grp}$ are given below:*

$$\begin{tikzcd}
G_1G_2G_3 & & G_4G_5 \arrow[ll, "\langle - \circ \pi_{G_1}, - \circ \pi_{G_2} \rangle"'] \\
G_1G_3 \arrow[u, "\langle \pi_{G_1}, \pi_{G_3} \rangle"] & & G \arrow[ll, "+"]
\end{tikzcd}$$

We use subscripts to denote the particular copy of G that a variable is referring to and use π_{G_i} as a shorthand for $\pi_{G_i}^{G_1 \ldots G_n}$.

An algebraic theory satisfies a set of equations if certain other diagrams commute in the category. For example in a theory of groups, like that of example 2.2.7, the associativity of $+$ is expressed by saying that the diagram:

$$\begin{tikzcd}
G_1G_2G_3 & & G_1G \arrow[ll, "\langle \pi_{G_1}, \pi_{G_2} + \pi_{G_3} \rangle"'] \\
GG_3 \arrow[u, "\langle \pi_{G_1} + \pi_{G_2}, \pi_{G_3} \rangle"] & & G \arrow[ll, "+"] \arrow[u, "+"']
\end{tikzcd}$$

commutes. Equations are imposed on free algebraic theories by giving a congruence relation on the arrows of that theory to force the diagrams, expressing equations, to commute.

Definition 2.2.8 ([Law63], [JAG75]) *A* Theory Congruence, *R over an algebraic theory $\mathcal{T}$, is an $S^* \times S^*$ indexed family of equivalence relations such that*

1. *$R(\nu, \omega) \subseteq \mathcal{T}(\nu, \omega) \times \mathcal{T}(\nu, \omega)$*

2. *if $\langle \varphi, \varphi' \rangle \in R(\nu, \omega)$ and $\langle \theta, \theta' \rangle \in R(\omega, \mu)$ then $\langle \theta \circ \varphi, \theta' \circ \varphi' \rangle \in R(\nu, \mu)$*

3. *if $\langle \theta_i, \theta'_i \rangle \in R(s_i, \omega)$ for all $i \in \{1, \ldots, n\}$, $s_i \in S$ and $\omega \in S^*$, then*

$$(\langle \theta_1, \ldots, \theta_n \rangle, \langle \theta'_1, \ldots, \theta'_n \rangle) \in \mathcal{R}(\nu, \omega)$$

where $\nu = s_1 \ldots s_n$

The Theory Congruence, *R_E, generated by a set of equations E is defined as follows;*

4. If $\langle X_w, t, t'\rangle \in E_s$ then $\langle t, t'\rangle \in R_E(s, w)$

***5.** R_E is the least equivalence relation closed under (4) and rules (2) and (3) above.*

Definition 2.2.9 ([JAG75]) *If R is a theory congruence and $\mathcal{T}$ an algebraic theory then the* factor theory, *$\mathcal{T}/R$, of $\mathcal{T}$ by R is defined as follows: the objects of $\mathcal{T}/R$ are the same as those of $\mathcal{T}$ and the arrows are equivalence classes of arrows in $\mathcal{T}$ such that $\varphi_1, \varphi_2 : u \to w$ are equivalent in $\mathcal{T}/R$ iff $\langle\varphi_1, \varphi_2\rangle \in R$.*

The theory presented by a signature and set of S sorted equations (diagrams) is defined to be the *factor theory* (see [JAG75]) $\mathcal{F}_\Sigma/R_E$ of the free algebraic theory presented by Σ by the theory congruence generated by E.

In the *free concrete theory* of [JAG75] each operation $\sigma : s \to w$ is defined as a term t in $T_\Sigma(X)$ in which there is at most one free variables in t for each w_i in w. For this reason we blur the distinction between terms in $T_\Sigma(X)$ and arrows in $\mathcal{F}_\Sigma$ and by abusing notation we write $t' \leq t$ if $t \to_{\mathcal{B}}^* t'$ using a set of rewrite $\mathcal{B}$ and t and t' are arrows in $\mathcal{F}_\Sigma$.

A reduction relation (see 2.1.10) may be translated into the language of S sorted theories by considering the pre-order defined by such a relation (as in section section 2.1). Let $t = \sigma \circ \langle\xi_1, \ldots, \xi_n\rangle$ and t' be two morphisms in the algebraic theory presented by (Σ, E). If $t \to_{\mathcal{B}}^* t'$ and

$$t = \sigma \circ \langle\xi_1, \ldots, \xi_n\rangle$$

then $\langle t, t'\rangle \in \mathcal{F}_\Sigma/R_E(s, w)$ for some $s \in S$ and $w \in S^*$ and so the diagram

$$\begin{tikzcd}
s & u \arrow[l, "\sigma"'] \\
 & w \arrow[u, "\langle\xi_1, \ldots, \xi_n\rangle"'] \arrow[ul, "t'"]
\end{tikzcd}$$

must commute with $t \geq t'$. The reduction pre-order can be extended to arbitrary u-tuples of operators $\langle \xi_1, \ldots, \xi_n \rangle : u \to w$ by defining

$$\langle \xi_1, \ldots, \xi_n \rangle \leq \langle \xi'_1, \ldots, \xi'_n \rangle$$

if and only if each $\xi_i \leq \xi'_i$. This ordering of the arrows in $\mathcal{F}_\Sigma$ is also preserved by composition.

2.3 Algebras for S Sorted Theories

There are a number of ways of describing the algebras for an S sorted algebraic theory. For single sorted theories Lawvere [Law63] describes algebras of a theory $\mathcal{T}$ as those functors in $\mathcal{SET}^{\mathcal{T}^{op}}$ which preserve products[4]. In [LG85] three equivalent ways of describing algebras are given including the interpretation of Lawvere's. We would not only like to describe the algebras of an S sorted theory $\mathcal{T}$ as pairs (A, γ) as in section 2.1 but also to have the constructions independent of the underlying category because of our desire to study algebras in λ calculus.

For the present fix a category $\mathcal{C}$ and consider an endofunctor $\Psi : \mathcal{C} \to \mathcal{C}$.

Definition 2.3.1 ([SP82]) *Let $\mathcal{C}$ be category and $\Psi : \mathcal{C} \to \mathcal{C}$ an endofunctor.*

1. *A Ψ* Algebra *is a pair (A, γ) such that γ is a $\mathcal{C}$ morphism $\gamma : \Psi A \to A$. Dually a Ψ* CoAlgebra *is a pair (γ, B) such that γ is a $\mathcal{C}$ morphism $\gamma : B \to \Psi B$.*

2. *A Ψ algebra (A, γ) is a* Fixed Point *of Ψ if γ is an isomorphism in $\mathcal{C}$.*

The algebraic structure of a Ψ algebra A is given by the map $\gamma : \Psi A \to A$.

Definition 2.3.2 *A Ψ* Homomorphism *is a morphism in $\mathcal{C}$ such that the diagram*

$$\begin{tikzcd}
\Psi A \arrow[r, "\Psi f"] \arrow[d, "\gamma_A"'] & \Psi B \arrow[d, "\gamma_B"] \\
A \arrow[r, "f"'] & B
\end{tikzcd}$$

commutes.

[4]Recall that the variables $\pi_w^{s_i}$ are essentially injections

The Ψ algebras and Ψ homomorphisms form a category with composition defined as follows: if $f : (A_1, \gamma_1) \rightarrow (A_2, \gamma_2)$ and $f : (A_2, \gamma_2) \rightarrow (A_3, \gamma_3)$ then their composition is defined by the composition of f and g in the underlying category $\mathcal{C}$. Denote the category of Ψ algebras and Ψ homomorphisms by Ψ_{Alg}. In this way each endofunctor Ψ defined on $\mathcal{C}$ specifies a category of algebras in $\mathcal{C}$ much the same as a signature and set of equations specify a category of algebras in universal algebra.

An *initial fixed point* is simply a fixed point of Ψ which is initial in the subcategory of fixed points of Ψ (dually the terminal fixed point is a terminal object in the category of fixed points and homomorphisms). Initial Ψ algebras and initial fixed points of Ψ are related by the following lemma.

Lemma 2.3.3 ([SP82]) *The initial Ψ algebra, if it exists, is also the initial fixed point of Ψ.*

The proof of this lemma may be found in either [SP82] or [MA86]. If $\mathcal{C}$ satisfies some additional properties then there is a canonical way to construct initial (and therefore by duality terminal) fixed points. The construction is given in the proof of the *Basic Lemma* of [SP82]. To formulate the basic lemma of [SP82] let ω be the category with the natural numbers as objects and morphisms all pairs (i,j) of natural numbers i and j. If (i,j) and (j,k) are any two morphisms in ω their composition is given by (i,k) and the identity for i is (i,i). An ω diagram $\triangle$ in a category $\mathcal{C}$ is a functor $\triangle : \omega \rightarrow \mathcal{C}$.

Definition 2.3.4 ([LS81]) *$\mathcal{C}$ is an ω* Category *iff $\mathcal{C}$ has an initial object and all ω diagrams have a colimit in $\mathcal{C}$. Dually, $\mathcal{C}$ is an ω^{op}* Category *iff $\mathcal{C}$ has a terminal object and all ω^{op} diagrams have a limit in $\mathcal{C}$* [5].

Let $\mathcal{C}$ be an ω category and $\perp$ be the initial object of $\mathcal{C}$. For each object A, let $\perp_A : \perp \rightarrow A$ be the unique arrow in $\mathcal{C}$ given by the initiality of $\perp$. A *Mediating Morphism* $\alpha : \mu \rightarrow \nu$ is the unique arrow from the limiting cone μ of the diagram $\triangle$ to any other cone ν for $\triangle$ (see [Mac71]).

[5] *More precisely* $\triangle^{op} : \omega \rightarrow \mathcal{C}^{op}$

$$\begin{tikzcd}
& \mu \arrow[dl, "\mu_0"'] \arrow[d, "\mu_1"'] \arrow[dr, "\mu_2"] & & \nu \arrow[ll, "\alpha"'] \arrow[dlll, "\nu_0"'] \arrow[dll, "\nu_1"'] \arrow[dl, "\nu_2"] \\
\perp \arrow[r, "\perp_\Psi"] & \Psi\perp \arrow[r, "\Psi\perp_\Psi"] & \Psi^2\perp \arrow[r, "\Psi^2\perp_\Psi"] & \cdots
\end{tikzcd}$$

Figure 2: Mediating morphisms

Now suppose that $\triangle$ is the ω chain

$$D_0 \xrightarrow{f_0} D_1 \xrightarrow{f_1} D_2 \xrightarrow{f_2} \cdots$$

in a category $\mathcal{C}$. Define $\triangle^-$ to be the chain

$$D_1 \xrightarrow{f_1} D_2 \xrightarrow{f_2} \cdots$$

in $\mathcal{C}$ and μ^- to be the colimit of $\triangle^-$. If F is a functor from the ω category $\mathcal{C}$ to the ω category $\mathcal{D}$ then $F\triangle$ is the chain

$$F(D_0) \xrightarrow{F(f_0)} F(D_1) \xrightarrow{F(f_1)} F(D_2) \xrightarrow{F(f_2)} \cdots$$

in $\mathcal{D}$ and if $\mu : \triangle \to A$ is a colimit for $\triangle$ in $\mathcal{C}$ then $F(\mu) : F\triangle \to F(A)$ in $\mathcal{D}$.

Lemma 2.3.5 (The Basic Lemma [SP82]) *Let $\mathcal{C}$ be an ω category and $\Psi : \mathcal{C} \to \mathcal{C}$. Also let $\triangle$ be the diagram given by*

$$\triangle = \langle \Psi^n\perp, \Psi^n\perp_{\Psi\perp} \rangle$$

If $\mu : \triangle \to A$ and $\Psi\mu : \Psi\triangle \to \Psi A$ are both colimiting cones for $\triangle$ in $\mathcal{C}$ then the initial Ψ algebra exists and is (A, α), where α is the map $\alpha : \Psi\mu \to \mu^-$.

The more familiar kinds of universal algebras can be defined in $\mathcal{SET}$ as a special case of the more general definition given above. Let Σ be an arbitrary signature.

Definition 2.3.6 ([MA86]) *The functor $X_\Sigma : \mathcal{SET}^S \to \mathcal{SET}^S$* corresponding *to a signature Σ is defined as follows: $X_\Sigma(A) = \overline{A}$ where $\overline{A}$ is the functor in $\mathcal{SET}^S$ such that for each $s \in S$*

$$\overline{A}(s) = \coprod\{A^\omega \,|\, \sigma \in \Sigma_{\omega s}\, and\, \omega \in S^*\}$$

The notation

$$\overline{A}(s) = \coprod\{A^{\omega} \mid \sigma \in \Sigma_{\omega s}\, and\, \omega \in S^*\}$$

is from [MA86] and means the disjoint union of the family of sets A^{ω} where for every $\omega \in S^*$, if $\Sigma_{\omega s}$ is not empty then for every operator $\sigma \in \Sigma_{\omega s}$ there is a copy of A^{ω} in the disjoint union.

An X_Σ algebra is then a pair (A, γ) consisting of an S-sorted set (functor in $\mathcal{SET}^S$) A and a map γ interpreting the algebraic structure on A. Since for each $s \in S$ A_s is a coproduct of S sorted sets the structure map for sort s may be given by case analysis as follows:

$$\gamma_s = [\gamma_{\sigma_1}, \ldots, \gamma_{\sigma_n}]$$

where each $\gamma_{\sigma_i} : A^{\omega_i} \rightarrow A_s$ interprets the operator symbol σ_i in A.

The category $\mathcal{SET}^S$ is an ω category (see [MA86]) with initial object $\emptyset$ and so we can apply the basic lemma 2.3.5 to construct the initial X_Σ algebra in $\mathcal{SET}^S$. If (T, γ) is an X_Σ algebra then we have the following S sorted family of maps in $\mathcal{SET}$ .where γ is the family of maps:

$$\begin{array}{rcl} X_\Sigma(T_{S_1}) & \overset{\gamma_{s_1}}{\rightarrow} & T_{s_1} \\ & \vdots & \\ X_\Sigma(T_{S_n}) & \overset{\gamma_{s_n}}{\rightarrow} & T_{s_n} \end{array}$$

If (T, γ) is the initial X_Σ algebra then each of the γ_{s_i} above are isomorphisms and so we obtain the equations:

$$\begin{array}{rcl} X_\Sigma(T_{S_1}) & \cong & T_{s_1} \\ & \vdots & \\ X_\Sigma(T_{S_n}) & \cong & T_{s_n} \end{array}$$

If X_Σ is the functor corresponding to a signature Σ then the initial fixed points of X_Σ, the initial algebra and the term algebra $T_\Sigma(\emptyset)$ (definition 2.1.2) are related by the following lemma.

Lemma 2.3.7 *Let $\Sigma = (S, \Omega)$ and $T_\Sigma(\emptyset)$ be the Σ term algebra on the empty set of generators as defined in 2.1.2. Then $(T_\Sigma(\emptyset), \gamma)$ is the initial X_Σ algebra, where each $s \in S$ and each operation of sort s in Σ*

$$\gamma_\sigma(\langle t_{s_1}, \ldots, t_{s_n} \rangle) = \sigma(t_1, \ldots, t_n)$$

We use lemma 2.3.7 to relate the definition of initial X_Σ to the definition of term algebras (definition 2.1.2).

Chapter 3

λ Calculus

In the previous chapter we introduced some fundamentals of algebraic specification, term rewriting and algebraic theories. The point made is that there is a correspondence between equational theories in universal algebra and the algebraic theories of Lawvere [Law63]. In recent years such connections have also been studied between the simply typed λ calculus and cartesian closed categories, for example, [Sco80, Lam80a, LS86, Poi86]. For λ calculus and cartesian closed categories the connection has been formalised more strongly than in the case of equational theories by stating that simply typed λ calculi are the *internal languages* of cartesian closed categories [LS86] (but see also [Lam74] and [Lam80b]).

The pure λ calculus also gives rise to a cartesian closed category [Sco80, Koy82], perhaps best described as its *internal* cartesian closed category, but it is of quite a different nature to those arising from simply typed λ calculi. In the case of the simply typed λ calculi one takes the types and terms to be the objects and arrows respectively of a cartesian closed category while in the case of the pure λ calculus the types and arrows of a cartesian closed category are embedded within the terms of the calculus.

For us the pure λ calculus can be considered to be a simple paradigmatical functional programming language embodying recursion, through the use of the fixed point combinator Y, and the idea from combinatory logic [CF68, Bar84] that functions of many arguments can be reduced to functions of one argument. The same idea is also present in the theory of cartesian closed categories [Lam80a] in form of the exponential transpose of functions. It is the case for those languages (like Standard ML [HMT90, MT90]) using the Milner-Hindley type inference system [Mil78, DM82] that expressions are first type checked and then evaluated using untyped expressions. By taking the pure λ calculus (or its models) and imposing a type discipline (in the form of a cartesian closed category) we hope to recapture some of the properties of this paradigm.

Again we claim no originality for the material in this chapter which may be found in [Bar84, Koy82, LS86, Mac71].

3.1 λ Algebras

The pure λ calculus is given by the following set of terms, axioms and rules of deduction [Bar84]. The set Λ of terms is defined to be the least set satisfying the following:

Variables $X \in \Lambda$ where X is a countable set of variables.

Abstraction If $M \in \Lambda$ and $x \in X$ then $\lambda x.M \in \Lambda$.

Application If M, $N \in \Lambda$ then $(M\,N) \in \Lambda$.

If a set of constants C is included in the definition then the set of λ terms with constants C is denoted by $\Lambda(C)$ otherwise the set of pure λ terms is denoted Λ.

λ calculus is an *equality* theory where equality between λ terms is written as $M =_{CNV} N$ and called *Conversion.* λ calculus is called an equality theory in [HS86] to distinguish it from equational theories in which equality is "first order". Conversion is formally defined by the following axioms and rules of inference:

Definition 3.1.1 *(α)* $\lambda x.M =_{CNV} \lambda y.M[y/x]$ *if y is not free in M*

(β) $(\lambda x.M)\,N =_{CNV} M[N/x]$

(ζ) If $M =_{CNV} N$ *then* $(X\,M) =_{CNV} (X\,N)$ *and* $(M\,X) =_{CNV} (N\,X)$

(ξ) If $M =_{CNV} N$ *then* $\lambda x.M =_{CNV} \lambda x.N$ *for some variable* $x \in X$.

The notation $M[N/x]$ means that N is substituted for every free occurrence of x in M. Formally it is defined as follows:

Definition 3.1.2 ([Bar84])

$$\begin{aligned}
x[N/x] &\equiv N \\
y[N/y] &\equiv y, \quad \textit{if } x \neq y \\
(\lambda x.M)[N/y] &\equiv \lambda x.M[N/y] \quad \textit{provided } x \neq y \\
(\lambda x.M)[N/y] &\equiv \lambda x.M \quad \textit{if } x \neq y \\
(M_1\; M_2)[N/x] &\equiv (M_1[N/x])\,(M_2[N/x])
\end{aligned}$$

If $M =_{CNV} N$ then we say that M and N are *convertible.* In the sequel we adopt the convention in [Bar84] that all (α) convertible terms are identified.

Turning now to models we now have the following.

Definition 3.1.3 ([Mey82, Bar84]) *A* Combinatory Algebra *is a set $\mathcal{M}$ together with a binary operation $\cdot$ (called* application*) and distinguished constants k and s satisfying the following two equations:*

$$\begin{aligned} (k \cdot x) \cdot y &= x \quad \textit{for all } x, y \in \mathcal{M} \\ ((s \cdot x) \cdot y) \cdot z &= (x \cdot z) \cdot (y \cdot z) \quad \textit{for all } x, y, z \in \mathcal{M} \end{aligned}$$

The theory presented by k and s is called combinatory logic *and is denoted by CL.*

Note that $\cdot$ is left associative and so some parentheses can be omitted in future. Combinatory algebras are *combinatory complete* which is to say that for every polynomial P with variables in $\{x_1, \ldots, x_n\}$ over a combinatory algebra $\mathcal{M}$ there is an element $f \in \mathcal{M}$ such that

$$\forall\, a_1 \ldots a_n \in \mathcal{M} \quad f\, a_1 \ldots a_n = P(a_1, \ldots, a_n)$$

There is a natural way to handle abstraction and application within such algebras. Suppose $\mathcal{M}$ is a combinatory algebra. Define the set of terms over $\mathcal{M}$ as follows:

- Variables $X_0,\ X_1,\ \ldots \in T(\mathcal{M})$
- If $a \in \mathcal{M}$ then $c_a \in T(\mathcal{M})$
- If $A \in T(\mathcal{M}),\ B \in T(\mathcal{M})$ then $(A \cdot B) \in T(\mathcal{M})$

We can now define an abstraction operator as follows:

Definition 3.1.4 ([Bar84]) *Define the operation of variable abstraction, λ^* by;*

$$\begin{aligned} \lambda^* x.x &= s \cdot k \cdot k && \textit{(or I for Identity)} \\ \lambda^* x.P &= (k \cdot P) && \textit{if } x \textit{ does not occur in } P \\ \lambda^* x.(M\,N) &= s \cdot (\lambda^* x.M) \cdot (\lambda^* x.N) && \end{aligned}$$

With this definition of abstraction a two way translation between combinatory algebras and λ terms may be defined.

Definition 3.1.5 *Let $\mathcal{M}$ be a combinatory algebra and $\Lambda(\mathcal{M})$ the set of λ terms with constants from $\mathcal{M}$. The mappings*

$$CL : \Lambda(\mathcal{M}) \rightarrow T(\mathcal{M})$$

and

$$L : T(\mathcal{M}) \rightarrow \Lambda(\mathcal{M})$$

are given by:

$$\begin{aligned} CL(x) &= x \\ CL(c) &= c \\ CL(M\,N) &= CL(M)\,CL(N) \\ CL(\lambda x.M) &= \lambda^* x.CL(M) \end{aligned} \qquad \begin{aligned} L(x) &= x \\ L(a) &= a \\ L(P \cdot Q) &= L(P)\,L(Q) \\ L(k) &= \lambda x.\lambda y.x \\ L(s) &= \lambda x.\lambda y.\lambda z.(x\,z)\,(y\,z) \end{aligned}$$

If $\rho : X \rightarrow \mathcal{M}$ is an assignment of values to variables in $\mathcal{M}$ then any term in $T(\mathcal{M})$ may be assigned a value in $\mathcal{M}$ as follows:

$$\begin{aligned} (x)_\rho &= \rho(x) \\ (M \cdot N)_\rho &= (M)_\rho \cdot (N)_\rho \end{aligned}$$

This now opens up the way to give a denotation for each of the λ terms in a combinatory algebra $\mathcal{M}$.

Definition 3.1.6 *The* Interpretation *of λ terms in $\mathcal{M}$ is defined by:*

$$[\![M]\!]_\rho = (CL(M))_\rho$$

$M = N$ *is* Valid *in ρ , or $\mathcal{M}, \rho \models M = N$ iff $[\![M]\!]_\rho = [\![N]\!]_\rho$ and* valid *iff it is valid in ρ for every $\rho : X \rightarrow \mathcal{M}$ which is written as $\mathcal{M} \models M = N$*

Here and in the sequel we use $[\![\_]\!]$ for the map taking each syntactic term onto its meaning or denotation in some abstract mathematical domain.

Definition 3.1.7 *A λ* algebra *is a combinatory algebra such that for all terms in $T(\mathcal{M})$*

$$\lambda \vdash L(M) = L(N) \quad \textit{implies} \quad \mathcal{M} \models M = N$$

An alternative characterisation of λ algebras is given by the following lemma:

Lemma 3.1.8 ([Bar84]) *Let $\mathcal{M}$ be a combinatory algebra. Then $\mathcal{M}$ is a λ algebra iff for all $M, N \in \Lambda(\mathcal{M})$*

1. $\lambda \vdash M = N$ *implies* $\mathcal{M} \models M = N$
2. $\mathcal{M} \models CL(L(k)) = k$ *and* $\mathcal{M} \models CL(L(s)) = s$

Essentially λ algebras are those combinatory algebras in which all the equations between λ terms deducible from the axioms by the rules of inference in definition 3.1.1 hold. Among the λ algebras there are the models generated by the terms (perhaps with constants) modulo convertibility, there is the graph model $\mathcal{P}(\omega)$, the models D^∞ and the Böhm tree models [Bar84].

Polynomials $\varphi(x_1, \ldots, x_n)$ over a λ algebra may often be represented within the algebra which gives rise to the concept of *representability.*

Definition 3.1.9 ([Bar84]) *Let $\varphi(x_1, \ldots, x_n)$ be a polynomial in n variables over a λ algebra $\mathcal{M}$. Then $\varphi(x_1, \ldots, x_n)$ is* Representable *over $\mathcal{M}$ if there exists an $f \in \mathcal{M}$ such that for all $a_1, \ldots, a_n \in \mathcal{M}$, we have*

$$f\, a_1, \ldots a_n = \varphi(a_1, \ldots, a_n)$$

Later we will need to reason about the properties of fix points and so we restrict the class of λ algebras under consideration. Firstly, however, we will need the concepts of *head normal form* and *solvability.*

Definition 3.1.10 ([Bar84]) 1. *A λ term M is a* Head Normal Form *(or hnf) if M is of the form*

$$\lambda x_1 \ldots \lambda x_n . x_i\, N_1 \ldots N_n$$

and M is a hnf if there is an N such that N is a hnf and $M =_{CNV} N$.

2. *M is* Solvable *if there are terms $\{N_1, \ldots, N_n\}$ such that $M\, N_1 \ldots N_n = I$, where I is $\lambda x.x$.*

Fact 3.1.11 ([Bar84]) *A term M has a head normal form iff it is solvable.*

Fact 3.1.12 ([**Bar84**]) *If M is a term with no hnf then so are $(M\,N)$, $M[N/x]$ and $\lambda x.M$ for all $N \in \Lambda$.*

Intuitively, if a term has a head normal form then there is always some finite part of it (the variable at the head) which can be computed, even if the remaining terms $N_1, \ldots, N_n$ fail to produce any results.

Adding closed equations between λ terms results in new theories as follows. First, a λ theory is *consistent* if we cannot prove all possible equations.

Definition 3.1.13 ([**Bar84**]) *Let $\mathcal{E}$ be a set of closed equations between λ terms and Th($\mathcal{E}$) the set of equations provable in $\lambda + \mathcal{E}$ by the rules of conversion. Then $\mathcal{E}$ is a λ* Theory *if $\mathcal{E}$ is consistent and $Th(\mathcal{E}) = \mathcal{E}$.*

A λ theory in which all terms with no head normal form have been equated is useful to reason about nonterminating functions. Doing so gives the following λ theory:

Definition 3.1.14 ([**Bar84**]) *Let*

$$\mathcal{H}_0 = \{M = N \mid M, N \; have\, no\, hnf\}$$

and $\mathcal{H} = Th(\mathcal{H}_0)$ be the λ theories defined by equating the terms with no hnf.

[Bar84]

Fact 3.1.15 *It is consistent to equate all the terms without a head normal form*

Introduce a new symbol into the class of terms of the λ calculus, Ω, together with the following (infinite) class of equations:

$$M = \Omega \quad if \; M \; does \; not \; have \; a \; head \; normal \; form$$

By fact 3.1.12 the following equations are also provable:

$$\begin{aligned}(\Omega\, M) &= \Omega \\ \lambda x.\Omega &= \Omega\end{aligned}$$

To explain the semantics of terms which do have a head normal form requires some machinery from the theory of domains [Sco76, Sto77] (or [Bir48] for the theory of ordered sets). Let $\sqsubseteq$ be a partial order on a set $\mathcal{M}$. A partially ordered set $(\mathcal{M}, \sqsubseteq)$ is an *ω complete partial order* if it has a least element $\bot$ and every ascending sequence

$$a_0 \sqsubseteq a_1 \sqsubseteq a_2 \ldots$$

has a least upper bound $\bigsqcup a_i$ in $\mathcal{M}$ [Sco76]. A subset $X \subseteq \mathcal{M}$ is *directed* if for every $a, b \in X$ there is a $c \in X$ such that $a \sqsubseteq c$ and $b \sqsubseteq c$. From this it follows that every finite subset of X has a least upper bound, $\bigsqcup X$, in X.

Definition 3.1.16
A function $f : \mathcal{M} \to \mathcal{M}$ is (directed) continuous if and only if for any directed set $X \subseteq \mathcal{M}$

$$\bigsqcup f(X) = f(\bigsqcup X)$$

We denote the space of continuous functions on $\mathcal{M}$ is by $[\mathcal{M} \to \mathcal{M}]$.

Fact 3.1.17 ([Bar84, Sto77]) *If M_1 and M_2 are ω complete partial orders then so is $M_1 \times M_2$, the product of M_1 and M_2. The ordering on $M_1 \times M_2$ is given by*

$$(m_1, m_2) \sqsubseteq (m_1', m_2') \quad \textit{iff} \quad m_1 \sqsubseteq m_1' \textit{ and } m_2 \sqsubseteq m_2'$$

Theorem 3.1.18 ([Bar84]) *1. Every $f \in [\mathcal{M} \to \mathcal{M}]$ has a fixed point;*

2. Moreover there exists a function $Fix \in [[\mathcal{M} \to \mathcal{M}] \to \mathcal{M}]$ such that for all $f \in [\mathcal{M} \to \mathcal{M}]$, $Fix(f)$ is the least fixed point of f.

$Fix(f)$ is usually constructed as $\bigsqcup \{f^n(\bot)\}_{n \in \omega}$. The usual interpretation of the fixed point combinator Y in complete partial orders is as the *fix* operator, while $Y(f)$ for any term f is then $[\![Y(f)]\!] = fix([\![f]\!])$ [Sco76, Sto77]. It is also usual to interpret Ω as $\bot$, the least element of $\mathcal{M}$.

Definition 3.1.19 *1. A λ algebra $(\mathcal{M}, \cdot)$ is* continuous *if $\mathcal{M}$ is a complete partial order and $\cdot$ is a continuous operation on $\mathcal{M}^{\mathcal{M}} \times \mathcal{M}$.*

2. An interpretation of the pure λ calculus in $\mathcal{M}$ is standard *if:*

(a) for any λ term M with no head normal form $[\![M]\!] = \perp$ (and consequently $[\![M]\!] \sqsubseteq a$ for any other $a \in \mathcal{M}$);

(b) for any λ term f, $[\![Y f]\!] = fix([\![f]\!])$ in $\mathcal{M}$.

3.2 Cartesian Closed Categories and λ Calculus

Cartesian closed categories are usually defined by stating that the functors $\mathcal{C} \rightarrow 1$, the diagonal $\mathcal{C} \xrightarrow{\Delta} \mathcal{C} \times \mathcal{C}$ and $\mathcal{C} \xrightarrow{-\times A} \mathcal{C}$ all have right adjoints for any object A of $\mathcal{C}$. Here we adopt an axiomatic presentation due to Lambek and Scott [LS86]. What follows is taken from [LS86, Lam80a, Sco80, Koy82, Bar84].

Definition 3.2.1 *A* Cartesian Closed Category *is a category with a terminal object, denoted here by $1_{\mathcal{C}}$, and for every pair of objects A and B a product object $A \times B$ and an exponential object $(A \rightarrow B)$. For every arrow $f : A \times B \rightarrow C$ there exists an arrow $f^* : A \rightarrow (B \rightarrow C)$, called the exponential transpose of f and special arrows $ev_{A,B} : B^A \times A \rightarrow B$. The arrows of $\mathcal{C}$ must satisfy the following axioms:*

C1 for every object A there is an arrow $id_A : A \rightarrow A$ such that for all $f : A \rightarrow B$ and $g : C \rightarrow A$

$$f \circ id_A = f$$

and

$$id_A \circ g = g$$

C2 for any arrows $f : A \rightarrow B$, $g : B \rightarrow C$ and $h : C \rightarrow D$

$$(h \circ g) \circ f = h \circ (g \circ f)$$

C3 for any $f : A \rightarrow 1$, $f = \bigcirc_A$ where $\bigcirc_A$ is the unique arrow from A to 1;

C4 Let π and π' be projections for the product $A \times B$ and $f : C \rightarrow A$, $g : C \rightarrow B$, then $\pi \circ \langle f,g \rangle = f$ and $\pi' \circ \langle f,g \rangle = g$

C5 *for any arrow* $h : C \rightarrow A \times B$

$$\langle \pi \circ h, \pi' \circ h \rangle = h$$

C6 *if* $f : A \times B \rightarrow C$ *then* $ev_{BC} \circ f^* \times Id = f$

C7 *if* $g : A \rightarrow C^B$ *then* $(ev_{BC} \circ g \times Id)^* = g$

Axioms (C1) and (C2) are simply the axioms for any category. From these we can deduce the following identity

$$id_B \circ f \circ id_A = f$$

for any $f : A \rightarrow B$. Axiom (C3) states that $1_{\mathcal{C}}$ is a terminal object in the category while (C4) and (C5) are the axioms for surjective pairing. Axioms (C6) and (C7) taken together imply the following isomorphism

$$Hom(A \times B, C) \cong Hom(A, C^B)$$

and consequently the exponential transpose is a 1:1 mapping between these *Hom* sets.

A cartesian closed category with all finite coproducts is referred to as a *Bi-cartesian* closed category [LS86]. Essentially it is a cartesian closed category with an initial object $\emptyset_{\mathcal{C}}$ and for every pair of objects A and B their coproduct $A + B$ together with maps $inj_A : A \rightarrow A + B$, $inj_B : B \rightarrow A + B$ and $[f, g] : A + B \rightarrow C$ for every $f : A \rightarrow C$ and $g : B \rightarrow C$. The arrows of a cartesian closed category must satisfy the axioms (C1) to (C7) as well as the following three axioms:

C8 for every object A of $\mathcal{C}$ there exists a map $\emptyset_A : \emptyset_{\mathcal{C}} \rightarrow A$ such that for any map $f : \emptyset \rightarrow A$, $f = \emptyset_A$

C9 for every pair of maps $f : A \rightarrow C$ and $g : B \rightarrow C$, $[f, g] \circ inj_A = f$ and $[f, g] \circ inj_B = g$

C10 for any $h : A + B \rightarrow C$, $[h \circ inj_A, h \circ inj_B] = h$

By (C8) $Hom(\emptyset_{\mathcal{C}}, A)$ contains exactly one element, while (C9) and (C10) together imply the following isomorphism [LS86]:

$$Hom(A, C) \times Hom(B, C) \cong Hom(A + B, C)$$

Furthermore the isomorphism $Hom(A, C) \cong Hom(1_{\mathcal{C}}, C^A)$ characteristic of cartesian closed categories implies the existence of a map

$$\zeta^C_{AB} : C^A \times C^B \to C^{A+B}$$

For any arrows $f : A \to C$ and $g : B \to C$ the arrow $[f, g] : A + B \to C$ may be defined as

$$[f,g] \overset{def}{=} ev_{A+B,C} \circ \langle \zeta^C_{A,B} \circ \langle f \circ (\pi_{1A})^*, f \circ (\pi_{1B})^* \rangle \circ \bigcirc_{A+B}, Id_{A+B} \rangle \qquad (1)$$

Another facet of products and coproducts in bicartesian closed categories is that $\times$ distributes over $+$ giving the following two isomorphisms (see [LS86]):

$$A \times (B + C) \cong (A \times B) + (A \times C) \qquad (2)$$

$$(A + B) \times C \cong (A \times C) + (B \times C) \qquad (3)$$

For example the isomorphism in equation 2 above is given by the arrow

$$[id_A \times inj_B, id_A \times inj_C] : A \times B + A \times C \to A \times (B + C)$$

Products and sums in a bicartesian closed category can be expressed in terms of functors. We already mentioned that products can be defined by stating that a right adjoint to the diagonal functor $\Delta : \mathcal{C} \to \mathcal{C} \times \mathcal{C}$ exists and similarly coproducts can be defined by stating that a left adjoint to the diagonal exists [AM75, Mac71]. It is not within the scope of this work to delve further except to note that a product functor

$$\_ \times \_ : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$$

and sum functor

$$\_ + \_ : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$$

may be defined for bicartesian closed categories by this means.

A *Natural Numbers Object* [LS86] in a cartesian closed category is an initial object in the category of all diagrams of the form:

$$\begin{tikzcd} 0 \arrow[r, "0"] & N \arrow[r, "S"] & N \end{tikzcd}$$

which amounts to saying that for any object A and arrows $a : 1_C \to A$ and $f : A \to A$ there is a *unique* $h : N \to A$ making the following diagram,

$$\begin{tikzcd}
1_C \arrow[r, "0"] \arrow[dr, "a"'] & N \arrow[r, "S"] \arrow[d, "h"] & N \arrow[d, "h"] \\
& A \arrow[r, "f"] & A
\end{tikzcd}$$

commute. If all that can be asserted is the existence of h, with no reference to its uniqueness, then this is called a *Weak Natural Numbers Object* [LS86].

We now present two different ways of constructing cartesian closed categories from λ calculi. For the present consider a simply typed λ calculus with natural numbers [LS86].

Types The set of types, T, is defined inductively as follows:

1. 1 and N are (the basic or ground) types
2. if α and β are types then so are $\alpha \to \beta$ and $\alpha \times \beta$

Terms Let $\{X_\alpha\}_{\alpha \in T}$ be a family of countable sets of variables and write $a : A$ to say that a is a term of type A. Then the terms are the least sets indexed by types closed under the following rules.

1. $* : 1$
2. if $a : \alpha$, $b : \beta$ and $c : \alpha \times \beta$ then $\langle a, b\rangle : \alpha \times \beta$, $\pi(c) : \alpha$ and $\pi'(c) : \beta$
3. if $f : \alpha \to \beta$ and $a : \alpha$ then $apply_{\alpha\beta}(f, a) : \beta$
4. if $x : \alpha$ and $\Phi(x) : \beta$ then $\lambda x.\Phi(x) : \alpha \to \beta$
5. $0 : N$ and if $n : N$ then $S(n) : N$
6. if $A : \alpha$, $f : \alpha \to \alpha$ and $n : N$ then $I(a, f, n) : \alpha$

The rules and axioms for this simply typed λ calculus are given in figure 3. We call this theory λ^τ.

According to [LS86] we can construct a cartesian closed category from a simply typed λ calculus. Given a simply typed λ calculus such as λ^τ, a a cartesian closed category $CCC(\lambda^\tau)$ is constructed as follows:

Logical Rules

$$e = * \quad \text{for any term } e \text{ of type } 1 \quad (\mathit{unit})$$

$$\lambda x.\Phi(x) = \lambda y.\Phi(x)[y/x] \quad (\alpha)$$

$$(\lambda x.\Phi(x))\, a = \Phi(x)[a/x] \quad (\beta)$$

$$\pi(\langle x, y\rangle) = x \quad \pi'(\langle x, y\rangle) = y \quad (\delta)$$

$$\langle \pi(z), \pi'(z)\rangle = z$$

Non Logical Axioms

$$I(a, f, 0) = a \quad \mathit{for\ all}\ a : \alpha\ \mathit{and}\ f : \alpha \to \alpha$$

$$I(a, f, S(n)) = f\,(h(a, f, n))$$

Rules of Inference

$$\frac{\Phi(x) = \Psi(x)}{\lambda x.\Phi(x) = \lambda x.\Psi(x)} \quad (\xi)$$

$$\frac{\Phi(x) = \Psi(x)}{X\,\Phi(x) = X\,\Psi(x)} \quad \frac{\Phi(x) = \Psi(x)}{\Phi(x)\,X = \Psi(x)\,X} \quad (\zeta)$$

and *Reflexivity*, *Symmetry* and *Transitivity*

Figure 3: Rules for the Simply Typed λ^{τ}

Objects The objects of $CCC(\lambda^\tau)$ are the types of λ^τ.

Arrows The arrows in $hom(\alpha, \beta)$ are equivalence classes of pairs $(x : \alpha, \phi(x))$ where $x : \alpha$ is a variable of type α and $\phi(x)$ is a term term of type β in λ^τ in which the only free variables are those of type α. Two arrows $(x : \alpha, \phi(x))$ and $(x : \alpha, \psi(x))$ are equivalent if and only if $\lambda^\tau \vdash \phi(x) = \psi(x)$.

For any object α the identity is $(x : \alpha, x)$ while the composition of $(x : \alpha, \phi(x))$ and $(y : \beta, \psi(y))$ is given by $(x : \alpha, \psi(\phi(x)))$, that is, composition is given by substitution.

The cartesian closed structure may now be recovered by making the following definitions.

$$\begin{aligned}
\bigcirc_\alpha &\overset{def}{=} (x : \alpha, *) \\
\pi_{\alpha\beta} &\overset{def}{=} (z : \alpha \times \beta, \pi(z)) \\
\pi'_{\alpha\beta} &\overset{def}{=} (z : \alpha \times \beta, \pi'(z)) \\
\langle (z : \alpha\Phi(z)), (z : \alpha, \Psi(z)) \rangle &\overset{def}{=} (z : \alpha, \langle \Phi(z), \Psi(z) \rangle) \\
(z : \alpha \times \beta, \chi(z))^* &\overset{def}{=} (x : \alpha, \lambda y : \beta . \chi(\langle x, y \rangle)) \\
ev_{\alpha,\beta} &\overset{def}{=} (y : \beta^\alpha \times \alpha, apply(\pi(y), \pi'(y)))
\end{aligned}$$

A weak natural numbers object may be recovered by choosing N to be the type of natural numbers and then making the following definitions:

$$\begin{aligned}
0 &\overset{def}{=} (x : 1_C, 0) \\
S &\overset{def}{=} (x : N, S(x)) \\
I_B &\overset{def}{=} (x : (B \times B^B) \times N, I(\pi(\pi(x)), \pi'(\pi(x)), \pi'(x)))
\end{aligned}$$

Then there exists an h making the following diagram:

$$\begin{tikzcd}
1_C \arrow[r, "0"] \arrow[dr, "a"'] & N \arrow[r, "S"] \arrow[d, "h"] & N \arrow[d, "h"] \\
& A \arrow[r, "f"] & A
\end{tikzcd}$$

commute in $CCC(\lambda^\tau)$ which can be seen by putting $h = (x : N, I_A(a, f, x))$ and then checking that the equations

$$h \circ (x\,;\, 1_C, 0) = a \quad \text{and} \quad f \circ h = h \circ S$$

hold.

Proposition 3.2.2 *The cartesian closed category generated by λ^τ is the free cartesian closed category generated by λ^τ*

The proof is given in [LS86]. It is easy to check that the definition above does indeed give a cartesian closed category by simply verifying that all the axioms (C1) through to (C7) hold. Thus, from a simply typed λ calculus a cartesian closed category has been obtained.

Considering now the pure λ calculus a different kind of cartesian closed category may be constructed [Sco80, Koy82, Bar84]. It turns out that this construction is relevant to any λ algebra and so we present it thus. First, some notation is required. We use the translation $L : T(\mathcal{M}) \longrightarrow \Lambda(\mathcal{M})$ of the previous section to translate elements of $\mathcal{M}$ into pure λ terms with (possibly) constants from $\mathcal{M}$. Denote $L(a)$ by $\overline{a}$. Note also that it may be the case that $a \in \mathcal{M}$ is not equal to the interpretation of a pure λ term (under the mapping CL) and so it is possible for $\overline{a}$ to simply be a constant from $\mathcal{M}$. On the other hand all terms which are the interpretation of pure λ terms will be translated back into pure λ terms, as is the case for all the closed terms [Bar84].

Definition 3.2.3 *Let $\mathcal{M}$ be a λ algebra and for any $a, b \in \mathcal{M}$ define composition by:*

$$a \circ b = [\![\lambda x.\overline{a}(\overline{b}\,x)]\!]$$

The Karoubi Envelope *of $\mathcal{M}$, denoted by $\mathcal{K}(\mathcal{M})$, is defined by the following data:*

$$\begin{array}{rcl} \textit{Objects} & : & \{\, a \in \mathcal{M} \,|\, a = a \circ a \,\} \\ \textit{Arrows} & : & \textit{Hom}(a, b) = \{\, (b, f, a) \,|\, b \circ f \circ a = f \,\} \end{array}$$

The identity for an object a is $id_a = (a, a, a)$ and the composition of $(b, f, a) : a \longrightarrow b$ and $(c, g, b) : b \longrightarrow c$ is given by $(c, g \circ f, a)$.

Theorem 3.2.4 *$\mathcal{K}(\mathcal{M})$ is a cartesian closed category.*

Proof Essentially this is done by making the definitions below and then verifying the axioms of definition 3.2.1. As in [Bar84] let $[\![J]\!] = [\![\lambda x.\lambda y.y]\!]$.

Terminal objects : Define $1_{\mathcal{K}} = [\![J]\!]$. The unique arrow from any other a to $1_{\mathcal{K}}$ is given by $\bigcirc_a = (1_{\mathcal{K}}, J, a)$.

Product objects : for any $a, b \in \mathcal{M}$ define

$$a \times b = [\![\lambda x.\lambda y.y\,(\overline{a}\,(x\,K))\,(\overline{b}\,(x\,J))]\!]$$

with projections

$$\pi_{ab} = [\![\lambda x.\overline{a}\,(x\,K)]\!] \quad \textit{and} \quad \pi'_{ab} = [\![\lambda x.\overline{b}\,(x\,J)]\!]$$

If $f : a \to b$ and $g : a \to c$ then

$$\langle f, g \rangle = [\![\lambda x.\lambda y.y(\overline{b}\,(f\,(\overline{a}\,x)))\,(\overline{c}\,(g\,(\overline{a}\,x)))]\!]$$

Exponentials : for any $a, b \in \mathcal{M}$ the exponential or function space object is given by

$$(a \to b) = [\![\lambda f.\overline{b} \circ f \circ \overline{a}]\!]$$

and

$$ev_{ab} = [\![\lambda x.\pi_{b^a \times a}(x)\,\pi'_{b^a \times a}(x)]\!]$$

If $f : a \times b \to c$ then the exponential transpose of f is given by

$$f^* = [\![\lambda x.\lambda y.f\,(\lambda z.z\,(\overline{a}\,x)\,(\overline{b}\,y))]\!]$$

With these definitions the axioms of a cartesian closed category can now be verified by direct calculation [Koy82]. □

The objects of $\mathcal{K}(\mathcal{M})$ can be considered to be simple *types*. The *elements* or values of a type a are defined to be the fixed points of a. If m is a fixed point of a then this is written as $m : a$.

Two examples of objects of $\mathcal{K}(\mathcal{M})$ are $U \overset{def}{=} [\![\lambda x.x]\!]$ and $J \overset{def}{=} [\![\lambda x.\lambda y.y]\!]$. As seen already J is a terminal object in $\mathcal{K}(\mathcal{M})$ and U we call a *universe object.* The reason for this is that the arrows in $Hom(J, U)$ are in 1:1 correspondence with the elements of $\mathcal{M}$. This correspondence is achieved through the following map:

$$\forall\ m \in \mathcal{M} \quad m \mapsto k \cdot m$$

where $k \cdot m : [\![J]\!] \to [\![U]\!]$ in $\mathcal{K}(\mathcal{M})$. Furthermore, a retract is defined as follows:

Definition 3.2.5 *A* retract *in a category C is a map $f : A \to B$ such that there exists an inverse $g : B \to A$ satisfying $f \circ g = Id_B$.*

It can be shown that every object in $\mathcal{K}(\mathcal{M})$ is a retract of U [LS86] which can be seen by defining the maps $a : a \to U$ and $a : U \to a$. The object $[\![U]\!]$ is significant here because $[\![U]\!]$ is the *universe* from which all the values of types are drawn. Before we can model algebras within this simple model of types some additional structure is required and this we do in the next chapter.

Chapter 4

Models of Equational Theories In Lambda Algebras

The previous chapter dealt with the construction of a cartesian closed categories from arbitrary λ algebras. Cartesian closed categories do possess all finite products but to express functors like X_Σ sums are needed. What is also needed are colimits of ω chains. This is so that solutions to the recursive domain equations like those at the end of chapter 2 are guaranteed to exist. For the case of an arbitrary λ algebra this is not so easy. Indeed there are two problems here: the first is that there does not appear to be an appropriate general construction for the sum of two objects within the Karoubi envelope of an arbitrary λ algebra and the second is that an arbitrary ω chain may not possess a colimit in a cartesian closed category in which only finite colimits may be constructed.

In this chapter we construct models of S sorted theories within *continuous* λ algebras with the prime purpose of seeking out the conditions under which such models can be explicitly constructed. Starting in section 4.1 we give the construction of a category, in the style of $\mathcal{K}(\mathcal{M})$ from the elements of a continuous λ algebra $\mathcal{M}$, which also includes finite sums. In section 4.2 we show that anarchic algebras can be constructed within this category by the methods outlined in chapter 2.4 and then in section 4.3 we look at the problem of finding solutions to systems of equations and in particular to deriving recursive functions for the operations in Σ from sets of rewrite rules obtained from the original Σ equations.

4.1 The Category $\mathcal{C}(\mathcal{M})$

Starting from $\mathcal{K}(\mathcal{M})$ we now wish to define algebras in $\mathcal{K}(\mathcal{M})$ by the use of the techniques outlined in section 2.3. Since $\mathcal{K}(\mathcal{M})$ is cartesian closed it has all finite products but there is a problem in dealing with sums in a similar fashion.

To illustrate the difficulty in defining the sum of two objects in $\mathcal{K}(\mathcal{M})$ for an arbitrary λ algebra $\mathcal{M}$ consider the case where $\mathcal{M}$ is the λ algebra given by (equivalence classes of) pure λ terms modulo the conversion rules. Disjoint sum [Sco76] is usually defined by the following equivalence class:

$$a + b \overset{def}{=} [\lambda x.\pi(x)\,\langle K, a\,\pi'(x)\rangle\,\langle J, b\,\pi'(x)\rangle]$$

Then

$$a + b \circ a + b = a + b$$

in $\mathcal{M}$ if an only if $a + b \circ a + b \in [a + b]$, that is,

$$a + b \circ a + b =_{CNV} a + b$$

So we have

$$\begin{aligned} a + b \circ a + b &\overset{def}{=} \lambda x.(a + b)\,((a + b)\,x) \\ &=_{CNV} \lambda x.\pi(\pi(x)\,\pi(x)\,\langle K, a\,\pi'(x)\rangle\,\langle J, b\,\pi'(x)\rangle) \\ &\qquad \langle K, a\,\pi'(\pi(x)\,\langle K, a\,\pi'(x)\rangle\,\langle J, b\,\pi'(x)\rangle)\rangle \\ &\qquad \langle J, b\,\pi'(\pi(x)\,\langle K, a\,\pi'(x)\rangle\,\langle J, b\,\pi'(x)\rangle)\rangle \end{aligned}$$

which is in head normal form and is different from

$$\lambda x.\pi(x)\,\langle K, a\,\pi'(x)\rangle\,\langle J, b\,\pi'(x)\rangle$$

Convertibility is not strong enough to show that the sum $a + b$ defined above is an object of the Karoubi envelope constructed from the pure λ terms.

We wish to construct a category from arbitrary λ algebras in which the sum of two objects $a + b$ is itself an object. A second criterion is that such a category posseses colimits of ω chains. We begin by giving a definition of *closures* and *mappings* between closures which are intended to be more general than the objects and arrows of definition 3.2.3. First recall that $m : a$ means that m is an element of the set $\{m \mid a \cdot m = m\}$, that is, the set of elements of type a. Now given a λ algebra $\mathcal{M}$ we define the following two sets:

1. $Clos(\mathcal{M}) = \{a \mid \forall m \in \mathcal{M}. (a \circ a)m = m \text{ iff } am = m\}$ which is called the set of *closures* of $\mathcal{M}$;

2. $Map(a,b) = \{(b,f,a) \mid \forall m : a, \; b(fm) = fm\}$ which is called the set of *mappings* in $\mathcal{M}$.

The composition of two mappings (b,f,a) and (c,g,b) is given by

$$(c,g,b) \circ (b,f,a) = (c, [\![\lambda x.\overline{g}(\overline{f}\,x)]\!], a)$$

while the identity mapping for a closure a is again (a,a,a).

An intuition behind the definition of the category $\mathcal{K}(\mathcal{M})$ is that each object is its own identity function which, because of the axioms for a category ((C1) and (C2) of definition 3.2.1) is idempotent. In weakening the definition each object is still intended to have this property except now a closure a is to be thought of as an identity if it has the same set of fixed points as $a \circ a$. With this definition of identity mapping for a closure a the old definition of arrows (3.2.3) is no longer adequate to prove that $a \circ a = a$, and so we adopt a slightly weaker definition[1]. The definition of mappings above also implies the following simple identity which is characteristic of arrows in $\mathcal{K}(\mathcal{M})$:

$$\forall m \in \mathcal{M} \quad a(m) = m \Rightarrow (b \circ f \circ a)(m) = f(m)$$

We still do not have a category, for example, the axiom (C5) is not valid, and so an equivalence relation on mappings is imposed to recover the category structure. The following lemma now holds.

Lemma 4.1.1 ([Sco76]) *If $\mathcal{M}$ is a λ algebra then for any $m \in \mathcal{M}$*

$$(a+b)(m) = m \quad \textit{iff} \quad (a + b \circ a + b)(m) = m$$

[1]The idea that an element $f \in \mathcal{M}$ defines a mapping from one subset A of $\mathcal{M}$ to another B if for all $a : A \; f(a) : B$ can also be found in [MPS86]

For our definition of $\_+\_$ case analysis is given by:

$$[f,g] \overset{def}{\equiv} [\![\lambda u.(\pi\,(u\,K))\,(f\,(u\,J))\,(g\,(u\,J))]\!]$$

and the injections by

$$\begin{aligned} inj_a &\overset{def}{\equiv} [\![\lambda u.\langle K, a\,(\pi'\,u)\rangle]\!] \\ inj_b &\overset{def}{\equiv} [\![\lambda u.\langle J, b\,(\pi'\,u)\rangle]\!] \end{aligned}$$

To see that closures and mappings do not form a category consider again the λ algebra of pure λ terms modulo convertibility and the axiom $id_b \circ f = f$ for any $(b,f,a) : a \to b$. We have

$$(b,b,b) \circ (b,f,a) \;=\; (b, \lambda z.b\,(f\,z), a)$$

but in general $\lambda z.b\,(f\,z) \neq_{CNV} f$. For any $m : a$ however, we know by the definition above that $b\,(f\,m) = f(m)$ and also that $f(a(m)) = f(m)$.

From the definition of closures it thus seems reasonable that we should have for any $(b,[\![f]\!],a),(b,[\![g]\!],a) \in Hom(a,b)$, $(b,[\![f]\!],a) = (b,[\![g]\!],a)$ if and only if

$$\forall m : a, \quad [\![f]\!] \cdot m = [\![g]\!] \cdot m \tag{4}$$

where $[\![\_]\!]$ interprets closed λ terms in $\mathcal{M}$. Taking the quotient of each set $Map(a,b)$ with respect to this axiom should be adequate to regain the category structure, but if we also wish to retain cartesian closure (as we do) then this is not good enough. If a category is cartesian closed then for any objects A, B and C,

$$Hom(A \times B, C) \cong Hom(A, B \to C) \tag{5}$$

but if we simply use the axiom (4) to take the quotient then in the isomorphism we have considered two maps to be extensionally equal over the values of type $A \times B$ on the left hand side while only taking into account the type A on the right hand side. Thus, by using just the equation (4) the isomorphism (5) will no longer hold. Let $\mathcal{M}$ be the λ algebra of pure terms modulo conversion and $(c, f, a \times b)$ and $(c, g, a \times b)$ two arrows such that for all elements $\langle m, n\rangle$ of type $a \times b$, $f(\langle m, n\rangle) = g(\langle m, n\rangle)$. By definition 3.2.3 $f^* = \lambda x.\lambda y.f\langle x, y\rangle)$ which means for any $m : a$, $f^*(m) = \lambda y.f(\langle m, y\rangle)$ and similarly $g^*(m) = \lambda y.g(\langle m, y\rangle)$, but to show $\lambda y.f(\langle a, y\rangle) = \lambda y.g(\langle a, y\rangle)$ we need to show $f(\langle m, y\rangle) = g(\langle m, y\rangle)$ which in general may not be provable by conversion alone because of the need to take into account the free variable y. In general, $\forall m : a\ \forall n : b\ f(\langle m, n\rangle) = g(\langle m, n\rangle)$ does not imply that $f^*(a) =_{CNV} g^*(a)$.

To get a cartesian closed category from the closures and mappings we adopt a technique of J. Zucker (see [Bar84] appendix A) called *extensional collapse.* Extensional collapse is a technique for obtaining an extensional model of the simply typed λ calculus from an arbitrary model of the simply typed λ calculus. Let T be a set of types such that:

1. $0 \in T$ where 0 is a (unique) base type;

2. $\tau_1, \tau_2 \in T$ implies $\tau_1 \to \tau_2 \in T$.

A model of the simply typed λ calculus is a pair

$$\mathcal{M}_T = \langle \{M_\tau\}_{\tau \in T}, \{\cdot_{\sigma\tau}\}_{\sigma\tau \in T}\rangle$$

where $\{M_\tau\}_{\tau \in T}$ is a T indexed family of sets such that M_τ is the set of values for the type τ and $\cdot_{\sigma\tau}$ is the application operation such that if $f \in M_{\sigma \to \tau}$ and $a \in M_\sigma$ then $f \cdot_{\sigma\tau} a \in M_\tau$.

Now define an equivalence relation on the set $\{M_\tau\}_{\tau \in T}$ as follows [Bar84]:

1. $\forall\ x, y \in M_0 \quad x \equiv_0 y \quad \text{iff} \quad x = y$

2. $\forall\ x, y \in M_{\sigma \to \tau} \quad x \equiv_\tau y \quad \text{iff} \quad \forall\ z, z' \in M_\sigma.\,(z \equiv_\sigma z' \Rightarrow x \cdot z \equiv_\tau y \cdot z')$

Put $\mathcal{N}_\tau = \{x \in \mathcal{M}_\tau \mid x \equiv_\tau x\}$, the set of well behaved terms of $\mathcal{M}$. The *extensional collapse*, $\mathcal{M}_T^E$ of $\mathcal{M}_T$ is defined as:

$$\mathcal{M}_T^E = \langle \{\mathcal{N}_\tau / \equiv_\tau\}_{\tau \in T}, \cdot_{\sigma\tau}\rangle$$

where if $x \in \mathcal{N}_{\sigma \to \tau}$ and $y \in \mathcal{N}_\sigma$ then $[x] \cdot_{\sigma\tau} [y] = [x \cdot_{\sigma\tau} y]$

We start by considering the following subset of all the closures which we call the *finite closures* of $\mathcal{M}$:

1. $[\![\Omega]\!]$, $[\![J]\!] = 1_C$ and $[\![U]\!]$ are finite (basic) closures;

2. if a and b are finite closures of $\mathcal{M}$ then so are $a \times b$, $a + b$ and $a \to b$.

Definition 4.1.2 *Let a and b be two finite closures and $f : a \to b$ and $g : a \to b$ be any two mappings. Define the relation $\equiv_b$ by induction on the finite closures of $\mathcal{M}$ as follows:*

1. if b is a basic closure of $\mathcal{M}$ then

$$(b, f, a) \equiv_b (b, g, a) \quad \textit{iff} \quad \forall\, x : 1_C \to a, \quad f \circ x = g \circ x$$

2. If b is $\sigma_1 \times \sigma_2$ then let $f = \langle f_1, f_2 \rangle$ and $g = \langle g_1, g_2 \rangle$ and define

$$(\sigma_1 \times \sigma_2, f, a) \equiv_{\sigma_1 \times \sigma_2} (\sigma_1 \times \sigma_2, g, a) \quad \textit{iff} \quad f_1 \equiv_{\sigma_1} g_1 \quad \textit{and} \quad f_2 \equiv_{\sigma_2} g_2$$

3. If b is $\sigma_1 \to \sigma_2$ then

$$(\sigma_1 \to \sigma_2, f, a) \equiv_{\sigma_1 \to \sigma_2} (\sigma_1 \to \sigma_2, g, a)$$

iff for all maps

$$x : 1_{C(\mathcal{M})} \to \sigma_1, \quad ev \circ \langle f, x \rangle \equiv_{\sigma_2} ev \circ \langle g, x \rangle$$

Lemma 4.1.3 *$\equiv_\tau$ is an equivalence relation.*

Proof That $\equiv_\tau$ is reflexive and symmetric follows immediately from the definition 4.1.2. Transitivity can be shown by induction on the type τ. The only interesting case is when $f, g, h : a \to c^b$; and $f \equiv_{b \to c} g$ and $g \equiv_{b \to c} h$. By 4.1.2(3) we have for all $a : b \to c$

$$ev \circ \langle f, a \rangle \equiv_c ev \circ \langle g, a \rangle \equiv_c ev \circ \langle h, a \rangle$$

by the induction assumption and so for all $a : b \to c$ $f \equiv_{b \to c} h$. □

We then have the following:

Lemma 4.1.4 *Let $\mathcal{M}$ be a continuous λ algebra and let F be an element of $\mathcal{M}$ such that if a is any finite closure of $\mathcal{M}$ then $F_o(a)$ is a finite closure of $\mathcal{M}$. Then $Y\,F$ is a closure of $\mathcal{M}$.*

Proof let ω be the set of natural numbers. We need to show that

$$\forall m \in \mathcal{M} \quad (Y(F) \circ Y(F))(m) = m \text{ iff } Y(F)(m) = m$$

Note that $m = \bigsqcup\{m\}$. Then by the continuity of application and $\circ$:

$$\begin{aligned}(\bigsqcup\{F^n(\bot) \mid n \in \omega\} \circ \bigsqcup\{F^n(\bot) \mid n \in \omega\})\bigsqcup\{m\} &= \bigsqcup\{(F^n(\bot) \circ F^n(\bot))(m)\} \\ &= m\end{aligned}$$

But for each $n \geq 0$ $[\![\Omega]\!] = \bot$ is a finite retract and by the premise $F^n(\bot)$ is also a finite retract. By the definition of $Clos(\mathcal{M})$

$$(F^n(\bot) \circ F^n(\bot))(m) = m \text{iff} F^n(\bot)(m) = m$$

But then

$$\bigsqcup\{(F^n(\bot) \circ F^n(\bot))(m)\} = m \text{ iff } \bigsqcup\{F^n(\bot)(m)\} = m$$

□

If $F \in \mathcal{M}$ is a function mapping closures to closures then by lemma 4.1.4 $Y(F)$ is also a closure and referred to as a *limit closure.*

Definition 4.1.5 *If $\mathcal{M}$ is a λ algebra then define $C(\mathcal{M})$ to be the category given by the following data:*

1. *The objects of $C(\mathcal{M})$ are the finite closures of $\mathcal{M}$. If $F \in \mathcal{M}$ is any function taking finite closures of $\mathcal{M}$ to finite closures of $\mathcal{M}$ then $Y(F)$ is also an object of $C(\mathcal{M})$.*
2. *$Hom(a, b)$ is the set $Map(a, b)/\equiv_b$.*

We now have:

Theorem 4.1.6 *$\mathcal{C}(\mathcal{M})$ is a cartesian closed category.*

Proof As in [Koy82] make the following definitions:

1. The terminal object of $\mathcal{C}(\mathcal{M})$ is the same as for $\mathcal{K}(\mathcal{M})$, i.e $1_{\mathcal{C}} = [\![\lambda x.\lambda y.y]\!]$. The arrow from any other a to $1_{\mathcal{C}}$ is given by $\bigcirc_a : a \rightarrow 1_{\mathcal{C}}$ is $(1_{\mathcal{C}}, [\![\lambda x.\lambda y.y]\!], a)$.

2. Product objects are the same as for $\mathcal{K}(\mathcal{M})$ with projections

$$\pi_{a\times b} \overset{def}{\equiv} (a, [\![\lambda x.\overline{a}(x\,K)]\!], a\times b)$$
$$\pi'_{a\times b} \overset{def}{\equiv} (b, [\![\lambda x.\overline{b}(x\,J)]\!], a\times b)$$

   If $(b,f,a) : a \rightarrow b$ and $(c,g,a) : a \rightarrow c$ then $\langle (b,f,a),(c,g,a)\rangle : a \rightarrow b\times c$ is given by

$$(b\times c, [\![\lambda x.\lambda y.y\,\overline{b}(\overline{f}\,(\overline{a}\,x))\,\overline{c}(\,\overline{g}\,(\overline{b}\,x))]\!], a)$$

   .

3. The exponential object associtated with a and b is given by

$$(a \rightarrow b) \;=\; [\![\lambda f.\overline{b}\circ f\circ\overline{a}]\!]$$

   and the exponential transpose of $f : a\times b \rightarrow c$ is given by

$$f^* \;=\; ((b\rightarrow c), [\![\lambda x.\lambda y.\overline{f}(\lambda z.z\,x\,y)]\!], a)$$
$$ev_{ab} \;=\; (b, [\![\lambda x.(\overline{a}\rightarrow\overline{b})\,(\pi\,x)\,(\overline{a}\,(\pi'\,x))]\!], (a\rightarrow b)\times a)$$

The axioms for a cartesian closed category from definition 3.2.1 can now be shown to hold by using the definition of closure and mapping.

For example, equations 4 and 5 of 3.2.1 can be verified as follows. Let $f : a\times b \rightarrow c$ and $g : a \rightarrow (c^b)$ be arrows in $\mathcal{C}(\mathcal{M})$.
Then for f:

$$ev\circ f^*\times Id_b \;\equiv_c\; (c, [\![\lambda u.f\,(\lambda x.x\,(\overline{a}\,(u\,K))\,(\overline{b}\,(u\,J)))]\!], a\times b)$$

by conversion and applying the definition of $\equiv_\tau$. Now if $m : a \times b$ then $a\,(m\,K) \equiv_a m\,K$ and $b(m\,J) \equiv_b m\,J$. Then

$$\begin{aligned}
ev \circ f^* \times Id_b(m) &\equiv_c (c, [\![(\lambda u.f\,(\lambda x.x\,(\overline{a}\,(u\,K))\,(\overline{b}\,(u\,J))))\,m]\!], a \times b) \\
&\equiv_c (c, [\![f\,m]\!], a \times b) \quad \text{by definition of closures and } (\beta) \\
&\equiv_c (c, f, a \times b) \quad \text{by definition of mapping}
\end{aligned}$$

and so $ev \circ f^* \times Id \equiv_c f$.

Similarly for g:

$$\begin{aligned}
(ev \circ g \times Id)^* &\equiv_{b \to c} (\lambda u.(\lambda x.(\pi\,x)\,(\pi'\,x))\,(\lambda z.z\,(g\,(\overline{a}\,(u\,K))))\,(\overline{b}\,(u\,J)))^* \\
&\equiv_{b \to c} (\lambda u.(g\,(\overline{a}\,(u\,K)))\,(\overline{b}\,(u\,J)))^* \\
&\equiv_{b \to c} \lambda x.\lambda y.g\,(a\,x)\,(b\,x) \\
&\equiv_{b \to c} g
\end{aligned}$$

again using the definition of closures and mappings. □

Intuitively the morphisms of $\mathcal{C}(\mathcal{M})$ are equal if they are extensionally equal as functions their domain rather than as functions over all of $\mathcal{M}$, that is, a mapping $f : a \to b$ is *total* taking every fixed point of a to a fixed point of b.

We now have sums but in general coproducts do not exist which can be shown by considering the diagram in figure 4. Call a retract *a strict* if $a \cdot [\![\Omega]\!] = [\![\Omega]\!]$ in $\mathcal{M}$. The definition of $\mathcal{C}(\mathcal{M})$ requires all arrows to be considered as total functions from $a + b$ to $a \times b$ but since $a \times b$ is not a strict retract but the arrow $[f, g]$ is strict then

$$[f, g]\,\Omega = \Omega \neq (a \times b)\,\Omega$$

and so $[f, g]\,\Omega$ is not of type $a \times b$. The consequence of this is that the usual distributivity of products over sums:

$$\begin{aligned}
(A + B) \times C &\neq (A \times C) + (B \times C) \\
A \times (B + C) &\neq (A \times B) + (A \times C)
\end{aligned} \qquad (6)$$

$$\begin{tikzcd}
 & a+b \arrow[dd, "{[f,g]}"] & \\
a \arrow[ur, "inj_a"] \arrow[dr, "f"'] & & b \arrow[ul, "inj_b"'] \arrow[dl, "g"] \\
 & a \times b &
\end{tikzcd}$$

Figure 4: Sums and Products in $\mathcal{C}(\mathcal{M})$

which one might have hoped for, does not hold in $\mathcal{C}(\mathcal{M})$, but in general there are maps:

$$\begin{aligned}
(A+B)\times C &\rightarrow (A\times C)+(B\times C) \\
A\times(B+C) &\rightarrow (A\times B)+(A\times C)
\end{aligned} \tag{7}$$

The problem is that the inverses are not defined on $[\![\Omega]\!]$ and so are not arrows in $\mathcal{C}(\mathcal{M})$. The following, however, does hold:

Lemma 4.1.7 *Let a and b be any objects of $\mathcal{C}(\mathcal{M})$ and $c_\perp$ a strict retract, that is $[\![\Omega]\!] : c_\perp$. Then there is a* unique *map h making the diagram;*

$$\begin{tikzcd}
 & a+b \arrow[dd, "{[f,g]}"] & \\
a \arrow[ur, "inj_a"] \arrow[dr, "f"'] & & b \arrow[ul, "inj_b"'] \arrow[dl, "g"] \\
 & c_\perp &
\end{tikzcd}$$

commute.

Proof Firstly let $f : a \rightarrow c_\perp,\ \ g : a \rightarrow c_\perp$ and define h by $[f,g]$.
Now $[f,g]\cdot[\![\Omega]\!] = [\![\Omega]\!]$ since $[f,g]$ is strict and since $c_\perp$ is also strict then $(c_\perp, [f,g], a+b)$ is a map in $\mathcal{C}(\mathcal{M})$.

We can show that the diagram commutes by using (β) reduction and definition of disjoint sum above. The uniqueness of h follows from the definition of $\equiv_\tau$ and fixed point induction (see [Sto77]). $\square$

If A, B and C are therefore strict retracts then the isomorphisms in (6) do hold.

4.2 Anarchic Algebras in $\mathcal{C}(\mathcal{M})$

Algebras in $\mathcal{C}(\mathcal{M})$ may be defined using the techniques of chapter 2.3 but since $\mathcal{C}(\mathcal{M})$ has no initial object we dualise the basic lemma and use colimits of ω^{op} chains rather than limits of ω chains. To do this we need to ensure that the colimits exists in $\mathcal{C}(\mathcal{M})$. The idea below is to define a class of functors for which the colimit of an ω^{op} chain can be constructed in $\mathcal{C}(\mathcal{M})$.

Definition 4.2.1 *Let $\Psi : \mathcal{C}(\mathcal{M}) \rightarrow \mathcal{C}(\mathcal{M})$ be an endofunctor. Then Ψ is* Representable *if there exist polynomials Ψ_O and Ψ_A over $\mathcal{M}$ such that*

1. *for any object a of $\mathcal{C}(\mathcal{M})$ $\Psi(a) = \Psi_O(a)$*
2. *for any arrow (b, f, a) of $\mathcal{C}(\mathcal{M})$ $\Psi((b, f, a)) = (\Psi_O(b), \Psi_A(f), \Psi_O(a))$*
3. *Ψ_O and Ψ_A are representable.*

Also we make use of the following property.

Lemma 4.2.2 *If F_1 and F_2 are representable endofunctors over $\mathcal{C}(\mathcal{M})$ then so is their composite $F_1 \circ F_2$.*

Proof Let F_1 and F_2 be represented by $\varphi = (\varphi_O, \varphi_A)$ and $\psi = (\psi_O, \psi_A)$ respectively. Let $a \in Obj(\mathcal{C}(\mathcal{M}))$ and put $a' = \varphi_O(a) = F(a)$. Then

$$\begin{aligned} F_1 \circ F_2\, a &= \psi_O\, a' \quad \text{by representability} \\ &= \psi_O \varphi_O\, a \end{aligned}$$

which is representable [Bar84]. Likewise for any arrow (b, f, a) of $\mathcal{C}(\mathcal{M})$

$$\begin{aligned} F_1 \circ F_2((b,f,a)) &= F_1(\varphi_O(b), \varphi_A(f), \varphi_O(a)) \\ &= (\psi_O \circ \varphi_O(b), \psi_A \circ \varphi_A(f), \psi_O \circ \varphi_O(a)) \end{aligned}$$

where $\psi_A \circ \varphi_A(f)$ is representable [Bar84]. □

Lemma 4.2.3 *Let a be an object of $\mathcal{C}(\mathcal{M})$.*

1. *The functor $\_ \times a$ taking every object b to $b \times a$ and every map $(c, f, b) : b \to c$ to $(c \times a, f \times id_a, b \times a)$ is representable.*

2. *The functor $\_ + a$ taking every object b to $b + a$ and every map $f : b \to c$ to $(c + a, [f \circ inj_c, inj_a], b + a)$ is representable.*

Proof Make the following definitions:

$$\begin{aligned} \_ \times a &\overset{def}{\equiv} [\![\lambda b.\lambda x.\lambda y.y\,(b\,(x\,K))\,(\overline{a}\,(x\,J))]\!] \\ \_ + a &\overset{def}{\equiv} [\![\lambda b.\lambda x.(x\,K)\,\langle K, b\,(x\,J)\rangle\,\langle J, \overline{a}\,(x\,J)\rangle]\!] \end{aligned}$$

Now verify that $[\![\lambda b.\lambda x.\lambda y.y\,(b\,(x\,K))\,(\overline{a}\,(x\,J))]\!]$ does satisfy the properties for a functor. Let (c, g, b) and (b, g, a) be two maps in $\mathcal{C}(\mathcal{M})$. Then

$$g \circ f \times a = [\![\lambda y.y\,(\overline{g}\,(\overline{f}\,(x\,K)))\,(\overline{a}\,(x\,J))]\!]$$

while

$$\begin{aligned} g \times a \circ f \times a &= [\![\lambda x.\lambda y.y\,(g\,(f\,(x\,K)))\,(a\,(a\,(x\,J)))]\!] \\ &\equiv_{c\times a} [\![\lambda x.\lambda y.y\,(g\,(f\,(x\,K)))\,(a\,(x\,J))]\!] \end{aligned}$$

since for any objects a of $\mathcal{C}(\mathcal{M})$ $\lambda x.a\,(a\,x) = a \circ a \equiv_{\tau} a$. The proof that the representation of $\_ + a$ is a functor follows along similar lines.
□

So far we have considered any λ algebra but below we need to restrict the choice in order to guarantee limits of chains. Consider an ω^{op} diagram in $\mathcal{C}(\mathcal{M})$, and a functor F represented by the pair (F_O, F_A).

$$\begin{tikzcd} a_0 & a_1 \arrow[l, "F_O(a_0)"'] & a_2 \arrow[l, "F_O(F_O(a_0))"'] & \cdots \end{tikzcd}$$

If $a_0 = \perp$ then the least fixed point for the chain $\{F_o^n(\perp)\}_{n\in\omega}$ in any continuous λ algebra is, by 3.1.19, $Y\,F_O$. This turns out also to be a colimit for the ω^{op} chain.

Theorem 4.2.4 *Let F be a representable endofunctor over $\mathcal{C}(\mathcal{M})$ which is represented by (F_O, F_A) and further that $F_A = F_O$. If*

$$\Delta^{op} = \langle F_O^n(1_\mathcal{C}), F_A^n(\bigcirc_{(YF)})\rangle$$

is an ω^{op} chain in $\mathcal{C}(\mathcal{M})$ then $(Y\,F_O)$ is a colimit for the diagram Δ^{op}.

Proof First we show by induction on n that $\langle Y(F_O), \mu_n\rangle$, where

$$\mu_n = (F_O^n(1_\mathcal{C}), F_A^n(\bigcirc_{(YF)}), Y\,F_O)$$

is a cone for Δ^{op}, i.e that for each n the diagram

$$\begin{tikzcd} \llbracket F_O^n\, 1_\mathcal{C} \rrbracket & & \llbracket F_O^{n+1}\, 1_\mathcal{C} \rrbracket \arrow[ll, "F_A^n \bigcirc_{1_\mathcal{C}}"'] \\ & & \llbracket Y\,F_O \rrbracket \arrow[ull, "F_A^n \bigcirc_{YF}"] \arrow[u, "F_A^{n+1} \bigcirc_{YF}"'] \end{tikzcd}$$

commutes. Recall that the map $\bigcirc_a : a \to 1_\mathcal{C}$ is simply $(1_\mathcal{C}, \llbracket J \rrbracket, a)$.
For $n = 0$;

$$\begin{aligned} (1_\mathcal{C}, \llbracket J \rrbracket, F_O\, 1_\mathcal{C}) \circ (F_O\, 1_\mathcal{C}, \llbracket \overline{F_A}\, J \rrbracket, Y\,F_O) &= (1_\mathcal{C}, \llbracket \lambda u. J\,((\overline{F}\,J)\,u) \rrbracket, Y\,F_O) \\ &= (1_\mathcal{C}, \llbracket J \rrbracket, Y\,F_O) \end{aligned}$$

For the induction case $n + 1$;

$$\begin{aligned}
&(F_O^n\,1_C, [\![\overline{F_A}^n\,J]\!], F_O^{n+1}\,1_C) \circ (F_O^{n+1}\,1_C, [\![\overline{F_A}^{n+1}\,J]\!], Y\,F_O) \\
&\quad = (F_O^n\,1_C, [\![\overline{F_A}^n(J \circ (F_A\,J))]\!], Y\,F_O) \text{ by the functoriality of } F \\
&\quad = (F_O^n\,1_C, [\![\overline{F_A}^n\,J]\!], Y\,F_O) \\
&\quad = \mu_n
\end{aligned}$$

Therefore $(Y\,F_O, \mu_n)$ is a cone for Δ^{op}.

Now consider any other cone $\langle B, \nu_n \rangle$ for Δ^{op}. For each n we have,

$$\begin{tikzcd}
 & B \arrow[dl, "\nu_n"'] \arrow[d, "\nu_{n+1}"] \\
F_O^n(\bigcirc_{F\,1_C}) & F_O^n\,1_C \arrow[l, "F_A^n\,1_C"]
\end{tikzcd}$$

For each n:

$$\begin{aligned}
\nu_n &= F^n(\bigcirc_{F(1_C)}) \circ \nu_{n+1} \\
&= F^n(\bigcirc_{F(1_C)}) \circ (F^{n+1}(\bigcirc_{F(1_C)}) \circ \nu_{n+2}) \\
&= F^n(\bigcirc_{F(1_C)}) \circ (F^{n+1}(\bigcirc_{F(1_C)}) \circ F^{n+2}(\bigcirc_{F(1_C)}) \circ \ldots
\end{aligned}$$

and so each ν_n is the infinite composition

$$\nu_n = F^n(\bigcirc_{F(1_C)}) \circ F^{n+2}(\bigcirc_{F(1_C)}) \circ \ldots$$

that is

$$\begin{aligned}
\nu_n &= F^n(\bigcirc_{F(1_C)}) \circ F^{n+1}(\bigcirc_{F(1_C)}) \circ \ldots \\
&= F^n(\bigcirc_{F(1_C)} \circ F(\bigcirc_{F(1_C)}) \circ \ldots) \quad \text{by the} \mathit{functoriality} \text{ of } F \\
&= F^n(\bigcirc_{F(1_C)}) \quad \text{by definition of } \bigcirc_{F(1_C)}
\end{aligned}$$

Now define α as $(Y\;F_O, Y\;F_O, B)$. It is easy to show α is an arrow of $\mathcal{C}(\mathcal{M})$ by applying the definitions in 4.1.5. Now we show that for each

n the diagram

$$\begin{tikzcd}
& & B \arrow[dll, "F^n_A(\bigcirc_{F(1_C)})"'] \arrow[dd, "\alpha"] \\
F^n_O 1_C & & \\
& & (Y\ F) \arrow[ull, "F^n_A(\bigcirc_{F(1_C)})"]
\end{tikzcd}$$

commutes.

For $n = 0$.

$$\begin{aligned}
\bigcirc_{F(1_C)} \circ \alpha &= (1_C, [\![J]\!], Y\ F_O) \circ (Y\ F_O, Y\ F_O, B) \\
&= (1_C, [\![J \circ (Y\ \overline{F})_O]\!], B) \\
&= (1_C, [\![J]\!], B) \\
&= \bigcirc_{F(1_C)}
\end{aligned}$$

and for $n = k$

$$\begin{aligned}
&F^k(\bigcirc_{F(1_C)}) \circ (Y\ F_O, Y\ F_O, B) \\
&\qquad = (F^k_O(1_C), F^k_A(\bigcirc_{F(1_C)}), Y\ F_O) \circ (Y\ F_O, Y\ F_O, B) \\
&\qquad = (F^k_O(1_C), F^k_A(\bigcirc_{F(1_C)}) \circ Y\ F_O, B) \\
&\qquad = (F^k_O(1_C), F^k_A(\bigcirc_{F(1_C)}) \circ F^k_O(Y\ F_O), B) \\
&\qquad = (F^k_O(1_C), F^k_A(\bigcirc_{F(1_C)} \circ Y\ F_O), B) \\
&\qquad\quad \text{by the functoriality } F \\
&\qquad\quad \text{and the premise that } F_A = F_O \\
&\qquad = (F^k_O(1_C), F^k_A(\bigcirc_{F(1_C)}), B) \quad \text{by definition of } \bigcirc_{F(1_C)}
\end{aligned}$$

and so for any other object B of $\mathcal{C}(\mathcal{M})$ there exists an arrow $\alpha : B \to Y\ F_O$.

Finally we show α is unique. To this end assume α' is any other arrow in $\mathcal{C}(\mathcal{M})$ making the diagram commute. First some items that need noting. If $b{:}B$ and α' makes the diagram above commute then we must have $(Y\ F)(\alpha'(b)) = \alpha'(b)$ by the definition of arrows in $\mathcal{C}(\mathcal{M})$.

Furthermore for every $n \in \mathcal{N}$, where $\mathcal{N}$ is the set of natural numbers, we have:

$$F^n(\bigcirc_{1_C}) \circ \alpha'(b) = F^n(\bigcirc_{1_C})(b)$$

by the fact that α' makes the above diagram commute. Now we can transform this equality as follows. Let $\perp : 1_{\mathcal{C}} \to \perp$ where $[\![\Omega]\!] = \perp \in \mathcal{M}$. For any other arrow $f : 1_{\mathcal{C}} \to \perp$, $\perp \circ f 1_{\mathcal{C}} = \perp$ by fact 3.1.12 and so $\perp$ is also a terminal object in $\mathcal{C}(\mathcal{M})$. Thus for every $n \in \mathcal{N}$:

$$F^n(\perp) \circ F^n(\bigcirc_{1_C}) \circ \alpha'(b) = F^n(\perp) \circ F^n(\bigcirc_{1_C})(b) \qquad (8)$$

which by the functoriality of F and fact 3.1.12 gives:

$$F^n(\perp) \circ \alpha'(b) = F^n(\perp)(b)$$

for all $b{:}B$. Now

$$\begin{aligned}
\alpha(b) &= [\![(Y\ F)]\!](b) \\
&= \bigsqcup\{F^n\}_{n\in\mathcal{N}} \bigsqcup\{b\}_{n\in\mathcal{N}} \\
&= \bigsqcup\{F^n(b)\}_{n\in\mathcal{N}} \quad \text{by continuity of application} \\
&= \bigsqcup\{F^n \circ \alpha'(b)\}_{n\in\mathcal{N}} \quad \text{by 8} \\
&= (\bigsqcup\{F^n\}_{n\in\mathcal{N}} \circ \bigsqcup\{\alpha'\}_{n\in\mathcal{N}})(b) \quad \text{again by continuity of application} \\
&= [\![Y\ F]\!] \circ \alpha'(b) \\
&= \alpha'(b)
\end{aligned}$$

which, by the definition $\equiv_c$ implies $\alpha \equiv_c \alpha'$. Thus $Y\ F$ is the terminal object in the category of cones over the diagram $\triangle^{op}$ and consequently the limit. □

Theorem 4.2.5 (The Existence of Algebras) *Let $\mathcal{M}$ be a* continuous λ *algebra, Ψ a representable endofunctor over $\mathcal{C}(\mathcal{M})$ and $\triangle^{op} = \langle \Psi^n \bigcirc_{\mathcal{C}}, \Psi^n \bigcirc_{\mathcal{C}} \rangle$ be an ω^{op} chain in $\mathcal{C}(\mathcal{M})$. Then $(Y\ \Psi_O, id_{Y\ \Psi})$ is an algebra for Ψ.*

Proof We show this by applying the basic lemma (2.3.5) and constructing a co-algebra for Ψ which is also an algebra for Ψ.

That $Y\,\Psi$ is an object of $\mathcal{C}(\mathcal{M})$ follows immediately from 4.1.4 and by 4.2.4 this is a colimit for $\triangle^{op}$. Applying the basic lemma we then have that $(Y\,\Psi, Id_{(Y\,\Psi)})$ is a terminal Ψ co-algebra, but since the structure map is simply the identity function for $Y\,\Psi$ then $(Y\,\Psi, Id_{Y\,\Psi})$ is also an algebra for Ψ. □

Now let Σ be an S sorted signature where $S = \{s_1, \ldots, s_n\}$ and Ψ_i be the functor corresponding to sort i analogous to the functor X_Σ in definition 2.3.6. If each sort in $\mathcal{C}(\mathcal{M})$ is given by the fixed point of a Ψ_i then we require solutions to the following set of domain equations in $\mathcal{C}(\mathcal{M})$:

$$\begin{aligned} \Gamma_1 &= \Psi_1 \langle \Gamma_1, \ldots, \Gamma_n \rangle \\ &\vdots \\ \Gamma_n &= \Psi_n \langle \Gamma_1, \ldots, \Gamma_n \rangle \end{aligned} \tag{9}$$

Each Ψ_i is a functor consisting purely of compositions of the basic objects of $\mathcal{C}(\mathcal{M})$ and the sum and product functors. The following theorem giving the solution to such sets of equations is well known and can be found in, for example, [MW85].

Theorem 4.2.6 *Let $f_1, \ldots, f_n$ be n functions defined by mutual recursion as follows:*

$$\begin{aligned} f_1 &= \Phi_1(f_1, \ldots, f_n) \\ &\vdots \\ f_n &= \Phi_n(f_1, \ldots, f_n) \end{aligned} \tag{10}$$

If

$$F = \lambda X.\langle \Phi_1(\pi_1(X), \ldots, \pi_n(X)), \ldots, \Phi_n(\pi_1(X), \ldots, \pi_n(X))\rangle$$

then $Y(F)$ is a solution for the set of equations in (10) where each $f_i = \pi_i(Y(F))$.

Consider the set of equations in (9) above. If each Ψ_i is representable then this set of equations can be solved by putting

$$\Psi = \lambda X.\langle \Phi_1(\pi_1(X), \ldots, \pi_n(X)), \ldots, \Phi_n(\pi_1(X), \ldots, \pi_n(X))\rangle$$

and then using theorem 4.2.6 to get the following set of solutions for the Γ_i as retracts in $\mathcal{C}(\mathcal{M})$:

$$\begin{array}{rcl} \Gamma_1 & = & \pi_1(Y\,\Psi) \\ & \vdots & \\ \Gamma_n & = & \pi_n(Y\,\Psi) \end{array}$$

By lemma 4.2.2 Ψ is representable if each Ψ_i is representable and by the fact that the identities for products are defined pointwise in $\mathcal{C}(\mathcal{M})$ and by theorem 4.2.5 it follows that $\langle Y\,\Psi, id_{Y\,\Psi}\rangle$ is a Ψ algebra.

Notation 4.2.7 *Let Γ^ω denote the product $\Gamma_{s_1} \times \ldots \times \Gamma_{s_n}$.*

Now let $\sigma \in \Sigma_{\omega s}$ and $\langle Y\,\Psi, id_{Y\,\Psi}\rangle$ be a Ψ algebra. Suppose that

$$\Psi \overset{def}{=} \lambda X. \coprod \{\Gamma^\omega(X) | \, \sigma \in \Sigma_{\omega s} \, and \, \omega \in S^*\}$$

(as in definition 2.3.6). Then

$$Y\,\Psi \equiv \coprod \{\Gamma^\omega(Y\,\Psi) | \, \sigma \in \Sigma_{\omega s} \, and \, \omega \in S^*\}$$

such that the operation corresponding to σ is the injection $inj_\omega : \Gamma^\omega \to \Gamma_s$.

Fact 4.2.8 *Each Γ_s is the disjoint sum of the domains of each of the operations of sort s.*

In the sequel the Γ_i will be referred to as the *carriers* of the Ψ algebra and the operation in $(Y\,\Psi)$ corresponding to $\sigma \in \Sigma_{\omega s}$ will be denoted by $\overline{\sigma}$.

Example 4.2.9 *Consider the following specification of lists of natural numbers:*

Spec *Nat*

  Sorts *Nat*

  Operations $0 \quad : \rightarrow Nat$

  $\qquad\qquad S \quad : Nat \rightarrow Nat$

  End

Spec *List_of_Nat*

  Nat +

  Sorts *List*

  Operations $Nil \quad : \rightarrow List$

  $\qquad\qquad Cons : Nat \times List \rightarrow List$

  End

The functor corresponding to *List_Of_Nat* is

$$List \overset{def}{=} [\![\lambda L.\langle 1 + \pi(L), 1 + \pi'(L)\rangle]\!]$$

which, by theorem 4.2.5 has the solution $(Y\,List, id_{Y\,List})$ with

$$\begin{aligned} \Gamma_{Nat} &\overset{def}{=} [\![\pi(Y\,List)]\!] \\ \Gamma_{List} &\overset{def}{=} [\![\pi'(Y\,List)]\!] \end{aligned}$$

Here *0* is represented by the element of $\mathcal{M}$ $[\![\lambda x.\langle \overline{1}, I\rangle]\!]$ (or $inj_1 : 1_{\mathcal{C}} \rightarrow (Y\,List)$), while S is represented by $[\![\lambda x.\langle \overline{2}, \Gamma_{Nat}(x)\rangle]\!]$. Similarly *Nil* is represented by $[\![\lambda x.\langle \overline{3}, I\rangle]\!]$ and *Cons* by $[\![\lambda x.\langle \overline{4}, \Gamma_{Nat} \times \Gamma_{List}(x)\rangle]\!]$. We use the Church numerals $\overline{n}$ to denote different injections into the sum.

4.3 Algebras Satisfying Equations in $\mathcal{C}(\mathcal{M})$

The algebras given in the last section were privileged in the sense that there is an explicit way of constructing these from the signature Σ. In general algebras satisfying sets of Σ equations do not correspond to the algebras that can be constructed using theorem 4.2.5 but are actually retracts of anarchic algebras given by that theorem. If we now consider equational presentations such that the equations can be treated as a set of rewrite rules then this retract may be recursively defined (see chapter 2.1).

Consider a nonvoid signature Σ and the initial fixed point T_Σ of the functor X_Σ, that is, the pair (T_Σ, γ) where γ is an isomorphism. If we now assume that the word problem for T_Σ is solvable then there exists a (recursive) map $\tau : T_\Sigma \rightarrow T_\Sigma$ taking each element of T_Σ to its canonical representative, in which case the following composition is also defined:

$$\begin{tikzcd} X_\Sigma(T_\Sigma) \arrow[r, "\gamma"] & T_\Sigma \arrow[r, "\tau"] & T_\Sigma \end{tikzcd}$$

Consequently $(T_\Sigma, \tau \circ \gamma)$ is also a X_Σ algebra.

Definition 4.3.1 *Let* $A \overset{def}{=} A_1 + \ldots + A_n$ *and* B *be objects of* $\mathcal{C}(\mathcal{M})$ *. Then if*

$$\phi : (A_1 + \ldots + A_n) \times B \rightarrow A_1 \times B + \ldots + A_n \times B$$

or

$$\phi : B \times (A_1 + \ldots + A_n) \rightarrow B \times A_1 + \ldots + B \times A_n$$

we say that $A \times B$ is Expanded along ϕ.

Consider the set of solutions $\{\Gamma_1, \ldots, \Gamma_n\}$ to the equation (9) of section 4.3 given by theorem 4.2.6. Each Γ_i is of the form

$$A_1 + \ldots + A_n$$

where each A_i is a product of some set of carriers

$$\{\Gamma_1, \ldots, \Gamma_k\} \subseteq \{\Gamma_1, \ldots, \Gamma_n\}$$

Suppose $\Gamma_i = A_1 + \ldots + A_p$ and now consider a single expansion of the finite product of the $\{\Gamma_1, \ldots, \Gamma_k\}$ along some map ϕ as follows:

$$\begin{tikzcd} \Gamma_1 \times \ldots \times (\Gamma_{i-1} \times ((A_1 + \ldots + A_p) \times (\Gamma_{i+1} \ldots \times \Gamma_k))) \arrow[d, "\phi"] \\ \Gamma_1 \times \ldots \times (\Gamma_{i-1} \times (A_1 \times \Gamma_{i+1} \ldots \times \Gamma_k + \ldots A_p \times \Gamma_{i+1} \ldots \times \Gamma_k) \end{tikzcd}$$

Then by a finite number of additional expansions we can achieve the following expansion of the original product:

$$\Gamma_1 \times \ldots \times \Gamma_{i-1} \times A_1 \times \Gamma_{i+1} \ldots \times \Gamma_k + \ldots + \Gamma_1 \times \ldots \times \Gamma_{i-1} \times A_p \times \Gamma_{i+1} \ldots \times \Gamma_k$$

We say that the product $\Gamma_1 \times \ldots \times \Gamma_k$ has been *expanded* to *Standard Form*[2]. Every product of the Γ_i which results from the equation (9) of chapter 4.3 can be expanded to a standard form although this is not unique nor is it isomorphic to the original product in $\mathcal{C}(\mathcal{M})$.

Lemma 4.3.2 (Derived Operations) *Let Ψ be an endofunctor and $\langle \Gamma_1, \ldots, \Gamma_n \rangle$ the carriers for a Ψ algebra in $\mathcal{C}(\mathcal{M})$. Then:*

1. *If $\xi_i : \Gamma^{\omega_i} \to \Gamma_{s_i}$ is an injection, $A = \coprod A_j$ is an expansion of $\Gamma_{s_1} \times \ldots \times \Gamma_{s_n}$ along φ, $\omega = s_1 \ldots s_n$ and $\varphi : \Gamma^{\omega} \to A$ then there exists an injection $inj_{\Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n}}$ of $\Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n}$ into $A_1 + \ldots + A_j$ making the diagram*

$$\begin{tikzcd}
 & A_1 + \ldots + A_j \\
\Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n} \arrow[ur, "inj_{\Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n}}"] \arrow[r, "\xi_1 \times \ldots \times \xi_n"'] & \Gamma_{s_1} \times \ldots \Gamma_{s_n} \arrow[u, "\varphi"']
\end{tikzcd}$$

   commute in $\mathcal{C}(\mathcal{M})$.

2. *Let $\overline{\sigma} : \Gamma^{\omega} \to \Gamma^{s}$ be an operation of the Ψ algebra $(Y\ \Psi)$ and $\overline{\xi_1} \times \ldots \times \overline{\xi_n}$ be a product of injections such that each ξ_i is a map $\Gamma^{\omega_i} \to \Gamma_{s_i}$. Then there exists an injection $\overline{\sigma}$ such that $\overline{\sigma} \circ \overline{\xi_1} \times \ldots \times \overline{\xi_n}$ is an injection of $\Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n}$ into Γ_s.*

Proof Proof sketch:

1. We wish to show that there is an injection $inj_{\Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n}}$ of the domain of the map $\xi_1 \times \ldots \times \xi_n$ into an expansion of $\Gamma_{s_1} \times \ldots \times \Gamma_{s_n}$. We do this by induction on n, the number of multiplicands in the

[2]Considering the resemblance to disjunctive normal forms in boolean algebra perhaps "disjunctive Standard Form" would have been a more apt name

product:

Basis case follows directly from fact 4.2.8.

Induction Case Assume that the lemma is true for an n-fold product, that is, the following diagram commutes:

$$\begin{tikzcd}
& A_1 + \ldots + A_j \\
(\Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n}) \arrow[ur, "inj_{\omega_1 \times \ldots \times \omega_n}"] \arrow[r, "\xi_1 \times \ldots \times \xi_n"'] & \Gamma_{s_1} \times \ldots \Gamma_{s_n} \arrow[u, "\varphi"']
\end{tikzcd}$$

Now for an *n+1* fold product the functoriality of $\_ \times \_$ ensures that the diagram:

$$\begin{tikzcd}
& \Gamma_{s_{n+1}} \times (A_1 + \ldots + A_j) \\
\Gamma_{\omega_{n+1}} \times (\Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n}) \arrow[ur, "\xi_{n+1} \times (inj_{\omega_1 \times \ldots \times \omega_n})"] \arrow[r, "\xi_{n+1} \times (\xi_1 \times \ldots \times \xi_n)"'] & \Gamma^{s_{n+1}} \times (\Gamma_{s_1} \times \ldots \Gamma_{s_n}) \arrow[u, "\Gamma_{s_{n+1}} \times \varphi"']
\end{tikzcd}$$

also commutes. By fact 4.2.8 and the assumption that $\xi_{n+1} : \Gamma^{\omega_{n+1}} \to \Gamma_{s_{n+1}}$ is an injection $\Gamma_{s_{n+1}}$ must have an expansion of the form

$$B_1 + \ldots + \Gamma^{\omega_n} + \ldots + B_m$$

Now for some A_l in $A_1 + \ldots + A_j$, $1 \leq l \leq j$, such that

$$A_l = \Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n}$$

there is an expansion γ such that $\Gamma_{s_{n+1}} \times (A_1 + \ldots + A_j)$ is of the form

$$B_1' + \ldots + \Gamma^{\omega_{n+1}} \times \Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n} + \ldots B_r'$$

and so there is an injection of $\Gamma^{\omega_{n+1}} \times \Gamma^1 \times \ldots \times \Gamma^{\omega_n}$ into an expansion of $\Gamma_{s_{n+1}} \times \Gamma_{s_1} \times \ldots \times \Gamma_{s_n}$.

2. To show that $\overline{\sigma} \circ \xi_1 \times \ldots \times \xi_n$ is an injection of $\Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n}$ into a standard form of Γ_s consider the diagrams

$$\begin{tikzcd}
 & A_1 + \ldots + \Gamma_1 \times \ldots \times \Gamma_n + \ldots + A_p \\
\Gamma^{\Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n}} \arrow[ur, "inj_{\Gamma^{\omega_1} \times \ldots \times \Gamma^{\omega_n}}"] \arrow[r, "\xi_1 \times \ldots \times \xi_n"'] & \Gamma^{\omega} \arrow[u, "\varphi"']
\end{tikzcd}$$

$$\begin{tikzcd}
 & B_1 + \ldots + \Gamma_1 \times \ldots \times \Gamma_n + \ldots + B_j \\
\Gamma^{\omega} \arrow[ur, "inj_{\overline{\sigma}}"] \arrow[r, "\overline{\sigma}"'] & \Gamma_s \arrow[u, "\varphi'"']
\end{tikzcd}$$

where $\omega = s_1 \ldots s_n$ and φ and φ' are expansions.

Now notice that the map

$$B_1 + \ldots + \varphi + \ldots + B_j$$

is a proper expansion of $B_1 + \ldots + \Gamma^{\omega} + \ldots + B_j$.

□

Now consider a presentation (Σ, E) of an equational theory. If the set of equations E can be turned into a confluent and strongly normalising set of rewrite rules then by theorem 2.1.15 there exists a recursive functions $\tau : T_\Sigma \to T_\Sigma$. We now outline an algorithm to construct τ from a set of Church-Rosser strongly normalising rewrite rules.

Definition 4.3.3 *let $\phi : X \to T_\Sigma(Y)$ be a substitution. We say that a term t* matches *a term t' via ϕ if*

$$\overline{\phi}(t') = t$$

where $\overline{\phi}$ is the extension of ϕ to the set of all terms with variables in X.

Definition 4.3.4 *Let $\mathcal{B}$ be a strongly normalising and Church-Rosser set of rewrite rules. Define the function τ on ground terms by the rules:*

1. *If there is a rule $\sigma(\xi_1, \ldots, \xi_n) \to \varphi(\xi'_1, \ldots, \xi'_k)$ in $\mathcal{B}$ and substitution ϕ such that $\sigma(t_1, \ldots, t_n)$ matches $\sigma(\xi_1, \ldots, \xi_n)$ via ϕ then put*

$$\varphi(t'_1, \ldots, t'_n) = \overline{\phi}(\varphi(\xi'_1, \ldots, \xi'_k))$$

   and define:

$$\tau(\sigma(t_1, \ldots, t_n)) = \tau(\phi(\tau(t'_1), \ldots, \tau(t'_k)))$$

2. *Otherwise put*

$$\tau(\sigma(t_1, \ldots, t_n)) = \varphi(\tau(t'_1), \ldots, \tau(t'_k))$$

First we show that the image of τ is indeed a transversal for $\equiv_E$ in T_Σ.

Lemma 4.3.5 *If σ is a ground term in normal form then*

$$\tau(\sigma) = \sigma$$

Proof By structural induction and the definition of τ. □

We use the notation $t \xrightarrow{1} t'$ if $\langle t, t \rangle \in \mathcal{B}$ and say that t reduces to t' in *one step*. We also write $t \xrightarrow{k} t'$ if there is a sequence of one step reductions such that

$$t \xrightarrow{1} t_1 \xrightarrow{1} \ldots t_{k-1} \xrightarrow{1} t'$$

We write $t\downarrow$ as a shorthand for $t \to^*_{\mathcal{B}} t\downarrow$ and $t\downarrow$ is in normal form. The *bound* of t is the maximum length of the reduction sequence of t to normal form. Denote the bound of a term t by *bound(t)*.

Lemma 4.3.6 *For any $\sigma(t_1, \ldots, t_n)$ and t_i, $i \in \{1, \ldots, n\}$*

1. *$bound(\sigma(t_1, \ldots, t_n)) \geq t_i$*

2. *$bound(\sigma(t_1, \ldots, t_n)) \geq bound(\sigma(t_1\downarrow, \ldots, t_n\downarrow))$*

Proof Follows from the definition of bound. □

Lemma 4.3.7 *Let $\sigma(t_1, \ldots, t_n)$ be a ground term. Then*

$$\tau(\sigma(t_1, \ldots, t_n)) = \sigma(t_1, \ldots, t_n)\downarrow$$

Proof Let $\mathcal{B}$ be a set of strongly normalising, Church-Rosser set of rewrite rules. Then all sequences of 1 step reductions of $\sigma(t_1, \ldots, t_n)$ to normal form are finite.

Now, by induction on the bound of $\sigma(t_1, \ldots, t_n)$ we show that

$$\sigma \rightarrow^*_{\mathcal{B}} \sigma\!\downarrow \quad \text{implies} \quad \tau(\sigma) = \sigma\!\downarrow$$

Basis case : $bound(\sigma(t_1, \ldots, t_n)) = 1$

If $\sigma(t_1, \ldots, t_n) \xrightarrow{1} \varphi(t'_1, \ldots, t'_m)$ then $\varphi(t'_1, \ldots, t'_m)$ is in normal form and each t'_i is in normal form. Now there are two possible cases: either there is a rule

$$\sigma(\xi_1, \ldots, \xi_n) \rightarrow \varphi(\xi'_1, \ldots, \xi'_m)$$

in $\mathcal{B}$ such that $\sigma(t_1, \ldots, t_n)$ matches $\sigma(\xi_1, \ldots, \xi_n)$ via ϕ or there is not. We give the case in which there is such a rule since the other case can be shown in similar fashion.

By definition 4.3.4

$$\tau(\sigma(t_1, \ldots, t_n) = \tau(\varphi(\tau(t_1), \ldots, \tau(t_m)))$$

and by lemma 4.3.5 each $\tau(t_i) = t_i$, therefore:

$$\begin{aligned} \tau(\varphi(\tau(t_1), \ldots, \tau(t_m))) &= \tau(\varphi(t_1, \ldots, t_m)) \\ &= \varphi(t_1, \ldots, t_m) \end{aligned}$$

where $\varphi(t_1, \ldots, t_m)$ is the normal form of $\sigma(t_1, \ldots, t_n)$.

Induction case : Assume $\forall\, k \leq n$

$$\begin{aligned} &\varphi(t_1 \ldots, t_m) \xrightarrow{k} \varphi(t_1 \ldots, t_m)\!\downarrow \\ &\text{implies} \\ &\tau(\varphi(t'_1 \ldots, t'_m)) = \varphi(t'_1 \ldots, t'_m)\!\downarrow \end{aligned}$$

and the bound of $\varphi(t_1 \ldots, t_m)$ is n. Again we show only the case in which there is a rule

$$\sigma(\xi_1, \ldots, \xi_n) \rightarrow_{\mathcal{B}} \phi(\xi'_1, \ldots, \xi'_k)$$

in $\mathcal{B}$ in which case:

$$\tau(\sigma(t_1,\ldots,t_n)) = \tau(\varphi(\tau(t'_1),\ldots,\tau(t'_m)))$$

For each t'_i we have that $bound(\varphi(t'_1\ldots,t'_m)) \geq bound(t'_i)$ by lemma 4.3.6 and so by the induction assumption $\tau(t'_i) = t'_i\downarrow$. Then

$$\begin{aligned}\tau(\sigma(t_1,\ldots,t_n)) &= \tau(\varphi(\tau(t'_1),\ldots,\tau(t'_m))) \\ &= \tau(\varphi(t'_1\downarrow,\ldots,t'_n\downarrow))\end{aligned}$$

By lemma 4.3.6 and the induction assumption again we have that $bound(\varphi(t'_1,\ldots,t'_m)) \geq bound(\varphi(t'_1\downarrow,\ldots,t'_m\downarrow))$ and so

$$\tau(\varphi(t'_1\downarrow,\ldots,t'_n\downarrow)) = \varphi(t'_1,\ldots,t'_m)\downarrow$$

That $\sigma(t_1,\ldots,t_n)\downarrow = \varphi(t'_1,\ldots,t'_n)$ follows from the Church-Rosser property of $\mathcal{B}$. □

Another property of τ that we will have occasion to use is the following:

Lemma 4.3.8 *τ is idempotent.*

Proof by lemmas 4.3.5 and 4.3.7. □

Theorem 4.3.9 *The image of τ is a transversal for $\equiv_E$ in T_Σ.*

Proof By lemmas 4.3.5 and 4.3.7 the image of T_Σ under τ is the set of $\rightarrow_\mathcal{B}$ normal forms and so also a subset of T_Σ. Since $\rightarrow_\mathcal{B}$ is both Church-Rosser and strongly normalising each equivalence class $[t] \in T_\Sigma / \equiv_E$ contains a unique such normal form. □

Thus we have defined a recursive function τ which is idempotent by lemma 4.3.8 and which is a retract of T_Σ. We now give the definition of τ in $\mathcal{C}(\mathcal{M})$.

Theorem 4.3.10 *For any signature Σ and set of strongly normalising, Church-Rosser rewrite rules $\mathcal{B}$, let $\overline{\sigma}$,$\overline{\sigma}\downarrow : 1 \rightarrow \Gamma_s$ be the operations in $Y(\Psi)$ corresponding to the the ground terms σ and $\sigma\downarrow$ respectively. Then there exists an arrow $\tau : \Gamma_s \rightarrow \Gamma_s$ such that $\tau \circ \overline{\sigma} \equiv_{\Gamma_s} \overline{\sigma}\downarrow$.*

$$\begin{tikzcd}
 & & A \\
\Gamma_i^{\omega_1} \times \dots \Gamma_i^{\omega_n} \arrow[urr, "inj_{\Gamma^{\omega_1} \times \dots \times \Gamma^{\omega_n}}"] \arrow[rr, "\overline{\xi}_1^i \times \dots \times \overline{\xi^i}_n"] \arrow[drr, "\overline{\varphi}_i"'] & & \Gamma_{s_1} \times \dots \times \Gamma_{s_n} \arrow[u, "\gamma"'] \arrow[d, "\overline{\sigma}"] \\
 & & \Gamma_s A
\end{tikzcd}$$

Figure 5: The Definition of an Operation in $\mathcal{C}(\mathcal{M})$

To give a proof of theorem 4.3.10 we first present an algorithm to construct τ and give two preliminary lemmas.

Consider the set of all operations of sort $s \in S$, $\{\sigma_1, \dots, \sigma_k\}$, and the set of all rewrite rules in B of sort s:

$$\begin{array}{rcl}
\sigma_1(\xi_1^1, \dots, \xi_n^1) & \to & \varphi_1 \\
& \vdots & \\
\sigma_k(\xi_1^k, \dots, \xi_n^k) & \to & \varphi_k
\end{array}$$

1. Define A to be the standard form of Γ_s such that (1) there is an injection

$$inj_\sigma \circ inj_{\xi_1^i} \times \dots \times inj_{\xi^i_{k_i}}\ \Gamma^{\omega_1} \times \dots \Gamma^{\omega_n} \to A$$

   for each of the rules in (11) (this is guaranteed by lemma 4.3.2) and so if Γ^{ω_i} is the domain of $\sigma_i(\xi_1^i, \dots, \xi_{k_j}^i)$ in $\mathcal{C}(\mathcal{M})$ then Γ^{ω_i} is a summand of A.

2. Now let the number of summands of A be p. We want τ to make each of the equations in (11) valid, that is, for each $i \in \{1, \dots, m\}$ the diagram in figure 6 must commute. Now we have p summands and k equations and we wish to construct a set $\{\phi_1, \dots, \phi_p\}$ so that each ϕ_i corresponds to the action of τ on the i-th summand of A, that is, $\phi_i : \Gamma^{\omega_k} \to \Gamma_s$. Let $\{\tau_s\}_{s \in S}$ be a family of new variables, one for each sort $s \in S$. Define the ϕ_k as follows:

$$\begin{tikzcd}
\Gamma^{\omega_1} \times \ldots \Gamma^{\omega_n} \arrow[r, "\overline{\xi^i}_1 \times \ldots \times \overline{\xi^i}_n"] \arrow[d, "\overline{\varphi}_i"'] & \Gamma_{s_1} \times \ldots \times \Gamma_{s_n} \arrow[d, "\overline{\sigma_i}"] \\
\Gamma_s & \Gamma_s \arrow[l, "\tau"']
\end{tikzcd}$$

Figure 6: Definition of τ

- If there is a rule $\sigma_k(\xi_1, \ldots, \xi_k) \to \varphi(\xi'_1, \ldots, \xi'_l)$ in $\mathcal{B}$ such that

$$\overline{\sigma_k}(\overline{\xi_1}, \ldots, \overline{\xi_k}) : \Gamma^{\omega_k} \to A$$

is an arrow in $\mathcal{C}(\mathcal{M})$ then put

$$\phi_k = \tau_s \circ \overline{\varphi}_k \circ \langle \tau_{s_1} \circ \overline{\xi'_1}, \ldots, \tau_{s_l} \circ \overline{\xi'_l} \rangle$$

- otherwise put $\phi_k = \overline{\sigma_k} \circ \langle \tau_{s_1} \circ \overline{\xi_1}, \ldots, \tau_{s_l} \circ \overline{\xi_l} \rangle$

Let

$$\Phi_{s_i} = [\phi^i_1(\tau_{s_1}, \ldots, \tau_{s_n}), \ldots, \phi^i_p(\tau_{s_1}, \ldots, \tau_{s_n})] \circ \gamma_{s_i}$$

Now we have the n equations

$$\begin{aligned}
\tau_{s_1} &= \Phi_{s_1}(\tau_{s_1}, \ldots, \tau_{s_n}) \\
&\vdots \\
\tau_{s_n} &= \Phi_{s_n}(\tau_{s_1}, \ldots, \tau_{s_n})
\end{aligned}$$

and by theorem 4.2.6 has the solution

$$\tau = Y(\lambda T. \langle \Phi_{s_1}(\pi_1(T), \ldots, \pi_n(T)), \ldots, \Phi_{s_n}(\pi_1(T), \ldots, \pi_n(T)) \rangle)$$

where γ_{s_i} is the expansion of $\Gamma_{s_1} \times \ldots \times \Gamma_{s_n}$ making the diagram in figure 5 commute. Each τ_{s_i} can now be projected from the solution to the equations. We can now give a semantics for each operation $\sigma \in \Sigma_{\omega s}$ in $\mathcal{C}(\mathcal{M})$ defining $\overline{\sigma}$ as the arrow in $\mathcal{C}(\mathcal{M})$ making the following diagram commute.

$$\begin{tikzcd}
 & & A \arrow[dd, "\tau_s"] \\
 & \Gamma_s \arrow[ur, "\gamma_s"] & \\
\Gamma^{\omega} \arrow[ur, "\gamma_\Gamma"] \arrow[rr, "\overline{\sigma}"'] & & \Gamma_s
\end{tikzcd}$$

where each of the γ's is any appropriate expansion.

Lemma 4.3.11 *Let $\mathcal{B}$ be a set of rewrite rules. If σ is a $\mathcal{B}$ normal form and $\overline{\sigma} : 1_{\mathcal{C}} \to \Gamma_s$ is the operation in $Y\Psi$ corresponding to σ then $\tau_s \circ \overline{\sigma} \equiv_{\Gamma_s} \overline{\sigma}$.*

Proof In the case when $\overline{\sigma}$ is a ground term the diagram in figure 5 reduces to the following:

$$\begin{tikzcd}
1_{\mathcal{C}} \arrow[r, "\overline{\sigma}"] \arrow[dr, "inj_{1_{\mathcal{C}}}"'] & \Gamma_s \arrow[d, "\gamma"] \\
 & A
\end{tikzcd}$$

Since γ makes the diagram above commute then by the definition of τ we have

$$\begin{aligned}
\tau_s \circ \overline{\sigma} &= [\phi_1, \ldots, \phi_p] \circ \gamma(\overline{\sigma}) \\
&= \phi_j
\end{aligned}$$

for some ϕ_j. Since $\overline{\sigma}$ corresponds to a normal form then there is no rule in $\mathcal{B}$ of the form $\sigma \to \sigma'$ and so by the definition of τ, $\phi_j = \overline{\sigma}$. □

Lemma 4.3.12 *If σ is a ground term of sort s and $\sigma \xrightarrow{1} \sigma\!\downarrow$ then $\tau \circ \overline{\sigma} \equiv_{\Gamma_s} \overline{\sigma\!\downarrow}$*

Proof If $\sigma \xrightarrow{1} \sigma\!\downarrow$ then either:

1. there is a rule $\sigma(\xi_1, \ldots, \xi_n) \to \psi$ in B such that σ matches $\sigma(\xi_1, \ldots, \xi_n)$ via a substitution ϕ and $\phi(\psi) = \sigma\downarrow$, or
2. σ is of the form $\sigma(t_1, \ldots, t_i, \ldots, t_n)$, $\sigma\downarrow \equiv \sigma(t_1, \ldots, t_i\downarrow, \ldots, t_n)$ and $t_i \xrightarrow{1} t_i\downarrow$

In the first case there is a rule $\sigma(\xi_1, \ldots, \xi_n) \to \varphi(\xi'_1, \ldots, \xi'_m)$ such that $\sigma(t_1, \ldots, t_n) \to \varphi(t'_1, \ldots, t'_m)$ and so:

$$\begin{aligned}
\tau \circ \overline{\sigma} \circ \langle \overline{t}_1, \ldots, \overline{t}_n \rangle &= \tau_s \circ \overline{\varphi} \circ \langle \tau_{s_1} \circ \overline{t'}_1, \ldots, \tau_{s_m} \circ \overline{t'}_m \rangle \\
&\quad \text{by defnition of } \tau_s \\
&= \tau_s \circ \varphi \circ \langle \overline{t'}_1, \ldots, \overline{t'}_m \rangle \\
&= \varphi \circ \langle \overline{t'}_1, \ldots, \overline{t'}_m \rangle
\end{aligned}$$

by lemma 4.3.11.

In the second case we must have had a subterm u of t_i which reduces to $u\downarrow$ and so u matches some σ via a substitution ϕ and $\sigma \to \varphi$ is a rule in $\mathcal{B}$. This case can then be shown in a similar manner to the first. We assume without loss of generality that t_i and u are identical. Then by lemma 4.3.11 $\sigma \circ \langle \overline{t}_1, \ldots, t_i, \ldots, \overline{t}_n \rangle$ is the representation of $\sigma\downarrow$. □

Now we come to the proof of theorem 4.3.10.

Proof We do this by natural induction on the *bound* of a term σ as before.
The case of *bound*$(\sigma) = 1$ follows form lemma 4.3.12. The induction case is handled in the same way as theorem 4.3.7.
Assume $\forall\, k \leq n$

$$t_i \xrightarrow{k} t_i\downarrow \quad \text{implies} \quad \tau_s \circ \overline{t}_i \equiv_{\Gamma_s} \overline{t}_i\downarrow$$

Then,

$$\tau_s \circ \overline{\sigma} \circ \langle \overline{t}_1, \ldots, \overline{t}_n \rangle = \phi_k \circ \langle \overline{q}_1, \ldots, \overline{q}_j \rangle$$

for some terms $\langle q_1, \ldots, q_j \rangle$. Put

$$\begin{aligned}
\phi_k \circ \langle \overline{q}_1, \ldots, \overline{q}_j \rangle &= \tau_s \circ \overline{\varphi} \circ \langle \tau_{s_1} \circ \overline{t'}_1, \ldots, \tau_{s_n} \circ \overline{t'}_n \rangle \\
&= \tau_s \circ \varphi \circ \langle \overline{t'\downarrow}_1, \ldots, \overline{t'\downarrow}_n \rangle \\
&\quad \text{by induction assumption} \\
&= \overline{\varphi} \circ \langle \overline{t'\downarrow}_1, \ldots, \overline{t'\downarrow}_n \rangle
\end{aligned}$$

by applying the induction assumption twice. □

Corollary 4.3.13 $\forall\, m : \Gamma_s.\ (\tau \circ \tau) m \;=\; \tau\, m$, *that is* $\tau \circ \tau \equiv_\tau \tau$ *in* $\mathcal{C}(\mathcal{M})$

Algebras constructed by the algorithm in the proof of theorem 4.3.10 satisfy the equations of the original presentation (Σ, E) (which is shown by the following theorem) but what is interesting is that algebras can be constructed in $\mathcal{C}(\mathcal{M})$ simply by requiring that Σ gives a term algebra which is non-empty and that the equations define rules for computation.

Theorem 4.3.14 (τ Algebras) *Let* (Σ, E) *be a presentation. If* Σ *is* non-void *and there is exists a strongly normalising Church-Rosser set of rewrite rules* B *obtainable from* E *such that* B *solves the word problem for* $T_{\Sigma E}$ *and* X_Σ *the functor corresponding to* Σ *then there exists an arrow* τ *in* $\mathcal{C}(\mathcal{M})$ *such that* $(Y\ \Psi, \tau \circ id_{Y\ \Psi})$ *is a* X_Σ *algebra satisfying the equations in* E.

Proof By theorem 4.2.5 $(Y\ \Psi, id_{Y\ \Psi})$ is a Ψ algebra in $\mathcal{C}(\mathcal{M})$ and by theorem 4.3.10 if $t \to_{\mathcal{B}}^{*} t\!\downarrow$ then in $\mathcal{C}(\mathcal{M})$ $\tau \circ \overline{t} \equiv_{\Gamma_s} \overline{t\!\downarrow}$. If $\mathcal{B}$ solves the word problem for $T_{\Sigma E}$ then by theorem 2.1.7 if $E \vdash_E t = t'$ then there exist a t'' such that $t \to_{\mathcal{B}}^{*} t''$ and $t' \to_{\mathcal{B}}^{*} t''$. By theorem 4.3.10 this means that $\tau_s \circ \overline{t} \equiv_{\Gamma_s} \overline{t''}$ and $\tau_s \circ \overline{t'} \equiv_{\Gamma_s} \overline{t''}$ which by the transitivity of $\equiv_{\Gamma_s}$ implies $\tau_s \circ \overline{t} \equiv_{\Gamma_s} \tau_s \circ \overline{t'}$ in $\mathcal{C}(\mathcal{M})$. □

4.4 Examples

We introduce some notation for the examples which follow. Suppose we have a set of s sorted rewriting rules

$$\sigma_1 \;\to\; \psi_1$$

$$\vdots$$

$$\sigma_m \rightarrow \psi_m$$

then write

$$\begin{aligned} \tau(\sigma_1(\xi_1^1, \ldots, \xi_{n_1}^1)) &\rightarrow \tau(\psi_1(\tau(\varepsilon_1^1), \ldots, \tau(\varepsilon_{n_1'}^1))) \\ &\vdots \\ \tau(\sigma_m(\xi_1^m, \ldots, \xi_{n_m}^m)) &\rightarrow \tau(\psi_m(\tau(\varepsilon_1^m), \ldots, \tau(\varepsilon_{n_m'}^m))) \end{aligned}$$

to stand for the solution of the recursion equation as defined in theorem 4.3.10.

Example 4.4.1 *In the first example we look at a simple specification of truth values.*

Spec *Booleans*

Sorts *Bool*

Operations $T, F \;:\rightarrow Bool$

$\wedge, \vee \;: Bool \times Bool \rightarrow Bool$

$\neg \;: Bool \rightarrow Bool$

Axioms $\forall x \in Bool$

$T \wedge x = x$

$F \wedge x = F$

$T \vee x = T$

$F \vee x = x$

$\neg(T) = F$

$\neg(F) = T$

End

From definition 2.3.6 the functor we require is

$$Bool \equiv [\![\lambda B.1 + 1 + B \times B + B \times B + B]\!]$$

and is representable by lemma 4.2.2. We intend that the first product $B \times B$ be the domain of the $\wedge$ operation while the second is the domain of the $\vee$ operation.

The final summand B is intended to be the domain of the $\neg$ operation and so by theorem 4.2.5 $(Y\,Bool, id_{Y\,Bool})$ is a *Bool* algebra. Put $bool \overset{def}{=} (Y\,Bool)$. Then

$$id_{bool} : Bool(Y\,Bool) \rightarrow (Y\,Bool)$$

and so

$$id_{bool} = [T, F, \wedge, \vee, \neg] : 1 + 1 + bool \times bool + bool \times bool + bool \rightarrow bool$$

Then the operations in the (anarchic) *bool* algebra in $\mathcal{C}(\mathcal{M})$ are defined as follows:

$$\begin{aligned} T &\overset{def}{=} inj_1 \\ F &\overset{def}{=} inj_1 \\ \wedge &\overset{def}{=} inj_{bool \times bool} \\ \vee &\overset{def}{=} inj_{bool \times bool} \\ \neg &\overset{def}{=} inj_{bool} \end{aligned}$$

If we now we consider the equations of the *Booleans* specification as left to right rewriting rules then by the construction of τ in theorem 4.3.10 operations on *bool* can be defined as functions in $\mathcal{C}(\mathcal{M})$, for example,

$$\begin{aligned} \wedge &\overset{def}{\equiv} [\![[\pi', K(False) \circ \bigcirc_{1 \times Bool}, \ldots] \circ \gamma_{\wedge})]\!] \\ \vee &\overset{def}{\equiv} [\![[K(True) \circ \bigcirc_{1 \times Bool}, \pi', \ldots] \circ \gamma_{\vee})]\!] \end{aligned}$$

Since $\mathcal{C}(\mathcal{M})$ is a cartesian closed category then the exponential transposes of these operations can also be defined.

Alternatively we might wish to specify booleans using just T, F and the one operation *Nand* as follows:

Spec *Booleans*

Sorts *Bool*

Operations T, F $: \rightarrow Bool$

$Nand : Bool \times Bool \rightarrow Bool$

Axioms $\forall x \in Bool$

$$Nand(T,T) = F$$
$$Nand(T,F) = T$$
$$Nand(F,x) = T$$

End

In this case the functor is given by:

$$Bool' \equiv [\![\lambda B.1 + 1 + B \times B]\!]$$

and by theorem 4.2.5 we put $bool' = (Y\,Bool', id_{Y\,Bool'})$. The operation *Nand* is given by

$$Nand \overset{def}{=} [\![[K(False), K(True), K(True)] \circ \gamma_{Nand}]\!]$$

We can now give the usual definition of $\wedge$ in terms of *Nand.*

$$\wedge' \overset{def}{=} [\![Nand \circ \langle Nand, True \circ \bigcirc_{bool' \times bool'} \rangle]\!]$$

Despite being equal on the values *True* and *False* (if we choose the same representation in $\mathcal{C}(\mathcal{M})$ for both) $\wedge'$ and $\wedge$ are two different arrows in $\mathcal{C}(\mathcal{M})$. Indeed they are not even in the same *Hom* sets.

In our next example we consider free groups generated by an arbitrary type. Traditionally the free group generated by an arbitrary set is a universal algebra $(GRP(X), \eta)$ making the diagram

$$\begin{tikzcd}
X \arrow[r, "\eta"] \arrow[dr, "\rho"'] & GRP(X) \arrow[d, "\overline{\rho}"] \\
 & G
\end{tikzcd}$$

commute for any ρ and group G (see definition 2.1.2 and the following discussion there). In the diagram η is the inclusion of generators into $GRP(X)$. In the example which follows this is made explicit by introducing a coercion from the sort X into the sort GRP.

Example 4.4.2 *Groups generated by an arbitrary type* Γ_X.

Spec *Group*

Sorts *X Grp*

Operations $e \quad : \to Grp$

$inj_X : X \to Grp$

$\_+\_ \quad : Grp \times Grp \to Grp$

$- \quad : Grp \to Grp$

Axioms $\forall\ x, y, z \in X,$

$$e + inj(x) = inj(x)$$

$$-(inj(x)) + inj(x) = e$$

$$(inj(x)+inj(y))+inj(z) = inj(x)+(inj(y)+inj(z))$$

End

Consider an arbitrary type Γ_X. The functor *GRP* is defined as follows:

$$GRP \stackrel{def}{=} [\![\lambda G.1 + \Gamma_X + G \times G + G]\!]$$

The equations can be turned into a Church-Rosser, strongly normalising set of rewrite rules by applying the Knuth Bendix completion procedure. The resulting set of rules is given in figure 7 and in $\mathcal{C}(\mathcal{M})$ the corresponding arrow τ is given by the following set of recursive rules.

$$\begin{aligned}
\tau(e + inj(x)) &\to \tau(inj(x)) \\
\tau((inj(x) + inj(y)) + inj(z)) &\to \tau(\tau(inj(x)) + (\tau(inj(y)) + \tau(inj(z)))) \\
\tau(-inj(x) + inj(x)) &\to e \\
\tau(-inj(x) + (inj(x) + inj(z))) &\to \tau(inj(z)) \\
\tau(-e + inj(x)) &\to \tau(inj(x)) \\
\tau(-(-inj(x))) &\to \tau(inj(x)) \\
\tau(inj(x) + e) &\to \tau(inj(x)) \\
\tau(inj(x) + (-inj(x))) &\to e \\
\tau(inj(x) + (-inj(x) + inj(z))) &\to \tau(inj(z))
\end{aligned}$$

$$\begin{array}{rcl}
e + inj(x) & \rightarrow & inj(x) \\
(inj(x) + inj(y)) + inj(z) & \rightarrow & inj(x) + (inj(y) + inj(z)) \\
-inj(x) + inj(x) & \rightarrow & e \\
-inj(x) + (inj(x) + inj(z)) & \rightarrow & inj(z) \\
-e + inj(x) & \rightarrow & inj(x) \\
-(-inj(x)) & \rightarrow & inj(x) \\
inj(x) + e & \rightarrow & inj(x) \\
inj(x) + (-inj(x)) & \rightarrow & e \\
inj(x) + (-inj(x) + inj(z)) & \rightarrow & inj(z)
\end{array}$$

Figure 7: Rules to Decide the Word Problem for Free Groups

$$\begin{array}{rcl}
\tau(inj(x) + (inj(y) + inj(z))) & \rightarrow & \tau(inj(x)) + (\tau(inj(y)) + \tau(inj(z))) \\
\tau(e) & \rightarrow & e
\end{array}$$

In the case of normal forms like e we do not need to reapply τ because lemmas 4.3.7 and 4.3.5 already state that τ has no effect on operations representing values (normal forms).

Remark 4.4.3 *Some remarks need to be made.*

1. *The first is that there is essentially only one operation being defined in the example of groups, that is, the + operation. Although the axioms guarantee the existence of an inverse for any element x there are no rules specifically stating how to calculate such an inverse. Here inverses for elements of type Γ_X are represented by prefixing them with a $-$ (essentially an injection)*[3].

[3] *See for further reference [Kur65] or [MB67] for a definition of groups in terms of semigroups and [Coh81] for a presentation of groups without reference to the operation of inverse or identity*

2. *The second remark concerns the group operation $+$. Here $+$ simply stands for the group operation and bears no relation to any other computable function, say $+$ on the natural numbers, which also satisfies the group equations.*

Our final example continues with lists of natural numbers first introduced in example 4.2.9. We now consider adding two partial functions *head* and *tail*.

Example 4.4.4 *Lists of Natural Numbers.*

$$
\begin{array}{l}
\textbf{\textit{Spec}}\ List\_Of\_Naturals \\
\quad Naturals\ + \\
\quad \textbf{\textit{Sorts}}\ List \\
\quad \textbf{\textit{Operations}}\ \begin{array}{ll} Nil & :\rightarrow List \\ Cons & : Nat \times List \rightarrow List \\ Head & : List \rightarrow Nat \\ Tail & : List \rightarrow List \end{array} \\
\quad \textbf{\textit{Axioms}}\ \forall\, n \in Nat,\ l \in List \\
\qquad\qquad Head(Cons(n,l)) = n \\
\qquad\qquad Tail(Cons(n,l)) = l \\
\quad \textbf{\textit{End}}
\end{array}
$$

To overcome the fact that we are dealing only with total functions some elements representing error values may be added in the manner of [Gog78].

$$
\begin{array}{l}
\textbf{Spec}\ List\_Of\_Nat \\
\quad Naturals\ + \\
\quad \textbf{Sorts}\ List \\
\quad \textbf{Operations}\ \begin{array}{ll} Nil & :\rightarrow List \\ Cons & : Nat \times List \rightarrow List \\ Head & : List \rightarrow Nat \\ Tail & : List \rightarrow List \end{array} \\
\quad \textbf{Errors}\qquad\ \begin{array}{ll} ListE & :\rightarrow List \\ NatE & :\rightarrow Nat \end{array}
\end{array}
$$

Axioms $\forall n \in Nat, l \in List$

$$Head(Cons(n,l)) = n$$
$$Tail(Cons(n,l)) = l$$
$$Head(Nil) = NatE$$
$$Tail(Nil) = ListE$$

Error Axioms

$$Head(ListE) = NatE$$
$$Tail(ListE) = ListE$$
$$Cons(n, ListE) = ListE$$
$$Cons(NatE, l) = ListE$$

End

where *Nat* is given in example 4.2.9. The functor corresponding to this signature is

$$[\![\lambda L.\langle 1 + 1 + \pi(L), 1 + 1 + L\rangle]\!]$$

If we treat the equations directly as left to right rewrite rules then τ is given by the following set of recursion equations:

$$\begin{aligned}
\tau(Head(NIL)) &= NatE \\
\tau(Head(Cons(n,l))) &= \tau(n) \\
\tau(Tail(NIL)) &= ListE \\
\tau(Tail(Cons(n,l))) &= \tau(l) \\
\tau(Head(ListE)) &= NatE \\
\tau(Tail(ListE)) &= ListE \\
\tau(Cons(n, ListE)) &= ListE \\
\tau(Cons(NatE, l)) &= ListE
\end{aligned}$$

4.5 Summary

There are two aspects to the problem of constructing a representation in $\mathcal{C}(\mathcal{M})$ of an arbitrary presentation which are to first choose a representation and then to define functions over that representation. If the set E of equations is Knuth-Bendix completable then we have such a system of equations and if the specification is nonvoid then the ground term rewriting system is also Church-Rosser and strongly normalising. The set of normal forms then provide a choice of representation. A simple version of this idea is summed up in the *Principle of Definition*[4] defined by Huet and Hullot [GJ82]:

Definition 4.5.1 *Let Σ be a signature, $\mathcal{C}$ a set of* constructors *and $\mathcal{D}$ a set of* defined *operations such that $\Sigma = \mathcal{C} \uplus \mathcal{D}$. Also let E be set of equations. Then the* Principle of Definition *is expressed as the conjunction of the following two properties:*

1. *For every ground term M there exists a ground constructor term N such that $M =_E N$*

2. *For every pair of ground constructor terms M and N, $M =_E N$*

We do not consider such explicit partitions of the signature but rely on the completion procedure to indicate the *constructors* within the signature.

There are similarities between τ and the *Eval* function in [Plo77]. Plotkin [Plo77] considers a language with higher order functions, in which programs are closed terms of ground type. If $\mathcal{L}$ is a language with higher order functions and P is a program of ground type σ then the definition of *Eval* is

$$Eval(P) = c \quad \text{iff} \quad P \rightarrow^*_{\mathcal{L}} c$$

where $\rightarrow^*_{\mathcal{L}}$ is the reflexive and transitive closure of a reduction relation associated with the language $\mathcal{L}$. In the case of the function τ defined in 4.3.4 the language $\mathcal{L}$ is given by the signature Σ of the data type, the reduction relation $\rightarrow^*_{\mathcal{B}}$ is obtained

[4]In principle it is the same as the idea of *Sufficient Completeness* given in [VG78]

from the equations of the data type and for every closed term t of ground type

$$\tau(t) = c \quad \text{iff} \quad t \rightarrow_{B}^{*} c$$

The two conditions of theorem 4.3.14, that E can be completed to a strongly normalising and Church-Rosser system of rewrite rules and Σ is non-void, are now re-assessed. The condition that the rules be Church-Rosser is essential since if they are not $\tau : T_{\Sigma} \rightarrow T_{\Sigma}$ will not be a function and so not representable in $\mathcal{C}(\mathcal{M})$. The second condition is that of strong normalisation which is not forced on us by the category $\mathcal{C}(\mathcal{M})$. Consider, for example, the following presentation:

Example 4.5.2 *The natural numbers are given as follows:*

Spec *RECURSE*

 Sorts S

 Operations $a \quad : \rightarrow S$

 $f \quad : S \rightarrow S$

 $\_ \cdot \_ \quad : S \times S \rightarrow S$

 Axioms $\forall m \in S$

 $f(m) = m + f(m)$

 End

The corresponding definition of τ gives

$$\begin{aligned} \tau(f(a)) &= \tau(a) \circ \tau(f(a)) \\ &= a \circ \tau(f(a)) \\ &\vdots \end{aligned}$$

which is representable in $\mathcal{C}(\mathcal{M})$ as a lazy operation. We conjecture that the premise to theorem 4.3.14 may be weakened to exclude strong normalisation.

Chapter 5

Extending Equational Theories To Higher Order Equational Theories

In the previous chapter we studied the conditions under which models of some equational specifications can be constructed in $\mathcal{C}(\mathcal{M})$ (theorem 4.3.14). Given an equational specification which satisfies the conditions one can construct a model in $\mathcal{C}(\mathcal{M})$ but the original specification is now no longer a *complete* axiomatisation of this model. By theorem 4.3.14 deduction of theorems from the axioms of the specification is *sound*, that is, if $E \vdash_E e$ then any model constructed by theorem 4.3.14 satisfies e, but more equations are valid within this model than can be deduced using first order equational reasoning. Consider the specification of booleans given in example 4.4.1. One consequence which can not be deduced using purely first order reasoning is that

$$[\![(\wedge \circ \langle T, x\rangle)^*]\!] \equiv (\pi')^*$$

which follows from the fact that $\mathcal{M}$ is a λ algebra, and so satisfies all the rules of the λ calculus, and the equation

$$T \wedge x = x$$

which is valid in $(Y\,Bool, \tau)$. Is there an axiomatisation of the theory of the model of booleans given in example 4.4.1?

In giving an answer to this question a number of factors must be taken into account. The first is that the original equational theory must be preserved since any model constructed by theorem 4.3.14 is a model of the equational theory presented by the specification. The second is that all equations provable in the λ calculus are provable in any λ algebra $\mathcal{M}$ and so these also must appear in the theory but there are other possibilities, for example equations between terms of ground type involving a mixture of first order and higher order subterms.

In this chapter we give a typed λ calculus freely generated by an equational presentation and study some of its properties. We show that for the kind of specifications that give rise to the models of chapter 4 the proposed λ calculus is conservative over the equational theory while in chapter 6 the soundness of this calculus is shown.

5.1 A Simply Typed λ Calculus with Equationally Defined Operations

The simply typed λ calculus proposed here is obtained by adding abstraction, application and pairing to the operations of the signature Σ. While our ultimate aim is to study the passage from first order theories to higher order theories and then to the models proposed in chapter 4, for the present we fix a presentation (Σ, E) and study some properties of the proposed calculus. Given a presentation (regardless of whether it satisfies the preconditions of theorem 4.3.14) the simply typed λ calculus, $\lambda^{\Sigma E}$, based upon (Σ, E) is given by a set of *types*, a set of *terms* and a set of equations and inference rules which are given below.

Definition 5.1.1 *The set of* Types, *T, is inductively defined by the following rules:*

1. *1_λ is a type (called the* unit *type) and every $s \in S$ is a type. These are called the* base *types.*

2. *If α and β are types then $\alpha \times \beta$ and $\alpha \rightarrow \beta$ are also types.*

Let $\{X\}_\alpha$ be a countable set of variables for each type $\alpha \in T$. Then the set of terms is inductively defined by the following rules:

1. *if $x \in X_\alpha$ then x is a term of type α;*

2. *if $\sigma \in \Sigma_{\epsilon,s}$ then σ is a constant of type s (*Algebraic constants*);*

3. *if $\sigma \in \Sigma_{\omega,s}$ and $\vec{t}$ a vector of terms where each t_i is of type α_i then $\sigma(\vec{t})$ is a term of type s (*Algebraic terms*);*

4. *if $x \in X_\alpha$ and e is a term of type β then $\lambda x.e$ is a term of type $\alpha \rightarrow \beta$ (*Abstraction*)[1] and if e_1 is a term of type $\alpha \rightarrow \beta$ and e_2 is a term of type β then $e_1\, e_2$ is a term of type β (*Application*);*

[1] *Sometimes we will write $\lambda x_\alpha . e_\beta$ to make the types explicit*

5. *if e_1 and e_2 are terms of types α and β respectively then $\langle e_1, e_2 \rangle$ is a term of type $\alpha \times \beta$ (*pairing*) and finally if e is a term of type $\alpha \times \beta$ then $\pi_{\alpha\beta}(e)$ and $\pi'_{\alpha\beta}(e)$ are terms of types α and β respectively.*

6. *$*$ is the only term of type 1_λ*

The set of terms of type α is denoted by $\Lambda_{\Sigma,\alpha}(X)$ while we use $\Lambda_\Sigma(X)$ to denote the set of all terms regardless of type. Points *1.*, *2.* and *3.* of definition 5.1.1 are the same as the definition of the free Σ algebra generated by a (countable) set of first order variables Y. We may assume up to renaming that these are included in the set of all variables, that is, $Y_s \subseteq X_s$ for all $s \in S$, then there is an injection $\eta : T_\Sigma(Y) \to \Lambda_\Sigma(X)$. If $t \in T_\Sigma(Y)$ then we often write just t for $\eta(t)$.
The rules for substitution $M[N/x]$ are the same as those in definition 3.1.2 augmented by the following rules:

$$\begin{aligned}
\sigma(t_1, \ldots, t_n)[\vec{e}/\vec{x}] &= \sigma(t_1[\vec{e}/\vec{x}], \ldots, t_n[\vec{e}/\vec{x}]) \\
\langle e_1, e_2 \rangle[\vec{e}/\vec{x}] &= \langle e_1[\vec{e}/x], e_2[\vec{e}/x] \rangle \\
\pi(e)[\vec{e}/\vec{x}] &= \pi(e[\vec{e}/\vec{x}]) \\
\pi'(e)[\vec{e}/\vec{x}] &= \pi'(e[\vec{e}/\vec{x}])
\end{aligned}$$

where σ is an operator symbol from Σ. The rule given above for substitution on algebraic terms simply states that substitution is a homomorphism when applied to algebraic terms. If $\vec{x}$ is a vector of variables such that each variable x_i is also a member of X then we may think of each $\_[\vec{N}/\vec{x}]$ as a function

$$\_[\vec{N}/\vec{x}] : \Lambda_\Sigma(X) \to \Lambda_\Sigma(X) \tag{11}$$

Axioms are written as $e_1 =_{CNV} e_2$ where we use $=_{CNV}$ to mean that e_1 is equal to e_2 by the conversion rules, or rules of deduction, given in figure 8. The rules of deduction are simply those of the simply typed λ calculus with surjective pairing augmented by the equations of the specification.

Definition 5.1.2 *Let $\mathcal{F}$ be a set of $\lambda^{\Sigma E}$ equations of the form $M =_{CNV} N$. Then $\mathcal{F} \vdash_{\lambda,E} e$ where e is an equation, if:*

Logical Rules

$$(unit)\ e =_{CNV} * \quad \text{For any term } e \text{ of type } 1_\lambda$$

$$\lambda x_\alpha . e_\beta =_{CNV} \lambda y_\alpha . e_\beta[y/x] \quad (\alpha)$$

$$(\lambda x.e)\, e_1 =_{CNV} e[e_2/x] \quad (\beta)$$

$$\pi_{\alpha\beta}(\langle x_\alpha, y_\beta\rangle) =_{CNV} x_\alpha \quad \pi'_{\alpha\beta}(\langle x_\alpha, y_\beta\rangle) =_{CNV} y_\beta \ (\delta)$$

$$\langle \pi_{\alpha\beta}(z_{\alpha\times\beta}) \pi'_{\alpha\beta}(z_{\alpha\times\beta})\rangle =_{CNV} z$$

Non Logical Axioms

If $(X, t, t') \in E$ then $t =_{CNV} t'$ is a non logical axiom.

Inference Rules

$$\frac{M =_{CNV} N}{\lambda x.M =_{CNV} \lambda x.N} \quad (\xi)$$

$$\frac{M =_{CNV} N}{X\, M =_{CNV} X\, N} \quad \frac{M =_{CNV} N}{M\, X =_{CNV} N\, X} \quad (\zeta)$$

and *Reflexivity*, *Symmetry* and *Transitivity*

Figure 8: Deduction rules.

1. *$e \in \mathcal{F}$*

2. *e is an instance of one of the axiom schemas (α), (β) or one of the axiom schemas (δ)*

3. *If one of the equations in $\mathcal{F}$ is a premise of one of the rules (ξ), (ζ) or reflexivity, symmetry or transitivity and e is the consequent of that rule.*

From now on we adopt the same convention as in [Bar84] that (α) congruent terms are identified, and that if the terms M_i, $1 \leq i \leq n$ appear in a particular context then their free variables are chosen so that they are different from any bound variables in that context. Also we will omit type subscripts where it is clear from context which types are involved.

A concept which is useful for relating higher order theories to first order theories, as in universal algebra, is the of congruence. A congruence for a universal algebra A is an equivalence relation which is compatible with the Σ structure of A. Likewise a congruence for a higher order theory will need to be compatible with the operations of abstraction, application and surjective pairing. We consequently have the following:

Definition 5.1.3 *The equivalence relation $\equiv_{\lambda E}$ is defined as the least relation satisfying:*

1. *if ϕ be any substitution on terms and $\overline{\phi}$ the extension of ϕ to all terms of $\Lambda_\Sigma(X)$ and*

$$\overline{E} = \{(\overline{\phi}(t), \overline{\phi}(t')) \mid (X, t, t') \in E\}$$

then

$$(t_1, t_2) \in \overline{E} \quad \textit{implies} \quad t_1 \equiv_{\lambda E} t_2$$

*(*closure under substitution*);*

2. *if $\sigma \in \Sigma_{\omega,s}$ and $t_1 \equiv_{\lambda E} t'_1, \ldots, t_n \equiv_{\lambda E} t'_n$ then*

$$\sigma(t_1, \ldots, t_n) \equiv_{\lambda E} \sigma(t'_1, \ldots, t'_n)$$

(Σ compatibility*);*

3. *for any* e_β *and* e'_α, $(\lambda x_\alpha . e_\beta)\, e'_\alpha \equiv_{\lambda E} e[e'/x]$

4. $\pi_{\alpha\beta}(\langle e, e' \rangle) \equiv_{\lambda E} e$
   $\pi'_{\alpha\beta}(\langle e, e' \rangle) \equiv_{\lambda E} e'$
   $\langle \pi_{\alpha\beta}(h), \pi'_{\alpha\beta}(h) \rangle \equiv_{\lambda E} h$;

5. *if* $e \equiv_{\lambda E} e'$ *then for any* f *of the appropriate type,* $(e\, f) \equiv_{\lambda E} (e'\, f)$ *and* $(f\, e) \equiv_{\lambda E} (f\, e')$;

6. *if* $e \equiv_{\lambda E} e'$ *then* $\lambda x.e \equiv_{\lambda E} \lambda x.e'$;

7. *for any* e, $e \equiv_{\lambda E} e$;

8. *if* $e \equiv_{\lambda E} e'$ *then* $e' \equiv_{\lambda E} e$;

9. *if* $e_1 \equiv_{\lambda E} e_2$ *and* $e_2 \equiv_{\lambda E} e_3$ *then* $e_1 \equiv_{\lambda E} e_3$;

We now have the following theorem.

Theorem 5.1.4 $E \vdash_{\lambda,E} M =_{CNV} N$ *iff* $M \equiv_{\lambda E} N$.

Proof ($\Rightarrow$) By induction on the length of the derivation $E \vdash_{\lambda,E} M =_{CNV} N$.

Basic Case If $E \vdash_{\lambda,E} M =_{CNV} N$ in one step then by the definition of $\vdash_{\lambda,E}$ either of the following must have occured:

- $M =_{CNV} N \in E$, in which case $(M, N) \in \overline{E} \subseteq \equiv_{\lambda E}$ by definition.
- $M =_{CNV} N$ is an instance of one of the axiom schemas (α) (β) or (δ) each instance of which belongs to $\equiv_{\lambda E}$ by definition 5.1.3.

Induction Step Assume that $E \vdash_{\lambda,E} M =_{CNV} N$ implies $M \equiv_{\lambda E} N$ in n steps. Then any application of a rule of inference to $M =_{CNV} N$ returns a consequent $M' =_{CNV} N'$ as follows:

- by (ζ), $E \vdash_{\lambda,E} X\, M =_{CNV} X\, N$ which by definition 5.1.3 is in $\equiv_{\lambda E}$; Likewise $E \vdash_{\lambda,E} M\, X =_{CNV} N\, X$ is in $\equiv_{\lambda E}$;
- by (ξ) $E \vdash_{\lambda,E} \lambda x.M = \lambda x.N$ which by definition 5.1.3 is in $\equiv_{\lambda E}$;

- compatibility, reflexivity, symmetry and transitivity inference rules follow by similar means.

If $M' =_{CNV} N'$ is an instance of an axiom schema or extra-logical axiom then this is independent of the induction assumption and so follows by the same argument as the basis case.

($\Leftarrow$)

Inductively define the sequence of sets U_i as follows:

$$\begin{aligned}
U_0 &= \overline{E} \cup \\
&\{((\lambda x.M)\,N, M[n/x]) \mid M, N \in \Lambda_\Sigma(X)\} \cup \\
&\{(\pi_{\alpha\beta}(\langle M, N\rangle), M) \mid M, N \in \Lambda_\Sigma(X)\} \cup \\
&\{(\pi'_{\alpha\beta}(\langle M, N\rangle), N) \mid M, N \in \Lambda_\Sigma(X)\} \cup \\
&\{(\langle \pi_{\alpha\beta}(M), \pi'_{\alpha\beta}(M)\rangle, M) \mid M \in \Lambda_{\Sigma,\alpha\times\beta}(X) | \forall \alpha,\ \beta \in T\} \\
&\{(M, M) \mid M \in \Lambda_\Sigma(X)\} \\
U_{n+1} &= \{(X\,M, X\,N) \mid (M, N) \in U_n\} \cup \\
&\{(M\,X, N\,X) \mid (M, N) \in U_n\} \cup \\
&\{(\lambda x.M, \lambda x.N) \mid (M, N) \in U_n\} \cup \\
&\{(M, N) \mid (N, M) \in U_n\} \cup \\
&\{(M, N) \mid (M, P), (P, N) \in U_n\}
\end{aligned}$$

Let $U = \bigcup U_i$. Now by induction on i we have that $U_i \subseteq \equiv_{\lambda E}$ and by definition 5.1.3 $\equiv_{\lambda E} \subseteq U_i$ for some i and so $\equiv_{\lambda E} = U$. By induction on i if $\langle M, N\rangle \in U_i$ then $E \vdash_{\lambda,E} M =_{CNV} N$ □

From the definitions given above $\equiv_E \subseteq \equiv_{\lambda E}$ (see 2.1.5) and so by theorem 2.1.7 and by theorem 5.1.4 we also have

$$E \vdash_E \forall X.\, t = t' \ \textit{implies}\ E \vdash_{\lambda E} t =_{CNV} t'$$

where the free variables of t and t' are all in X. Equations which were universally quantified in E are now to be considered as equal "functions" (of their free variables). If $(X, t, t') \in E$ then $E \vdash_{\lambda,E} t =_{CNV} t'$ and so by applying the (ξ) inference rule once for each of the variables in X the following equation

$$\lambda x_1 \ldots \lambda x_n.t =_{CNV} \lambda x_1 \ldots \lambda x_n.t'$$

is also provable, that is, we have an equality between functions. Now by applying (ζ) and (β) several times we can also show that:

$$\forall \vec{e}.\ t[\vec{e}/\vec{x}] =_{CNV} t'[\vec{e}/x]$$

where the $\vec{e}$ may contain higher order terms[2].

The following theorem is now a direct consequence of theorem 5.1.4 and definition 5.1.3.

Theorem 5.1.5 *Let (Σ, E) be a presentation. Then every derivation $E \vdash_E M = N$ in the equational calculus can be translated to a derivation $E \vdash_{\lambda E} M =_{CNV} N$ in the λ calculus generated by the terms and equations of (Σ, E).*

Proof The proof of this theorem is based upon a mapping between proof trees given by the inference rules of the equational calculus and those of the calculus $\lambda^{\Sigma E}$.

- Let $X = \{x_1, \ldots, x_n\}$ and $\varphi : \{x_1, \ldots, x_n\} \rightarrow T_\Sigma(X)$ and be a substitution. Then each instance of the substitution schema of definition 2.1.3

$$\frac{\forall x \in X.\, t = t'}{\forall x \in X \bigcup Y.\, \overline{\varphi}(t) = \overline{\varphi}(t')}$$

  is replaced by

$$\cfrac{\cfrac{\cfrac{t = t'}{\lambda x_1 \ldots \lambda x_n.t = \lambda x_1 \ldots \lambda x_n.t'}}{(\lambda x_1 \ldots \lambda x_n.t)\,\overline{\varphi}(x_1)\ldots\overline{\varphi}i(x_n) = (\lambda x_1 \ldots \lambda x_n.t')\,\overline{\varphi}(x_1)\ldots\overline{\varphi}(x_n)}}{t[\overline{\varphi}(x_1)/x_1]\ldots[\overline{\varphi}(x_n)/x_n] = t'[\overline{\varphi}(x_1)/x_1]\ldots[\overline{\varphi}(x_n)/x_n]}$$

  in $\lambda^{\Sigma E}$ where abstraction (ξ), application (ζ) and (β) rules are used to achieve the same substitution;

- Let φ_1 and φ_2 be substitutions such that $\varphi_i(x) = x$ for all $x \in X$ except one x_i such that $\varphi_1(x_i) = t_1$ and $\varphi_2(x_i) = t_2$. Then for

[2]The implication here is that in $\lambda^{\Sigma E}$ an implicit universal quantification has been introduced, that is, if $E \vdash_{\lambda,E} t =_{CNV} t'$ then we have $\forall x \in \alpha.\, t(x) =_{CNV} t'$ where α is the type of the free variables of t and t'

some $t \in T_\Sigma(X)$ each instance of the schema;

$$\frac{\forall x \in X.\, t_1 \;=\; t_2}{\forall x \in X \bigcup Y.\, \overline{\varphi_1}(t) \;=\; \overline{\varphi_1}(t)}$$

is mapped to:

$$\cfrac{\cfrac{t_1 \;=\; t_2}{(\lambda x_i.t)\, t_1 \;=\; (\lambda x_i.t)\, t_2}}{t[t_1/x_i] \;=\; t[t_2/x_i]}$$

- All instances of the *Transitivity*, *Reflexivity* and *Symmetry* rules remain unchanged.

It now remains to show that given an equation $t = t'$ such that $E \vdash_E t = t'$ then $E \vdash_{\lambda,E} t =_{CNV} t'$. By induction on the length of the derivation $E \vdash_E t = t'$ we have

Basis case if $E \vdash_E t = t'$ in 1 step then $t = t'$ must be an instance of an axiom in which case $E \vdash_{\lambda,E} t =_{CNV} t'$, or t is identical to t' and the equation follows by an application of reflexivity in which case by reflexivity in $\lambda^{\Sigma E}$, $E \vdash_{\lambda,E} t =_{CNV} t$;

Induction case Assume $E \vdash_E t = t'$ implies $E \vdash_{\lambda,E} t =_{CNV} t'$. Now by the definition of substitution in definition 3.1.2 if $\varphi : \{x_1, \ldots, x_n\} \to T_Y$ then

$$\overline{\varphi}(t) = t[\varphi(x_1)/x_1] \ldots [\varphi(x_n)/x_n] \tag{12}$$

- An application of the substitution inference rule gives

$$E, t = t' \vdash_E \overline{\varphi}(t) = \overline{\varphi}(t')$$

The induction assumption together with the translation for the substitution rule above gives

$$\begin{aligned} &E, t =_{CNV} t' \vdash_{\lambda,E} \\ t[\overline{\varphi}(x_1)/x_1] \ldots [\overline{\varphi}(x_n)/x_n] \;&=\; t'[\overline{\varphi}(x_1)/x_1] \ldots [\overline{\varphi}(x_n)/x_n] \end{aligned}$$

which by (12) above and the definition of derivation is equivalent to

$$E \vdash_{\lambda,E} \overline{\varphi}(t) =_{CNV} \overline{\varphi}(t')$$

- An application of the rule of replacement gives

  $$E, t_1 = t_2 \vdash_{\lambda,E} \overline{\varphi}_1(t) = \overline{\varphi}_2(t)$$

  for some t where φ_1 and φ_2 meet the premises of the replacement rule of definition 2.1.3. The induction assumption together with the translation again give

  $$E, t_1 =_{CNV} t_2 \vdash_{\lambda,E} t[t_1/x_i] =_{CNV} t[t_2/x_i]$$

  which by (12) above and the definition of derivation is equivalent to

  $$E \vdash_{\lambda,E} \overline{\varphi}_1(t) =_{CNV} \overline{\varphi}_2(t)$$

- relexivity, symmetry and transitivity follow trivially.

The proof of the converse is given in the next section. □

The mapping referred to in the proof of theorem 5.1.5 gives derived rules in $\lambda^{\Sigma E}$ for the substitution and replacement rules of the equational calculus (definition 2.1.3). These rules now give us the means within the calculus $\lambda^{\Sigma E}$ to carry out first order equational proofs, or put another way, to realize proof theoretically what was hinted at by the fact that $\equiv_E \subseteq \equiv_{\lambda E}$.

Finally to make some important comparisons later a term model for $\lambda^{\Sigma E}$ may be constructed from $\equiv_{\lambda E}$.

Definition 5.1.6 *Define the term model,*

$$\Lambda_{\Sigma E}(X) \overset{def}{=} \langle \Lambda_\Sigma(X)/\equiv_{\lambda E}, \langle -, - \rangle, \{\pi_{\alpha\beta}\}_{\alpha\beta\in T}, \{\pi'_{\alpha\beta}\}_{\alpha\beta\in T}, \cdot, \Sigma \rangle$$

for $\lambda^{\Sigma E}$ by:

(i) *For each type α, the elements of type α are given by $\{\, [M_\alpha]_\lambda \mid M_\alpha \in \Lambda_\Sigma(X)\}$, where $[M_\alpha]_\lambda$ is the $\equiv_{\lambda E}$ equivalence class of M_α.*

(ii) The operations are given by:

- *pairing operations are defined as follows*

$$\begin{aligned}\pi_{\alpha\beta}([M_{\alpha\times\beta}]_\lambda) &= [\pi_{\alpha\beta}(M_{\alpha\times\beta})]_\lambda \\ \pi'_{\alpha\beta}([M_{\alpha\times\beta}]_\lambda) &= [\pi'_{\alpha\beta}(M_{\alpha\times\beta})]_\lambda \\ \langle [M]_\lambda, [N]_\lambda\rangle &= [\langle M, N\rangle]_\lambda\end{aligned}$$

- *application, denoted by $\cdot$, is defined by $[M_{\alpha\to\beta}]_\lambda \cdot [N_\alpha]_\lambda = [MN]_\lambda$*
- *algebraic operations are defined by $\sigma_\lambda([M_1]_\lambda, \ldots, [M_n]_\lambda) = [\sigma(M_1, \ldots, M_n)]_\lambda$ for each $\sigma \in \Sigma_{ws}$ and $[M_1]_\lambda, \ldots, [M_n]_\lambda \in \Lambda_{\Sigma,E}(X)$.*

Let $\rho : X \to \Lambda_{\Sigma,E}(X)$ be an assignment of values in $\Lambda_{\Sigma,E}()$ for the variables in X.

Definition 5.1.7 *The interpretation of terms in $\Lambda_{\Sigma,E}(X)$ defined by ρ is given by:*

1. $[\![x]\!]_\rho = \rho(x)$
2. $[\![\sigma(M_1, \ldots, M_n)]\!]_\rho = \sigma_\lambda([\![M_1]\!]_\rho, \ldots, [\![M_n]\!]_\rho)$
3. $[\![\langle M, N\rangle]\!]_\rho = \langle [\![M]\!]_\rho, [\![N]\!]_\rho\rangle$
   $[\![\pi(M_{\alpha\times\beta})]\!]_\rho = \pi([\![M_{\alpha\times\beta}]\!]_\rho)$
   $[\![\pi'(M_{\alpha\times\beta})]\!]_\rho = \pi'([\![M_{\alpha\times\beta}]\!]_\rho)$
4. $[\![MN]\!]_\rho = [\![M]\!]_\rho \cdot [\![N]\!]_\rho$
5. $[\![\lambda x.M]\!]_\rho = [\lambda x.M[P_1/x_1]\ldots[P_n/x_n]]_\lambda$ *if $\{x_1, \ldots, x_n\}$ are the free variables of M and $\rho(x_1) = P_1, \ldots, \rho(x_n) = P_n$*

Lemma 5.1.8 *If $M \equiv_{\lambda E} N$ and for each i such that $1 \leq i \leq n$, $P_i \equiv_{\lambda E} P'_i$ then*

$$M[P_1/x_1]\ldots[P_n/x_n] \equiv_{\lambda E} N[P'_1/x_1]\ldots[P'_n/x_n]$$

Proof By induction on the structure of M. □

$\Lambda_{\Sigma E}$ is indeed a model of $\lambda^{\Sigma E}$ as is shown by the following theorem:

Theorem 5.1.9 $E \vdash_{\lambda,E} M =_{CNV} N$ *iff* $\forall \rho : X \rightarrow \Lambda_{\Sigma,E}, [\![M]\!]_\rho = [\![N]\!]_\rho$.

Proof Let $\{x_1, \ldots, x_n\}$ be the free variables in M and N and let ρ be an arbitrary assignment such that $\rho(x_i) = [P_i]_\lambda$ Then if $E \vdash_{\lambda,E} M =_{CNV} N$ by theorem 5.1.4 $M \equiv_{\lambda E} N$. If $P_i' \in [P_i]_\lambda$ then by lemma 5.1.8

$$M[P_1/x_1], \ldots, [P_n/x_n] \equiv_{\lambda E} N[P_1'/x_1], \ldots, [P_n'/x_n]$$

and so $[\![M]\!]_\rho = [\![N]\!]_\rho$.

Conversely if $\forall \rho$, $[\![M]\!]_\rho = [\![N]\!]_\rho$ then for any $P_1, \ldots, P_n$ we have

$$\rho(x_1) = [P_i]_\lambda,\ M[P_1/x_n] \ldots [P_n/x_n] = N[P_1/x_1] \ldots [P_n/x_n]$$

In particular if $\rho(x_i) = x_i$ then $M \equiv_{\lambda E} N$ and by theorem 5.1.4 $E \vdash_{\lambda,E} M = N$.
□

It is worth noting that if we were to continue the development of term algebras along the lines of chapter 2.1 then we might consider the algebra $\Lambda_{\Sigma,E}(\emptyset)$. By definition 5.1.1 if X is empty then there are only algebraic terms (terms of type s for each $s \in S$) and pairs of algebraic terms (terms of type $s_1 \times s_2$ for any $s_1, s_2 \in S$).

5.2 Reduction Properties of $\lambda^{\Sigma E}$

We wish to analyse the relation $\equiv_{\lambda E}$ further in order to show that $\lambda^{\Sigma E}$ is a conservative extension of the equational calculus (definition 2.1.3). This we do below but first we study some reduction properties of $\lambda^{\Sigma E}$.

Definition 5.2.1 *Let $\mathcal{B}$ be a set of rewrite rules and define the reduction relation $\beta\delta\mathcal{B}$ by the following set of schema and rules:*

$$\begin{aligned} x &\rightarrow_{\beta\delta\mathcal{B}} * \\ (\lambda x.M)\, N &\rightarrow_{\beta\delta\mathcal{B}} M[N/x] \\ \pi(\langle M, N\rangle) &\rightarrow_{\beta\delta\mathcal{B}} M \end{aligned}$$

$$\begin{aligned} \pi'(\langle M, N\rangle) &\to_{\beta\delta\mathcal{B}} N \\ \langle \pi(P), \pi'(P)\rangle &\to_{\beta\delta\mathcal{B}} P \end{aligned}$$

as well as for each rule $t \to_{\mathcal{B}} t'$ from B a rule $t \to_{\beta\delta\mathcal{B}} t'$. As in definition 2.1.10 let

$$\mathcal{B}_{\beta\delta\mathcal{B}} \overset{def}{=} \{\langle M, N\rangle \mid t \to t' \text{ and } M = \phi t \text{ and } N = \phi(t')\}$$

for all substitutions ϕ of free variables of t and t'. Use $\to^{*}_{\beta\delta\mathcal{B}}$ to denote the transitive and reflexive closure of $\mathcal{B}_{\beta\delta\mathcal{B}}$.

A useful lemma is given in [Klo80] which states that (β) and (δ) reduction rules can be interchanged without affecting the result. The lemma below gives essentially the same result for $\mathcal{B}$ and $\beta\delta$ reductions but before presenting this lemma some notation is introduced. One step reductions involving just the (β) and (δ) rules will be denoted by $\to_{\beta\delta}$ while those involving just reductions from $\mathcal{B}$ by $\to_{\mathcal{B}}$. If $M \to^{*}_{\beta\delta\mathcal{B}} N$ by just using rules in $\mathcal{B}$ then we write $M \to^{*}_{\mathcal{B}} N$ and similarly we write $M \to^{*}_{\beta\delta} N$ if $M \to^{*}_{\beta\delta\mathcal{B}} N$ by using only (α), (β) and (δ) reductions.

Lemma 5.2.2 (Interchange Lemma) *If we have a series of one step reductions*

$$t_1 \to_{\mathcal{B}} t_2 \to_{\beta\delta} t_3$$

then there is a t' making the following reduction sequences confluent:

$$\begin{tikzcd} t_1 \arrow[r, "\mathcal{B}"] \arrow[d, "\beta\delta"'] & t_2 \arrow[d, "\beta\delta"] \\ t' \arrow[r, "\mathcal{B}"'] & t_3 \end{tikzcd}$$

Proof The proof is by induction on the structure of t_1. The only interesting case is if $t_1 = (\lambda x.\phi)\, a$.
Now either $a \to_{\mathcal{B}} a'$ or $\phi \to_{\mathcal{B}} \phi'$.

- If $a \rightarrow_{\mathcal{B}} a'$ then,

$$\begin{array}{rcll} (\lambda x.\phi)\, a & \rightarrow_{\mathcal{B}} & (\lambda x.\phi)\, a' & \textit{by compatibility} \\ & \rightarrow_{\beta\delta} & \phi[a'/x] & \textit{by } (\beta) \end{array}$$

in which case put $t' = \phi[a/x]$. Then

$$\begin{array}{rcll} (\lambda x.\phi)\, a & \rightarrow_{\beta\delta} & \phi[a/x] & \textit{by } (\beta) \\ & \rightarrow_{\mathcal{B}} & \phi[a'/x] & \textit{by induction on } \phi \end{array}$$

- If $\phi \rightarrow_{\mathcal{B}} \phi'$ then,

$$\begin{array}{rcll} (\lambda x.\phi)\, a & \rightarrow_{\mathcal{B}} & (\lambda x.\phi')\, a & \textit{by compatibility} \\ & \rightarrow_{\beta\delta} & \phi'[a/x] & \textit{by } (\beta) \end{array}$$

in which case put $t' = \phi[a/x]$. Then

$$\begin{array}{rcll} (\lambda x.\phi)\, a & \rightarrow_{\beta\delta} & \phi[a/x] & \textit{by } (\beta) \\ & \rightarrow_{\mathcal{B}} & \phi'[a/x] & \textit{by definition } \mathcal{B}_{\beta\delta\mathcal{B}} \end{array}$$

□

We now show that if a set of rewrite rules $\mathcal{B}$ obtained from a set of Σ equations E is Church-Rosser and strongly normalising then the set of all equations derivable from the axioms and rules of $\lambda^{\Sigma E}$ is a *conservative* extension of the set of all equations derivable by the axioms in E and the rules of definition 2.1.3. The proof of this proceeds in two stages. First, we show that if $\mathcal{B}$ is strongly normalising then so is the reduction relation $\rightarrow_{\beta\delta\mathcal{B}}$. It is well known that a simple induction on the structure of terms is insufficient to prove the strong normalisation property and so we adopt a simplified version of the *computability method* of Girard [Tai67, Gal].

Secondly we show that if $\mathcal{B}$ is Church-Rosser then $\rightarrow_{\beta\delta\mathcal{B}}$ is Church-Rosser. It is then simple to show that for any two algebraic terms t and t' if $t \rightarrow^{*}_{\beta\delta\mathcal{B}} t'$ then $t \rightarrow_{\mathcal{B}} t'$.

Say that a term $M \in \Lambda_{\Sigma}(X)$ is *strongly normalising* if M strongly normalises and let $SN_{\alpha} \subseteq \Lambda_{\Sigma}(X)$ be the set of terms which are strongly normalising. The computability method now proceeds as follows:

1. define a subset C_{α} of SN_{α} which can be thought of as the set of all *reducible* terms [Gal];

2. show that $\Lambda_{\Sigma\alpha}(X) \subseteq C_{\alpha}$.

Start with the following definition.

Definition 5.2.3 *The family of sets $\{C_{\alpha}\}_{\alpha \in T}$, where T is the set of types defined in 5.1.1, is defined inductively as follows:*

1. $C_{\alpha} = SN_{\alpha} = \{t \mid t \in \Lambda_{\Sigma\alpha}(X)\}$ if α is 1 or any $s \in S$;

2. $C_{\alpha\times\beta} = C_{\alpha} \times C_{\beta} \bigcup \{M \mid M \rightarrow^{*}_{\beta\delta\mathcal{B}} x\, N_1 \ldots N_n\}$ where $x \in X_{\tau_1 \rightarrow \ldots \rightarrow \tau_n \rightarrow \alpha\times\beta}$ and $N_i \in C_{\tau_i}$, $1 \leq i \leq n$;

3. $C_{\alpha\rightarrow\beta} = \{M \in \Lambda_{\Sigma\alpha\rightarrow\beta} \mid \forall\, N \in C_{\alpha},\ (M\, N) \in C_{\beta}\}$.

Lemma 5.2.4 *For all types $\alpha \in T$, $C_{\alpha} \subseteq SN_{\alpha}$.*

Proof The proof is by induction on types. The base cases of $\alpha = 1$ and $\alpha = s \in S$ follow immediately from definition 5.2.3. In the induction case we have:

1. If $M \in C_{\alpha\times\beta}$ then either $M \in C_{\alpha} \times C_{\beta}$ or

$$M \in \{M \mid M \rightarrow^{*}_{\beta\delta\mathcal{B}} x\, N_1 \ldots N_n\}$$

In the first case $M \equiv \langle M_1, M_2 \rangle$ where, by the induction assumption, $M_1 \in SN_{\alpha}$ and $M_2 \in SN_{\beta}$. If $M_1 \neq \pi(M')$ and $M_2 \neq \pi'(M')$

then $\langle M_1, M_2\rangle \in SN_{\alpha\times\beta}$ otherwise if $M_1 = \pi(M')$ and $M_2 = \pi'(M')$ then

$$\langle \pi(M'), \pi'(M')\rangle \rightarrow_\delta M'$$

and by the induction assumption $M' \in SN_{\alpha\times\beta}$. In the second case if $M \rightarrow^*_{\beta\delta\mathcal{B}} x\, N_1 \ldots N_n$ then by the induction assumption each $N_i \in SN_{\tau_i}$ and so we can reduce each N_i to normal form. Then

$$M \rightarrow^*_{\beta\delta\mathcal{B}} x\, N_1 \ldots N_n \rightarrow^*_{\beta\delta\mathcal{B}} x\, N_1{\downarrow} \ldots N_n{\downarrow}$$

where $x\, N_1{\downarrow} \ldots N_n{\downarrow} \in SN_{\alpha\times\beta}$.

2. If $M \in C_{\alpha\rightarrow\beta}$ then for all $N \in C_\alpha$ $(M{\cdot}N) \in C_\beta$ by definition 5.2.3. By the induction assumption all terms $C_\beta \subseteq SN_\beta$ and so $(M\, N) \in SN_\beta$, but then $M \in SN_{\alpha\rightarrow\beta}$ also.

□

Thus we have fulfilled the first part of the computability proof. The second requires some further results.

Lemma 5.2.5 ([Gal]) *For all types $\alpha \in T$ C_α satisfies the following two* closure *properties.*

(S1) *For every variable $x \in X_{\tau_1\rightarrow\ldots\rightarrow\tau_n\rightarrow\beta}$ and $N_i \in SN_{\tau_i}$, $1 \le i \le n$, $x\, N_1 \ldots N_n \in C_\beta$.*

(S2) *For all $M \in SN_{\alpha\rightarrow\tau_1\rightarrow\ldots\rightarrow\tau_n\rightarrow\beta}$, $N \in SN_\alpha$ and $N_i \in SN_{\tau_i}$, $1 \le i \le n$, if*

$$M[N/x]\, N_1 \ldots N_n \in C_\beta$$

then

$$(\lambda x.M)\, N\, N_1 \ldots N_n \in C_\beta$$

In addition to these two closure properties one further property is required.

Lemma 5.2.6 *If the set of rewrite rules $\mathcal{B}$ is strongly normalising, $t_i \in C_{s_i}$, $1 \le i \le n$, and $\sigma \in \Sigma_{ws}$ then $\sigma(t_1, \ldots, t_n) \in C_s$.*

Proof First note that for all $s \in S$ $C_s = SN_s$ and therefore each t_i is in SN_{s_i}. Now

$$\sigma(t_1, \dots, t_n) \to^*_{\beta\delta\mathcal{B}} \sigma(t_1 \downarrow, \dots, t_n \downarrow)$$

since $t_i \in SN_{s_i}$ and so there are no β or δ redexes left in $\sigma(t_1 \downarrow, \dots, t_n \downarrow)$. By the assumption that $\mathcal{B}$ is strongly normalising we can reduce $\sigma(t_1 \downarrow, \dots, t_n \downarrow)$ to normal form using only $\mathcal{B}$ reductions and therefore $\sigma(t_1, \dots, t_n) \in SN_s$.
□

Lemma 5.2.7 *If $M \in C_\alpha$, $x \in X_\beta$ and $N \in C_\beta$ then $M[N/x] \in C_\alpha$.*

Proof By induction on the structure of M. The base cases where $M = *$, $M = \sigma$ where $\sigma \in \Sigma_{\epsilon s}$ and $M = x \in X_\alpha$ all follow in a straight-forward manner. The induction cases are shown as follows.

- If $M = \sigma(t_1, \dots, t_n)$ then

$$M[N/x] = \sigma(t_1[N/x], \dots, t_n[N/x])$$

By the induction assumption each $t_i[N/x] \in C_{s_i}$ if $t_i \in C_{s_i}$ and so by lemma 5.2.6 $\sigma(t_1, \dots, t_n)[N/x] \in C_s$.

- If $M = \langle M_1, M_2 \rangle$ then

$$M[N/x] = \langle M_1[N/x], M_2[N/x] \rangle$$

By the induction assumption $M_1 \in C_\alpha$ and $M_2 \in C_\beta$ and so

$$\langle M_1, M_2 \rangle \in C_\alpha \times C_\beta \subseteq C_{\alpha \times \beta}$$

If $M \in C_\alpha = \pi(M')$ then $\pi(M')[N/x] = \pi(M'[N/x])$ which is in C_α by the induction assumption and similarly for $M = \pi'(M')$.

- If $M = (e_1\, e_2)$ then $(e_1\, e_2)[N/x] = (e_1[N/x])\,(e_2[N/x])$. By the induction assumption $e_1[N/x] \in C_{\alpha \to \beta}$ and $e_2[N/x] \in C_\alpha$ and by the definition of $C_{\alpha \to \beta}$ $(e_1[N/x])\,(e_2[N/x]) \in C_\beta$.

- If $M = \lambda y.e$ then $(\lambda y.M)[N/x] = \lambda y.M[N/x]$. By the induction assumption $M[N/x] \in C_\beta$ and so by (S2) of lemma 5.2.5 for all $N \in C_\alpha$, $(\lambda y.M)\,N \in C_\beta$, but this is just the requirement that $\lambda y.M$ be in $C_{\alpha\to\beta}$.

□

Theorem 5.2.8 $\to_{\beta\delta B}$ *is strongly normalising.*

Proof By lemma 5.2.4 for all type $\alpha \in T$ $C_\alpha \subseteq SN_\alpha$. To prove the theorem we show that for all types $\alpha \in T$ $\Lambda_{\Sigma\alpha} \subseteq C_\alpha$, that is we carry out part two of the computability method. We do this by induction on the structure of terms. The basic cases of $*$, constants σ in Σ_{ws} and variables follow almost immediately from the definitions. The induction cases are as follows:

- Terms of the form $\sigma(t_1, \ldots, t_n)$ are in C_s if $\sigma \in \Sigma_{ws}$ by the induction hypothesis that each $t_i \in C_{s_i}$ and by lemma 5.2.6.
- For terms of the form $\langle M_1, M_2\rangle$ the induction hypothesis gives $M_1 \in C_\alpha$ and $M_2 \in C_\beta$, but then

$$\langle M_1, M_2\rangle \in C_\alpha \times C_\beta \subseteq C_{\alpha\times\beta}$$

  For $\pi(M)$ the induction hypothesis gives $M \in C_{\alpha\times\beta}$. If M is $\langle M_1, M_2\rangle$ then by induction $\pi(\langle M_1, M_2\rangle) \in C_\alpha$ and otherwise

$$\pi(M) \to^*_{\beta\delta B} \pi(x\, N_1 \ldots N_n) \in SN_\alpha$$

- For terms $(M_1\, M_2)$ the induction hypothesis gives $M_1 \in C_{\alpha\to\beta}$ and $M_2 \in C_\alpha$ which by definition of $C_{\alpha\to\beta}$ gives $(M_1\, M_2) \in C_\beta$.
- For $\lambda x.M$ we have by lemma 5.2.7 that for all $N \in C_\alpha$ $M[N/x] \in C_\beta$, but then by the definition of $C_{\alpha\to\beta}$ $\lambda x.M \in C_{\alpha\to\beta}$.

□

This gives us the following:

Theorem 5.2.9 *If $\mathcal{B}$ is a strongly normalising and Church-Rosser set of rewrite rules then $\rightarrow_{\beta\delta\mathcal{B}}$ is also strongly normalising and Church-Rosser.*

> **Proof** If $\mathcal{B}$ is strongly normalising then so is $\rightarrow_{\beta\delta\mathcal{B}}$ by theorem 5.2.8. Next we show that it is also Church-Rosser. Recall that (β) is a schema and so should be considered as a whole family of rules where each term in $\Lambda_\Sigma(X)$ is put in place of the meta-variables in (β). We also assume that the other symbols of $\lambda^{\Sigma E}$ such as π, π' and $\langle\_,\_\rangle$ are disjoint from those of $\mathcal{B}$.
>
> Let $\varphi \rightarrow \psi \in \rightarrow_E$, then there are two critical pairs to consider,
>
> $$\langle(\lambda x.\psi)\, a, \varphi[a/x]\rangle \; and \; \langle(\lambda x.t)\, \psi, t[\varphi/x]\rangle$$
>
> The first is the superposition of $\varphi \rightarrow \psi$ on $(\lambda x.\varphi)\, a$ and the second from superposing $\varphi \rightarrow \psi$ on $(\lambda x.t)\, \varphi$.
>
> 1. In the first case, after (β) reducing, we have $\psi[a/x]$ and $\varphi[a/x]$. By the compatibility of $\rightarrow_{\beta\delta\mathcal{B}}$ with the $\lambda^{\Sigma E}$ structure $\varphi[a/x] \rightarrow_{\beta\delta N} \psi[a/x]$.
> 2. In the second case after (β) reducing, we have $t[\psi/x]$ and $t[\varphi/x]$. By induction on t and compatibility of $\rightarrow_{\beta\delta\mathcal{B}}$ with the $\lambda^{\Sigma E}$ structure $t[\varphi/x] \rightarrow_{\beta\delta\mathcal{B}} t[\psi/x]$.
>
> Then by the Newman theorem 2.1.12 and Knuth-Bendix theorem 2.1.14 $\rightarrow_{\beta\delta\mathcal{B}}$ is Church Rosser. □

We now have the following theorem.

Theorem 5.2.10 *Let $\mathcal{B}$ be a Church-Rosser and strongly normalising set of rewrite rules. If t and t' are algebraic terms such that $t, t' \in T_\Sigma(X)$ and $E \vdash_{\lambda E} t = t'$ then $E \vdash_E t = t'$.*

> **Proof** By theorem 5.2.9, if $E \vdash_{\lambda E} t = t'$ the $\exists\, t''$ such that $t \rightarrow^*_{\beta\delta\mathcal{B}} t''$ and $t' \rightarrow^*_{\beta\delta\mathcal{B}} t''$. But since t and t' are terms from $T_\Sigma(X)$ then they contain no $(\beta\delta)$ redexes, and so $t \rightarrow^*_{\mathcal{B}} t''$ and $t' \rightarrow^*_{\mathcal{B}} t''$. Since $\mathcal{B}$ reduction rules do not introduce terms not in $T_\Sigma(X)$ then $t'' \in T_\Sigma(X)$ and since $\mathcal{B}$ is Church-Rosser then $E \vdash_E t = t'$. □

5.3 Examples

We now briefly re-examine the examples of chapter 3.4.

Example 5.3.1 *In the first example we re-examine the simple specification of truth values.*

Spec *Booleans*

  Sorts *Bool*

  Operations $T, F \; : \rightarrow Bool$

  $\wedge, \vee \; : Bool \times Bool \rightarrow Bool$

  $\neg \; : Bool \rightarrow Bool$

  Axioms $\forall x \in Bool$

  $T \wedge x = x$

  $F \wedge x = F$

  $T \vee x = T$

  $F \vee x = x$

  $\neg(T) = F$

  $\neg(F) = T$

  End

Some functions that are definable in $\lambda Booleans$ are given below.

- The exponential transpose of the $\wedge$ and $\vee$ operators are simply $\lambda x.\lambda y.x \wedge y$ and $\lambda x.\lambda y.x \vee y$. From the axioms we have $T \wedge x = x$ which by (ξ) (figure 8) gives $\lambda x.T \wedge x = \lambda x.x$
- Identity functions of all types can be defined, for example, $\lambda x_{Bool}.x$.
- Iterators of all types can be defined. The iterator for a type α is given by the term

$$Y_\alpha \overset{def}{=} \lambda n_{(\alpha\rightarrow\alpha)\rightarrow(\alpha\rightarrow\alpha)}.\lambda_{\alpha\rightarrow\alpha}.\lambda x_\alpha.n\, f\, x$$

where the first parameter n is the Church numeral

$$\lambda f_{\alpha\rightarrow\alpha}.\lambda x_\alpha.x$$

- Sequences of boolean values can also be formed. The empty sequence is given by $*_1$ and the addition of a boolean value onto the front of a sequence is defined by

$$CONS_i \overset{def}{=} \lambda x_{Bool}.\lambda y_{Bool^i}.\langle x, y\rangle$$

  where $Bool^i$ is the product

$$Bool \times (Bool \times (\ldots(Bool \times Bool)\ldots)$$

  with i repetitions of the type *Bool*. The head of the sequence is given by $\lambda x_{Bool^{i+1}}.\pi_{Bool,Bool^i}(x)$ and the tail by $\lambda x_{Bool^{i+1}}.\pi'_{Bool,Bool^i}(x)$.

The theory generated by the lists presentation is a bit more interesting. For the present we ignore the error values *Head(Nil)* and *Tail(Nil)*.

Example 5.3.2

Spec *List_Of_Naturals*

Naturals +

Sorts *Nat List*

Operations	*Nil*	:	$\rightarrow List$
	Cons	:	$Nat \times List \rightarrow List$
	Head	:	$List \rightarrow List$
	Tail	:	$List \rightarrow List$

Axioms $\forall n \in Nat,\ l \in List$

$$Head(Cons(n,l)) = n$$
$$Tail(Cons(n,l)) = l$$

End

Functions definable over the presentation of lists, in addition to those already definable over Naturals, now include some useful higher order functions.

- The higher order (iterative) function *map* can be defined in the theory $\lambda List$, for example, MAP_n for the type *List* may be defined for any positive integer

n as follows:

$$\lambda n_{(List\to list)\to(List\to List)}.\lambda f_{Nat\to Nat}.\lambda l_{List}.n\,(\lambda l_{List}.Cons(f(Head(l)),Tail(l)))\,l$$

The *Map* functions are not definable by general recursion but can be defined for each list of length n. Since general recursion can only be introduced by the equations in of the presentation all the lists of this theory are finite.

- As for the *map* function above *Fold* can be similarly defined. $FOLD_n$ for any positive integer n can be defined by:

$$FOLD_n \stackrel{def}{=} n\,(\lambda F.\lambda f.\lambda a.\lambda l.f(Head(l))(Tail(l)))(\lambda z_1.\lambda z_2.\lambda z_3.z_2)$$

The point here is that with iteration and the higher order types the theories of *Boolean* and *List* are closer to what may be considered functional programming. Indeed what we lack here is a means of general recursion. A second point is that when considering a functional programming language as the target of an implementation or refinement step, all the higher order operations are available in defining concrete operations.

Finally we present an example of equational deduction taken from [EM85].

Example 5.3.3 *The theory of rings is equationally presented as follows:*

Spec *Ring*

Sorts *R*

Operations *0,1* $: \to R$

$+,\cdot$ $\quad R\times R\to R$

$-$ $\quad R\to R$

Axioms $\forall\, x,y,z\in R$

$$x+0 = x$$
$$x+(y+z) = (x+y)+z$$
$$-x+x = 0$$
$$x+y = y+x$$
$$1\cdot x = x$$

$$\begin{aligned} x \cdot (y \cdot z) &= (x \cdot y) \cdot z \\ x \cdot (y + z) &= x \cdot y + x \cdot z \\ (x + y) \cdot z &= x \cdot z + y \cdot z \end{aligned}$$

End

In [EM85] an example proof of the theorem $0 + x = x$ is given by using the equational calculus. The proof of the same theorem in the theory λ*Rings* is given in figure 5.3.

5.4 Simple Extensions to $\lambda^{\Sigma E}$

The simple extension we have in mind is the addition of a family of fixed point combinators Y_α, one for each type α, with the usual reduction rules:

$$Y_\alpha(f_{\alpha \to \alpha}) = f(Y(f))$$

This simple extension is not a conservative extension of the first order equational calculus. Consider the following example adapted from [Klo80].

Example 5.4.1 *Let*

$$\begin{aligned} Y_1 &\in ((\sigma \to \sigma) \to (\sigma \to \sigma)) \to (\sigma \to \sigma) \\ Y_2 &\in (\sigma \to \sigma) \to \sigma \\ x &\in \sigma \to \sigma \end{aligned}$$

where $\sigma \overset{def}{=} \alpha \times \beta$ *for any two ground types* α *and* β*. Put*

$$\begin{aligned} C &\overset{def}{=} Y_1(\lambda c.\lambda m_\sigma.x\ \langle \pi(m), \pi'(x\ (c\ m))\rangle) \\ A &\overset{def}{=} Y_2(\lambda z_\sigma.C\ z) \end{aligned}$$

1.	$x+(y+z)$	$=$	$(x+y)+z$	*axiom*
2.	$\lambda x.\lambda y.\lambda z.x+(y+z)$	$=$	$\lambda x.\lambda y.\lambda z.(x+y)+z$	(ξ)
3.	$(\lambda x.\lambda y.\lambda z.x+(y+z))\,x\,(-x)\,x$	$=$	$(\lambda x.\lambda y.\lambda z.x+(y+z))\,x\,(-x)\,x$	(ζ)
4.	$x+(-x+x)$	$=$	$(x+(-x))+x$	(β)
5.	*lemma* $-x+x$	$=$	0	
6.	$x+y$	$=$	$y+x$	*axiom*
7.	$\lambda x.\lambda y.x+y$	$=$	$\lambda x.\lambda y.y+x$	(ξ)
8.	$(\lambda x.\lambda y.x+y)\,(-x)\,x$	$=$	$(\lambda x.\lambda y.y+x)\,(-x)\,x$	(ζ)
9.	$-x+x$	$=$	$x+(-x)$	(β)
10.	$x+(-x)$	$=$	0	*axiom*
11.	$-x+x$	$=$	0	*transitivity*
12.	$x+y$	$=$	$x+y$	*reflexivity*
13.	$\lambda x.\lambda y.x+y$	$=$	$\lambda x.\lambda y.x+y$	(ξ)
14.	$(\lambda x.\lambda y.x+y)\,x\,(-x+x)$	$=$	$(\lambda x.\lambda y.x+y)\,x\,0$	(ζ) *and* 11.
15.	$x+(-x+x)$	$=$	$x+0$	(β)
16.	$(x+(-x))+x$	$=$	$x+(-x+x)$	
			commutativity of 4.	
17.	$(x+(-x))+x$	$=$	$x+0$	*transitivity*
18.	$(x+(-x)$	$=$	0	*axiom*
19.	$x+y$	$=$	$x+y$	*reflexivity*
20.	$\lambda x.\lambda y.x+y$	$=$	$\lambda x.\lambda y.x+y$	(ξ)
21.	$(\lambda x.\lambda y.x+y)\,(x+(-x))\,x$	$=$	$(\lambda x.\lambda y.x+y)\,0\,x$	(ζ)
22.	$(x+(-x))+x$	$=$	$0+x$	(ζ)
23.	$0+x$	$=$	$(x+(-x))+x$	
			commutativity of 22.	
24.	$0+x$	$=$	$x+0$	
			transitivity of 23. *and* 17.	
25	$x+0$	$=$	x	*axiom*
26	$0+x$	$=$	x	
			transitivity of 25. *and* 24.	

Then we have the following sequence of reductions:

$$\begin{tikzcd}
A \arrow[r] & C\,A \arrow[r] \arrow[d] & C\,(x\,(C\,A)) \\
& x\,\langle \pi(A), \pi'(CA) \rangle \arrow[d] & \\
& x\,\langle \pi(CA), \pi'(CA) \rangle \arrow[d] & \\
& x\,(CA) &
\end{tikzcd}$$

and it can be shown [Klo80] that $C\,(x\,(C\,A)) \neq x\,(C\,A)$ and so we have lost the Church-Rosser property in the extension.

A second example of the loss of the Church-Rosser property is given in [BT88].

Example 5.4.2 *Consider the following system of rewrite rules over natural numbers:*

$$\begin{aligned} x - x &\rightarrow 0 \\ suc(x) - x &\rightarrow suc(0) \end{aligned}$$

Then we have the following sequence of reductions:

$$\begin{tikzcd}
Y(suc) - Y(suc) \arrow[r] \arrow[d] & 0 \\
suc(Y(suc)) - Y(suc) \arrow[d] & \\
suc(0) &
\end{tikzcd}$$

from which we can conclude that $0 = suc(0)$.

In both examples 5.4.1 and 5.4.2 the original sets of rewrite rules were not *regular* in the sense of [Klo80]. The adjunction of a family of fixed point operators to the set of equations (considered as rewrite rules) of example 5.3.2 does not suffer from this problem. An open question is whether every set of rewrite rules which is regular can be conservatively extended by a family of fixed point operators.

5.5 Comparison with Other Work

In [BT88] a calculus much the same as $\lambda^{\Sigma E}$ is presented. The major theorems that are proved in [BT88] are:

1. if $\mathcal{B}$ is a confluent reduction relation then so is $\rightarrow_{\beta\mathcal{B}}$;

2. the simply typed λ calculus extended by an algebraic theory is a conservative extension of the algebraic theory.

The first is essentially a version of theorem 5.2.10 which is weaker in its premises. In theorem 5.2.10 we have given a stronger version in which both the properties of strong normalisation and the Church-Rosser are preserved when algebraic theories are combined with the simply typed λ calculus. In particular theorem 5.2.8 shows that strong normalisation is preserved in the combination.

The second result is of more importance since it is the conservativity result of chapter 5.2. Our proof is essentially a proof by rewriting while in [BT88] the proof is by giving an algorithm to transform proofs in $\lambda^{\Sigma E}$ to proofs using $\vdash_E$. This is more general than our rewriting proof. We have chosen to work with deduction trees which has resulted in the two derived rules in $\lambda^{\Sigma E}$ which "implement" equational deduction while in [BT88] chains of equivalences are used. One of the complexities of constructing a proof of $E \vdash_E e$ from $E \vdash_{\lambda,E} e$ is that a single step using $\vdash_E$ is equivalent to a number of steps in $\vdash_{\lambda,E}$ (see the translation in the proof of 5.1.4). To then translate a derivation in $\vdash_{\lambda,E}$ back to $\vdash_E$ would require that sequences of rules be found in the derivation which correspond to steps in $\vdash_E$. For our purposes the restricted case when a set of equations gives rise to a set of strongly normalising and Church-Rosser rewrite rules is sufficient because the algebras constructed in chapter 4.1 are based on the assumption that the equations have this property.

Chapter 6

Soundness and Free Higher Order Theories

At this point we have both a model, the model constructed from a Church-Rosser strongly normalising set of rewrite rules, and the conservative extension of the equational calculus, $\lambda^{\Sigma E}$. What we now show is that $\lambda^{\Sigma E}$ deduction is sound, that is, if $E \vdash_{\lambda,E} M = N$ then the two terms M and N are identical in $\mathcal{C}(\mathcal{M})$. The basic approach stems from the models of λ calculus in arbitrary cartesian closed categories of [Koy82] (but see also [Bar84]) and the correspondence between λ calculus and cartesian closed categories (as in [Poi86, LS86]).

There is a second application of this correspondence in the second section of this chapter where it is shown that the algebraic theory presented by (Σ, E) can be extended to a cartesian closed category and that this category has a universal property, in essence, that every algebra in $\mathcal{C}(\mathcal{M})$ constructed according to theorem 4.3.14 can be uniquely extended to a (higher order) model of this cartesian closed category (or equivalently of the simply typed λ calculus generated by (Σ, E)).

6.1 Soundness of $\lambda^{\Sigma E}$ Deduction in $\mathcal{C}(\mathcal{M})$

We start by giving the interpretation of terms of the calculus $\lambda^{\Sigma E}$ in $\mathcal{C}(\mathcal{M})$ relative to a model A. A is necessary for the interpretation of the algebraic terms and indeed if A can be constructed by theorem 4.3.14 then we assume that it is the model we have in mind.

Let (Σ, E) be a presentation and let A be a model given by:

Carrier The family of *types* $\{\Gamma_s\}_{s \in S}$;

Operations For each $\sigma \in \Sigma$ an operation σ_A.

Now for each type $\tau \in T$ of $\lambda^{\Sigma E}$ we associate an object of $\mathcal{C}(\mathcal{M})$ as follows:

$$\begin{aligned} [\![1]\!] &= 1_{\mathcal{C}(\mathcal{M})} \\ \forall s \in S \quad [\![s]\!] &= \Gamma_s \\ [\![\alpha \times \beta]\!] &= [\![\alpha]\!] \times [\![\beta]\!] \\ [\![\alpha \rightarrow \beta]\!] &= [\![\alpha]\!] \rightarrow [\![\beta]\!] \end{aligned}$$

Let $\rho = x_1 \dots x_n$ be a sequence of variables where each $x_i \in X_{\alpha_i}$ and define

$$\Gamma^\rho \overset{def}{=} [\![\alpha_1]\!] \times \dots \times [\![\alpha_n]\!]$$

The canonical projection functions are defined by the projections in $\mathcal{C}(\mathcal{M})$ and we let

$$\pi^\rho_{x_i} : \Gamma^\rho \rightarrow [\![\alpha_i]\!]$$

denote the projection from Γ^ρ onto the i^{th} coordinate. Let $\rho = x_1 \dots x_n$ and $\mu = y_1 \dots y_m$ be two sequences of variables such that

$$\{y_1, \dots, y_m\} \subseteq \{x_1, \dots, x_n\}$$

Following [Bar84] the map

$$\Pi^\rho_\mu : \Gamma^\rho \rightarrow \Gamma^\mu$$

is defined as $\langle \pi^{\rho}_{y_1}, \ldots, \pi^{\rho}_{y_m} \rangle$ [Bar84, Koy82]. If $\langle f_1, \ldots, f_n \rangle$ is a tuple of functions then

$$\pi^{\rho}_{x_i} \circ \langle f_1, \ldots, f_n \rangle = f_i$$

while

$$\Pi^{\rho}_{\mu} \circ \langle f_1, \ldots, f_n \rangle = \langle f'_1, \ldots, f'_m \rangle$$

where each $f'_i = \pi^{\rho}_{x_i}$.

Now let M be a term of type α in $\lambda^{\Sigma E}$ and $\rho = x_1 \ldots x_n$ be a sequence of variables such that the free variables of M, *FV(M)*, are a subset of the $\{x_1, \ldots, x_n\}$. Before giving the interpretation of a term M relative to A we introduce some notation. If $\rho = x_1 \ldots x_n$ then ρ, x is the sequence of variables $x_1 \ldots x_n, x$.

Definition 6.1.1 *Let (Σ, E) be a presentation and A a model of (Σ, E). If M is a term in $\lambda^{\Sigma E}$, $\rho = x_1 \ldots x_n$ a sequence of variables such that $FV(M) \subseteq \{x_1, \ldots, x_n\}$ then the denotation of Min $\mathcal{C}(\mathcal{M})$ is a map $[\![M]\!] : \Gamma^{\rho} \to [\![\alpha]\!]$, defined by the following rules:*

$$\begin{array}{rrcl}
 & [\![*]\!]_{\rho} & = & [\![I]\!] \\
\forall x \in X & [\![x]\!]_{\rho} & = & \pi^{\rho}_{x} \\
 & [\![P\,Q]\!]_{\rho} & = & ev \circ \langle [\![P]\!]_{\rho}, [\![Q]\!]_{\rho} \rangle \\
 & [\![\lambda x.P]\!]_{\rho} & = & ([\![P]\!]_{\rho})^{*} \\
 & [\![\langle P, Q \rangle]\!]_{\rho} & = & \langle [\![P]\!]_{\rho}, [\![Q]\!]_{\rho} \rangle \\
 & [\![\pi(t)]\!]_{\rho} & = & \pi \circ [\![t]\!]_{\rho} \\
 & [\![\pi'(t)]\!]_{\rho} & = & \pi' \circ [\![t]\!]_{\rho} \\
 & [\![\sigma(t_1, \ldots, t_n)]\!]_{\rho} & = & \sigma_A \circ \langle [\![t_1]\!]_{\rho}, \ldots, [\![t_n]\!]_{\rho} \rangle
\end{array}$$

The interpretation given in 6.1.1 is based upon the cartesian closed structure of $\mathcal{C}(\mathcal{M})$ and follows the interpretation of the pure λ calculus in arbitrary cartesian closed categories[1]. The following three lemmas are simple adaptations of lemma 5.5.4 in [Bar84] (but see also [Koy82]). The proofs of the first two lemmas are by

[1]see [Koy82] and [Bar84], chapter 5.5

structural induction but we only give some cases because the omitted cases are very similar in character and can be found in [Bar84].

Lemma 6.1.2 *Let M be a term in $\lambda^{\Sigma E}$. If $FV(M) \subseteq \{\mu\} \subseteq \{\rho\}$ then $[\![M]\!]_\rho = [\![M]\!]_\mu \circ \Pi^\rho_\mu$.*

Proof By induction on the structure of M.

1.

$$\begin{aligned}
[\![\pi(t)]\!]_\rho &= \pi \circ [\![t]\!]_\rho \\
&= \pi \circ ([\![t]\!]_\mu \circ \Pi^\rho_\mu \quad \textit{by induction hypothesis} \\
&= (\pi \circ [\![t]\!]_\mu) \circ \Pi^\rho_\mu \\
&= [\![\pi(t)]\!]_\mu \circ \Pi^\rho_\mu
\end{aligned}$$

2.

$$\begin{aligned}
[\![\lambda x.P]\!]_\mu \circ \Pi^\rho_\mu &= \Phi([\![P]\!]_{\mu,x}) \circ \Pi^\rho_\mu \\
&= \Phi([\![P]\!]_{\mu,x} \circ \Pi^\rho_\mu \times id_{\Gamma^x}) \\
&= \Phi([\![P]\!]_{\mu,x} \circ \Pi^{\rho,x}_{\mu,x}) \\
&= \Phi([\![P]\!]_{\rho,x}) \quad \textit{by induction hypothesis} \\
&= \Phi(\lambda x.[\![P]\!]_\rho)
\end{aligned}$$

3.

$$\begin{aligned}
[\![\sigma(t_1,\ldots,t_n)]\!]_\mu \circ \Pi^\rho_\mu &= \sigma_A \circ \langle [\![t_1]\!]_\mu, \ldots, [\![t_n]\!]_\mu \rangle \circ \Pi^\rho_\mu \\
&= \sigma_A \circ \langle [\![t_1]\!]_\mu \circ \Pi^\rho_\mu, \ldots, [\![t_n]\!]_\mu \circ \Pi^\rho_\mu \rangle \\
&= \sigma_A \circ \langle [\![t_1]\!]_\rho, \ldots, [\![t_n]\!]_\rho \rangle \quad \textit{by induction hypothesis} \\
&= [\![\sigma(t_1,\ldots,t_n)]\!]_\rho
\end{aligned}$$

□

Lemma 6.1.3 *If $FV(M) \subseteq \{\rho\} = \{\vec{x}\}$ and $\vec{N}$ can be substituted for the $\vec{x}$ and $FV(M) \subseteq \{\mu\}$ then*

$$[\![M[\vec{N}/\vec{x}]]\!]_\mu = [\![M]\!]_\rho \circ \langle [\![N_1]\!]_\mu, \ldots, [\![N_n]\!]_\mu \rangle$$

Proof By induction on the structure of M.

1.

$$\begin{aligned}
[\![\pi(t)]\!]_\rho \circ \langle [\![N_1]\!]_\mu, \ldots, [\![N_n]\!]_\mu \rangle &= \pi \circ ([\![t]\!]_\rho \circ \langle [\![N_1]\!]_\mu, \ldots, [\![N_n]\!]_\mu \rangle) \\
&= \pi \circ [\![t[\vec{N}/\vec{x}]]\!]_\mu \quad \textit{by induction hypothesis} \\
&= [\![\pi(t)[\vec{n}/\vec{x}]]\!]_\mu
\end{aligned}$$

2.

$$\begin{aligned}
[\![(\lambda x.P)[\vec{N}/\vec{x}]]\!]_\mu &= [\![(\lambda x.P)[\vec{N}, x/\vec{x}, x]]\!]_\mu \\
&= \Phi([\![P[\vec{N}, x/\vec{x}, x]]\!]_{\mu,x}) \\
&= \Phi([\![P]\!]_{\rho,x} \circ \langle [\![N_1]\!]_{\mu,x}, \ldots, [\![N_n]\!], [\![x]\!]_{\mu,x} \rangle \\
&\qquad \textit{by induction hypothesis} \\
&= \Phi([\![P]\!]_{\rho,x} \circ \langle [\![N_1]\!]_{\mu,x} \circ \Pi^{\mu,x}_\mu, \ldots, [\![N_n]\!] \circ \Pi^{\mu,x}_\mu, [\![x]\!]_{\mu,x}, \Pi^{\mu,x}_\mu \rangle) \\
&= \Phi([\![P]\!]_{\mu,x} \circ \langle [\![N_1]\!], \ldots, [\![N_n]\!] \rangle \times id_{\Gamma^x}) \\
&= \Phi([\![P]\!]_{\rho,x}) \circ \langle [\![N_1]\!], \ldots, [\![N_n]\!] \rangle \\
&= [\![\lambda x.P]\!]_\rho \circ \langle [\![N_1]\!], \ldots, [\![N_n]\!] \rangle
\end{aligned}$$

3.

$$\begin{aligned}
[\![\sigma(t_1, \ldots, t_n)]\!]_\rho \circ \langle [\![N_1]\!]_\mu, \ldots, [\![N_n]\!]_\mu \rangle &= \sigma_A \circ \langle [\![t_1]\!]_\rho, \ldots, [\![t_n]\!]_\rho \rangle \\
&\qquad \circ \langle [\![N_1]\!]_\mu, \ldots, [\![N_n]\!]_\mu \rangle \\
&= \sigma_A \circ \langle [\![t_1]\!]_\rho \circ [\![\vec{N}]\!], \ldots, [\![t_n]\!]_\rho \circ [\![\vec{N}]\!] \rangle \\
&= \sigma_A \circ \langle [\![t_1[\vec{N}/\vec{x}]]\!]_\mu, \ldots, [\![t_n[\vec{N}/\vec{x}]]\!]_\mu \rangle \\
&\qquad \textit{by induction hypothesis} \\
&= [\![\sigma(t_1, \ldots, t_n)[\vec{N}/\vec{x}]]\!]_\mu
\end{aligned}$$

where $[\![\vec{N}]\!] = \langle [\![N_1]\!]_\mu, \ldots, [\![N_n]\!]_\mu \rangle$

□

Lemma 6.1.4 *If $FV(M) \subseteq \{\rho\}$, $FV((\lambda x.P)\,Q) \subseteq \{\mu\}$ and $\{\rho\} \subseteq \{\mu\}$ then*

$$[\![M[N/x]]\!]_\mu = [\![M]\!]_\rho \circ \langle \Pi^\mu_\rho, [\![N]\!]_\mu \rangle$$

Proof

$$\begin{aligned} [\![M[N/x]]\!]_\mu &= [\![M[\vec{y}, N/\vec{y}, x]]\!]_\mu \\ &= [\![M]\!]_\rho \circ \langle \Pi^\mu_\rho, [\![N]\!]_\mu \rangle \quad \textit{by lemma } 6.1.3 \end{aligned}$$

□

Theorem 6.1.5 (Soundness) *If $E \vdash_{\lambda,E} M = N$ then $[\![M]\!]_\rho = [\![N]\!]_\rho$*

Proof By induction on the length of the derivation $E \vdash_{\lambda,E} M = N$.

Base case If $E \vdash_{\lambda,E} e$ in one step then e must be a logical axiom or non-logical axiom

1. In the case of the axiom (β) we have:

$$\begin{aligned} [\![(\lambda x.P)\,Q]\!]_\rho &= ev \circ \langle [\![\lambda x.P]\!]_\rho, [\![Q]\!]_\rho \rangle \\ &= ev \circ \langle \Phi([\![P]\!]_{\rho,x}), [\![Q]\!]_\rho \rangle \\ &= ev \circ \Phi([\![P]\!]_{\rho,x}) \times id_{\Gamma^x} \circ \langle id_{\Gamma^\rho}, [\![Q]\!]_\rho \rangle \\ &= [\![P]\!]_{\rho,x} \circ \langle id_{\Gamma^\rho}, [\![Q]\!]_\rho \rangle \\ &= [\![P[Q/x]]\!]_\rho \; \textit{by lemma } 6.1.4 \end{aligned}$$

2. If $e \in E$ then by theorem 4.3.14 A satisfies e.

3. Finally for the δ rules we have:

$$\begin{aligned} [\![\pi(\langle x, y \rangle]\!]_\rho &= \pi \circ [\![\langle x, y \rangle]\!]_\rho \\ &= \pi \circ \langle \pi^\rho_x, \pi^\rho_y \rangle \\ &= \pi^\rho_x \end{aligned}$$

The axiom $\pi'(\langle x, y\rangle = y$ is similar.

$$\begin{aligned} [\![\langle \pi(x), \pi'(x)\rangle]\!]_\rho &= \langle \pi \circ \pi_x^\rho, \pi_x'^\rho \rangle \\ &= \pi_x^\rho \\ &= [\![x]\!]_\rho \end{aligned}$$

Induction step Assume $E \vdash_{\lambda,E} P = Q$ in n steps implies that for all ρ $[\![P]\!]_\rho = [\![Q]\!]_\rho$.

1.

$$\begin{aligned} & [\![P]\!]_\rho &= {} & [\![Q]\!]_\rho \textit{ by assumption} \\ \textit{implies } & [\![P]\!]_\rho \circ \Pi_\rho^{\rho,x} &= {} & [\![Q]\!]_\rho \circ \Pi_\rho^{\rho,x} \textit{ by lemma } 6.1.2 \\ \textit{implies } & [\![P]\!]_{\rho,x} &= {} & [\![Q]\!]_{\rho,x} \\ \textit{implies } & \Phi([\![P]\!]_{\rho,x}) &= {} & \Phi([\![Q]\!]_{\rho,x}) \\ \textit{implies } & [\![\lambda x.P]\!]_\rho &= {} & [\![\lambda x.Q]\!]_\rho \end{aligned}$$

and therefore the (ξ) rule is sound.

2.

$$\begin{aligned} [\![X\,P]\!]_\rho &= ev \circ \langle [\![X]\!]_\rho, [\![P]\!]_\rho \rangle \\ &= ev \circ \langle [\![X]\!]_\rho, [\![Q]\!]_\rho \rangle \textit{ by induction hypothesis} \\ &= [\![X\,Q]\!]_\rho \end{aligned}$$

The case for the rule $P = Q \supset P\,X = Q\,X$ is similar.

□

6.2 The Universal Property of $\mathcal{C}(\mathcal{M})$

We conclude this section by proving one theorem relating algebraic (equational) theories with higher order theories (presented by $\lambda^{\Sigma E}$). In the previous section the correspondence between λ calculus was used to prove the soundness of deduction in $\lambda^{\Sigma E}$ with respect to a model of (Σ, E) by using the translation from λ calculus to cartesian closed categories in [Koy82]. Below we use the correspondence in the opposite direction in the manner described prior to 3.2.2.

Let (Σ, E) be the presentation of an equational theory and suppose that $\mathcal{B}$ is a strongly normalising and Church-Rosser set of rewrite rules. Let A be a the model of (Σ, E) given by theorem 4.3.14. Then we can define an algebra for the algebraic theory presented by (Σ, E) as a product preserving functor (see [Law63, KR89]):

$$A : \mathcal{T}_{\Sigma E}^{op} \to \mathcal{C}(\mathcal{M})$$

Definition 6.2.1 *Define a functor*

$$\mathcal{R} : \mathcal{T}_{\Sigma E}^{op} \to \mathcal{C}(\mathcal{M})$$

as the product preserving functor such that on

Objects $\mathcal{R}(1_{\mathcal{T}}) = 1_{\mathcal{C}}$ *and* $\mathcal{R}(s) = \Gamma_s$ *for every* $s \in S$

Morphisms *For every map* $\theta : s \to \mu$ *in* $\mathcal{T}_{\Sigma,E}$, $\mathcal{R}(\theta) = \overline{\theta}$ *where* $\overline{\theta}$ *is the derived operation in* $\mathcal{C}(\mathcal{M})$ *given by theorem 4.3.14 For every object* ω *of* $\mathcal{T}_{\Sigma,E}$, $\mathcal{R}(id_\omega) = \mathcal{R}(\omega)$.

It is easy to verify that $\mathcal{R}$ is a functor, for example,

$$\begin{aligned}
\mathcal{R}(\theta) \circ \mathcal{R}(\psi) &\overset{def}{=} \tau \circ \overline{\theta} \circ \tau \circ \overline{\psi} \\
&\equiv_\alpha \tau \circ \overline{\theta} \circ \langle \tau \circ \overline{\psi_1}, \ldots, \tau \circ \overline{\psi_n} \rangle \\
&\equiv_\alpha \tau \circ \overline{\theta} \circ \langle \overline{\psi_1 \downarrow}, \ldots, \tau \circ \overline{\psi_n \downarrow} \rangle \\
&\equiv_\alpha \tau \circ \overline{\theta} \circ \langle \overline{\psi_1}, \ldots, \overline{\psi_n} \rangle \\
&\equiv_\alpha \mathcal{R}(\theta \circ \psi)
\end{aligned}$$

since by theorem 4.3.10 both $\tau \circ \overline{\theta} \circ \langle \overline{\psi_1}, \ldots, \overline{\psi_n} \rangle$ and $\overline{\theta} \circ \langle \overline{\psi_1 \downarrow}, \ldots, \tau \circ \overline{\psi_n \downarrow} \rangle$ are equal to $(\overline{\theta} \circ \langle \overline{\psi_1}, \ldots, \overline{\psi_n} \rangle) \downarrow$

From the calculus $\lambda^{\Sigma E}$ a cartesian closed category may be constructed in the manner of [LS86]. First notice that any term with free variables $\{x_1 \in X_{\alpha_1}, \ldots, x_n \in X_{\alpha_n}\}$ is equal to a term in which the free variables have been replaced by a single variable $z \in X_{\alpha_1 \times \ldots \times \alpha_n}$. In the case of terms with just one free variable we have the following lemma:

Lemma 6.2.2 *If M, N and P be three terms each with a single free variable: x_α, y_β and z_γ respectively, then*

$$M[N/x][P/y] = M[N[P/y]/x]$$

Proof By induction on the structure of M and the definition of substitution. □

Definition 6.2.3 *If (Σ, E) is a presentation then the category based on (Σ, E), $\mathcal{C}_{\Sigma,E}$, is defined as follows:*

Objects *the objects of $\mathcal{C}_{\Sigma,E}$ are the types of $\lambda^{\Sigma E}$;*

Morphisms *if $M(x_{\alpha_1}, \ldots, x_{\alpha_n})$ is a term in $\Lambda_{\Sigma,\alpha}(\{y_{\beta_1}, \ldots, y_{\beta_k}\})$ such that*

$$\{x_{\alpha_1}, \ldots, x_{\alpha_n}\} \subseteq \{y_{\beta_1}, \ldots, y_{\beta_k}\}$$

and where $\rho = y_{\beta_1} \ldots y_{\beta_k}$ is an arrow in $Hom(\beta_1 \times \ldots \times \beta_k, \alpha)$ then the pair:

$$\langle z_{\beta_1}, \ldots, z_{\beta_k}, M(\pi^\rho_{x_1}(z), \ldots, \pi^\rho_{x_n}(z))\rangle$$

is an arrow in $\mathcal{C}_{\Sigma E}(\beta_1 \times \ldots \times \beta_k, \alpha)$.

The identity arrow for an object α is the pair $\langle x_\alpha, x_\alpha\rangle$. Composition is defined by

$$\langle x_\alpha, M(x)\rangle \circ \langle y_\gamma, N_\alpha(y)\rangle = \langle y_\gamma, M[N(y)/x]\rangle$$

Composition is associative by lemma 6.2.2.

Following [LS86] we can show that $\mathcal{C}_{\Sigma,E}$ is a cartesian closed category.

Theorem 6.2.4 *$\mathcal{C}_{\Sigma,E}$ is a cartesian closed category.*

Proof The proof is similar to that for the simply typed λ calculus given in [LS86]. We prove the theorem by again first defining the cartesian closed structure and then verifying the axioms of definition 3.2.1.

1. the terminal object of $\mathcal{C}_{\Sigma,E}$ is the type *1* and the arrow from any any type α to *1* is given by $\langle x_\alpha, *\rangle$;

2. products are defined by:

$$\langle\langle x_\alpha, P(x)\rangle, \langle x_\alpha, Q(x)\rangle\rangle \overset{def}{=} \langle x_\alpha, \langle P(x), Q(x)\rangle\rangle$$

with projections:

$$\pi \overset{def}{=} \langle x_{\alpha\times\beta}, \pi(x)\rangle \ \ and \ \ \pi' \overset{def}{=} \langle x_{\alpha\times\beta}, \pi'(x)\rangle$$

3. evaluation is defined by:

$$ev_{\alpha\beta} \overset{def}{=} \langle z_{\alpha\to\beta\times\alpha}, \pi(z)\ \pi'(z)\rangle$$

and if $f \equiv \langle x_{\alpha\times\beta}, f(x)\rangle$ the exponential transpose by of f is given by:

$$(f)^* \overset{def}{=} \langle x_\alpha, \lambda y_\beta . f(\langle x, y\rangle)\rangle$$

The axioms of definition 3.2.1 are now shown to hold by straightforward calculation, for example, the axiom $ev \circ f^* \times id = f$:

$$\begin{aligned} ev \circ f^* \times id &= \langle z_{\alpha\to\beta\times\alpha}, \pi(z)\ \pi'(z)\rangle \circ \langle x_{\alpha\times\beta}, \langle \lambda y_\beta . f(\langle \pi(x), y\rangle), \pi'(x)\rangle \\ &= \langle x_{\alpha\times\beta}, f(\langle \pi(x), \pi'(x)\rangle)\rangle \\ &= \langle x_{\alpha\times\beta}, f(x)\rangle \end{aligned}$$

□

We can relate $\mathcal{T}_{\Sigma,E}$ and $\mathcal{C}_{\Sigma,E}$ easily enough by a product preserving functor $\mathcal{F} : T^{op}_{\Sigma E} \to \mathcal{C}_{\Sigma E}$. More formally $\mathcal{F}$ can be defined on objects as follows:

$$\mathcal{F}(1) = 1 \quad \mathcal{F}(s) = s \ \ for \ \ any \ \ s \in S \quad \mathcal{F}(s_1 \ldots s_n) = s_1 \times \ldots \times s_n$$

and on arrows by:

$$\begin{aligned} \mathcal{F}(\emptyset_\omega : \epsilon \to \omega) &= \langle x_{s_1\times\ldots\times s_n}, *\rangle \\ & \quad\ if \ \ \omega = s_1 \ldots s_n \\ \mathcal{F}(\pi^\omega_{s_i}) &= \langle x_{s_1\times\ldots\times s_n}, \pi^\omega_{x_i}(x)\rangle \\ \mathcal{F}(\theta) &= \langle x_{s_1\times\ldots\times s_n}, \theta(\pi^\omega_{x_1}(x), \ldots, \pi^\omega_{x_n}(x))\rangle \end{aligned}$$

$$\begin{aligned} & \quad for\ \ any\ \ \theta \in \Sigma_{\omega,s} \\ \mathcal{F}(\theta) &= \mathcal{F}(\varphi) \circ \mathcal{F}(\psi) \\ & \quad if\ \ \theta = \varphi \circ \psi\ \ in\ \ \mathcal{T}_{\Sigma E} \\ \mathcal{F}(\langle \theta_1, \ldots, \theta_n \rangle) &= \langle \mathcal{F}(\theta_1), \ldots, \mathcal{F}(theta_n) \rangle \end{aligned}$$

Once an algebra for $\mathcal{T}_{\Sigma,E}$ is chosen in a cartesian closed category $\mathcal{K}$ then this essentially determines an algebra for $\mathcal{C}_{\Sigma,E}$ as a functor into $\mathcal{K}$.

Theorem 6.2.5 *If $\mathcal{K}$ is a cartesian closed category and $A : \mathcal{T}_{\Sigma E}^{op} \to \mathcal{K}$ any $\mathcal{T}_{\Sigma,E}$ algebra then A may be uniquely extended to a functor $A' : \mathcal{C}_{\Sigma E} \to \mathcal{K}$ which preserves the cartesian closed structure of $\mathcal{C}_{\Sigma,E}$ and make the diagram*

$$\begin{tikzcd} \mathcal{T}_{\Sigma E}^{op} \arrow[r, "F"] \arrow[dr, "A"'] & \mathcal{C}_{\Sigma E} \arrow[d, "A'"] \\ & \mathcal{K} \end{tikzcd}$$

commute.

Proof Define A' as follows:

on objects A' is defined inductively by:

$$\begin{aligned} A'(1) &= 1_{\mathcal{K}} \\ A'(s) &= A(s)\ \ for\ \ any\ \ s \in S \\ A'(\alpha_1 \times \ldots \times \alpha_n) &= A'(\alpha_1) \times \ldots \times A'(\alpha_n) \\ A'(\alpha \to \beta) &= A'(\alpha) \to A'(\beta) \end{aligned}$$

on arrows A' is defined by induction on the terms of $\lambda^{\Sigma E}$ by:

$$\begin{aligned} A'(\langle x_\alpha, * \rangle) &= \bigcirc_{A'(\alpha)} \\ A'(\langle x_{s_1 \times \ldots \times s_n}, \sigma(\pi^{\omega}_{x_1}(x), \ldots, \pi^{\omega}_{x_n}(x)) \rangle) &= A(\sigma)\ \ where\ \ \sigma : s \to \omega\ \ in\ \ \mathcal{T}_{\Sigma E} \end{aligned}$$

$$\begin{aligned}
A'(\langle P, Q\rangle) &= \langle A'(P), A'(Q)\rangle \\
A'(\langle x_{\alpha\times\beta}, \pi(x)\rangle) &= \pi_{A'(\alpha)A'(\beta)} \\
A'(\langle x_{\alpha\times\beta}, \pi'(x)\rangle) &= \pi'_{A'(\alpha)A'(\beta)} \\
A'(\langle x_\alpha, \lambda y_\beta . f(\langle x, y\rangle)\rangle) &= (A'(\langle z_{\alpha\times\beta}, f(z)\rangle))^* \\
A'(\langle z_{\alpha\to\beta\times\alpha}, \pi(z)\,\pi'(z)\rangle) &= ev_{A'(\alpha)A'(\beta)}
\end{aligned}$$

It now remains to show that the diagram above commutes and that A' is the only functor which preserves the cartesian closed structure and makes the diagram commute.

That A' makes the diagram commute follows by direct calculation from the definition of $\mathcal{F}$, for example, if $\theta : s \to \omega$ is an n-ary operation in $\mathcal{T}_{\Sigma,E}$ and $\theta \in \Sigma_{\omega,s}$:

$$\begin{aligned}
A' \circ \mathcal{F}(\theta) &= A'(\langle x_{s_1\times\ldots\times s_n}, \theta(\pi^\omega_{x_1}(x), \ldots, \pi^\omega_{x_n}(x))\rangle) \\
&= A(\sigma) \ by\ definition\ A'
\end{aligned}$$

To show that A' is the unique such functor making the diagram commute consider another functor $\mathcal{G}$ making the diagram commute. First by induction on the types of $\mathcal{C}_{\Sigma,E}$ for all types α, $A'(\alpha) = \mathcal{G}(\alpha)$.

basic case for the type *1*

$$A'(1) = A(1) = \mathcal{G}(1)$$

and for any $s \in S$

$$A'(s) = A(s) = \mathcal{G}(s)$$

induction case assume that $A'(\alpha) = \mathcal{G}(\alpha)$ and $A'(\beta) = \mathcal{G}(\beta)$ then

$$\begin{aligned}
A'(\alpha\times\beta) &= A' \times A')(\beta) \\
&= \mathcal{G}(\alpha) \times \mathcal{G}(\beta) \quad by\ induction\ assupmtion \\
&= \mathcal{G}(\alpha\times\beta)
\end{aligned}$$

and similarly $A'(\alpha \to \beta) = \mathcal{G}(\alpha \to \beta)$.

Now by induction on the structure of terms of $\lambda^{\Sigma E}$, we show that for any arrow $f : \alpha \to \beta$ in $\mathcal{C}_{\Sigma,E}$, $A'(f) = \mathcal{G}(f)$. Let $\langle x_\alpha, \phi(x)\rangle$ be an arrow $\alpha \to \beta$ in $\mathcal{C}_{\Sigma,E}$. Now we proceed by an induction on the structure of terms in $\lambda^{\Sigma E}$. Since both A' and $\mathcal{G}$ make the diagram commute then if $\phi(x)$ is an algebraic term in which no higher order variables $\pi^{\omega}_{\gamma\to\mu}$ occur then

$$A'(\langle x_\alpha, \phi(x)\rangle) = A(\langle x_\alpha, \phi(x)\rangle) = \mathcal{G}(\langle x_\alpha, \phi(x)\rangle)$$

which is true for the cases when $\phi(x)$ is *, a constant $\sigma \in \Sigma_{\epsilon,s}$ and an algebraic term $\sigma(t_1, \ldots, t_n)$. If $\phi(x) = \lambda y_\beta.P(\langle x, y\rangle)$ is an abstraction then

$$\begin{aligned} A'(\langle x_\alpha, \lambda y_\beta.P(\langle x,y\rangle)\rangle) &= (A'(P(z_{\alpha\times\beta}))^* \\ &= (\mathcal{G}(P(z_{\alpha\times\beta}))^* \quad \textit{by induction assumption} \\ &= \mathcal{G}(\langle x_\alpha, \lambda y_\beta.P(\langle x,y\rangle)\rangle) \\ &\quad \textit{since } \mathcal{G} \textit{ preserves the} \\ &\quad \textit{cartesian closed structure of } \mathcal{C}_{\Sigma E} \end{aligned}$$

and similarly

$$A'(\langle x_\alpha, P(x)\, Q(x)\rangle) = \mathcal{G}(\langle x_\alpha, P(x)\, Q(x)\rangle)$$

The cases for pairing and projections follow in a similar manner. □

Corollary 6.2.6 *Every $\mathcal{T}_{\Sigma,E}$ algebra may be uniquely extended to a $\mathcal{C}_{\Sigma,E}$ algebra in $\mathcal{C}(\mathcal{M})$.*

Algebras in $\mathcal{C}(\mathcal{M})$ can be related back to algebras in $\mathcal{SET}$ by considering the forgetful functor from $\mathcal{C}(\mathcal{M})$ to $\mathcal{SET}$.

Definition 6.2.7 *The (forgetful) functor $U_{\mathcal{C}}$ is defined as follows:*

1. *for any object a of $\mathcal{C}(\mathcal{M})$:*

$$U_{\mathcal{C}}(a) = \{m \in \mathcal{M} \mid a\, m = m\}$$

2. *for any arrow (b,f,a) in $\mathcal{C}(\mathcal{M})$:*

$$U_\mathcal{C}(f) = \overline{f} : U_\mathcal{C}(a) \rightarrow U_\mathcal{C}(b)$$

where $\overline{f}(m) = f \cdot m$ and $\cdot$ is the binary operation in $\mathcal{M}$.

$U_\mathcal{C}$ is the forgetful functor from $\mathcal{C}(\mathcal{M})$ to $\mathcal{SET}$. If $\mathcal{T}_{\Sigma,E}$ is an algebraic theory then the models of $\mathcal{T}_{\Sigma,E}$ in $\mathcal{C}(\mathcal{M})$ give rise to models of $\mathcal{T}_{\Sigma,E}$ in $\mathcal{SET}$ by *transport* along $U_\mathcal{C}$ (see [KR89]), for if A is a $\mathcal{T}_{\Sigma,E}$ algebra in $\mathcal{C}(\mathcal{M})$ then an algebra $\mathcal{L}_A$ can be defined as the functor making the diagram below:

$$\begin{tikzcd}
 & \mathcal{T}^{op}_{\Sigma E} \arrow[dl, "\mathcal{L}_A"'] \arrow[dr, "A"] & \\
\mathcal{SET} & & \mathcal{C}(\mathcal{M}) \arrow[ll, "U_\mathcal{C}"]
\end{tikzcd}$$

commute. For every sort $s \in S$, $A'(s)$ is simply the *set* of all values of type Γ_s in $\mathcal{C}(\mathcal{M})$.

Since $Hom(1,\_) : \mathcal{T}_{\Sigma E} \rightarrow \mathcal{SET}$ is the initial $\mathcal{T}_{\Sigma E}$ algebra then there is a unique homomorphism (natural transformation) $\iota : Hom(1,\_) \stackrel{\cdot}{\rightarrow} U_\mathcal{C} \circ A$ but in general $U_\mathcal{C} \circ A$ is not initial, for example, in the list algebra of example 5.3.2 we have in Γ_{list} the elements $[\![\Omega]\!]$ $[\![inj^{list}_{\Gamma_{Nat} \times \Gamma_{List}} \langle n, \Omega \rangle]\!]$ and many more expressions which involve the denotations of unsolvable terms. If (Σ, E) is a presentation then the scheme for constructing algebras in chapter 4 associates (the denotation of) a (β) normal form with every term of the free Σ algebra but the expressions above do not have (β) normal forms [Bar84] and so are not the image of any term in the initial algebra. Consequently there is *junk* in the carrier $U_\mathcal{C} \circ A(s)$ and so the *list* algebra $U_\mathcal{C} \circ A$ is not initial.

Chapter 7

Conclusions and Further Work

In this dissertation we have looked at the construction of algebras in λ algebras from sets of strongly normalising and Church-Rosser rewrite rules and have given sufficient conditions under which such algebras can be constructed. We have then presented a simply typed λ calculus for reasoning about algebras within the category $\mathcal{C}(\mathcal{M})$ which is sound but not complete. Turning to term rewriting properties we showed that given a presentation (Σ, E), if a strongly normalising Church-Rosser set of rewrite rules $\mathcal{B}$ can be obtained from E then the rewrite relation obtained by combining the rules of $\mathcal{B}$ with those of E is also strongly normalising and Church-Rosser.

We started by giving a simple *universe*, $\mathcal{C}(\mathcal{M})$, in which implementations can be made. Equational specifications then denote classes of algebras in $\mathcal{C}(\mathcal{M})$ and theorem 4.3.14 simply gives conditions under which one algebra from this class may be "picked". Surprisingly at first $\mathcal{C}(\mathcal{M})$ contains very few *basic* objects but these are enough to define the data type of natural numbers (as $Y(\lambda N.1 + N)$) and the partial recursive number theoretic functions[1].

An unsatisfactory element of $\mathcal{C}(\mathcal{M})$ is the treatment of sums. The addition of a sum construct to the original category $\mathcal{K}(\mathcal{M})$ was necessary for us to be able to reason about $\mathcal{C}(\mathcal{M})$ but this addition required some technical detail which

[1] Recall that we only required strong normalisation of term rewriting systems so that we could *construct* algebras in $\mathcal{C}(\mathcal{M})$ but did not require that all arrows in $\mathcal{C}(\mathcal{M})$ be the denotations of strongly normalising λ terms.

could perhaps be avoided if we had used a model theoretic approach rather than insisting that λ calculus be the target language for our constructions. Another candidate could be the ideal models of [MPS86] since they too embody the idea of a universe of values from which all types are retracts (or in the case of [MPS86] *ideals*) and already has a definition of sum. Another feature of the ideal model of [MPS86] is that it is a model of type polymorphism as found in languages like Standard ML. It would therefore be interesting to carry out the same program as in this dissertation on this *ideal* model to learn more of the interaction between type polymorphism and polymorphic equational specifications. We feel that this is one useful avenue of future research. Another avenue of research would attempt to remove some of the shortcomings of $\mathcal{C}(\mathcal{M})$. One example of this is the restriction to total functions. Specifications often define only partial functions which could be studied by choosing a suitable category of types and partial functions. This would be especially useful for understanding *expansion* which is required before case analysis on the representations of terms can be performed.

The algorithm for constructing τ in chapter 4.3 makes a number of assumptions.

1. The first assumption is that products and sums, at least, are present in any category in which τ is to be defined and that there are arrows, as in the equations 6 of chapter 4.1, which *expand* products over sums. The algorithm for generating τ, however, does not rely on the existence of coproducts.

2. The second assumption is that colimits of ω^{op} chains involving the sum and product functors exist (or dually limits of ω chains) in any category in which τ is to be constructed. In $\mathcal{C}(\mathcal{M})$ τ is defined as an arrow between types representing sorts all of which are limits of ω^{op} chains. The construction corresponding to colimits in $\mathcal{C}(\mathcal{M})$ is the least fixed point construction (see theorem 3.1.18).

It is not easy to see how to weaken either of these two assumptions since they are what is required of a category in order to define the functors corresponding to signatures (definition 2.3.6) and to find solutions for recursive domain equations.

The algorithm to construct τ is presented abstractly and one remaining task is to give more efficient algorithms to perform computations within an algebra.

The proof that $\lambda^{\Sigma E}$ is sound with respect to algebra in $\mathcal{C}(\mathcal{M})$ is itself a simple adaptation of the proof in [Koy82] that an arbitrary cartesian closed category with an object U such that $U \rightarrow U$ is a retract of U and a terminal object 1 is a λ algebra. The work involved in showing this theorem was in constructing a cartesian closed category from $\mathcal{M}$.

The calculus $\lambda^{\Sigma E}$ which we use to reason about algebras in $\mathcal{C}(\mathcal{M})$ is a simple variant of the simply typed λ calculus. Theorem 5.2.8, theorem 5.2.9 and theorem 5.2.10 state that the transition from equational reasoning to higher order reasoning is a smooth one as long as one does not consider signatures with operations of higher order types (see examples 5.4.1 and 5.4.2). The derived rules given in the proof of theorem 5.1.5 are a means of translating proofs of equational theorems from a set of equations E using the many sorted system of [EM85] to proofs of equational theorems using simply typed λ calculus with non-logical axioms E.

There are two problems however with $\lambda^{\Sigma E}$.

1. $\lambda^{\Sigma E}$ is sound with respect to models of algebras in $\mathcal{C}(\mathcal{M})$ (theorem 6.1.5) but not in general complete.

2. $\lambda^{\Sigma E}$deals only with the equational calculus.

The first of these is a problem if we wish to reason about algebraic terms which may involve infinite computations while the second problem would need to be overcome if we were to consider more powerful specification logics such as conditional equational logic or first order logic with equality. These areas need to be further investigated if we are to achieve a fuller understanding of the relationship between implementations (algebras) and specifications (logic).

Finally an indication of the use of our theory. Consider a simple extension to Standard ML's *abstype* facility [HMT90] which allows axioms. With this extension in mind we write the *Group* abstract data type of example 4.4.2 as follows:

```
abstype $\alpha$ Group with
    ops
        $e : \rightarrow \alpha\ Group$
        $+ : \alpha\ Group * \alpha\ Group \rightarrow \alpha\ Group$
        $- : \alpha\ Group \rightarrow \alpha\ Group$
    axioms $\forall\ x, y, z \in X$,
        $e + x = x$
        $-(x) + x = e$
        $(x + y) + z = x + (y + z)$
    end
```

Consider a function $f : \alpha\ Group \rightarrow \beta$ for some type β. Then to evaluate the application of f to a member a of α *Group* we would evaluate:

$$f(a) \overset{def}{=} f(\tau(a))$$

Adding constructs like these to programming languages to increase the power of the available abstraction facilities is again another area of future research.

Bibliography

[AM75] Michael A. Arbib and Ernest G. Manes. *Arrows, Structures and Functors.* Academic Press, New York, 1975.

[Bar84] H. P. Barendregt. *The Lambda-Calculus: its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, second edition, 1984.

[Bir35] G. Birkhoff. On the Structure of Abstract Algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433 - 454, October 1935.

[Bir48] G. Birkhoff. *Lattice Theory.* Colloquium Publications. American Mathematical Society, 1948.

[BM84] K. B. Bruce and Albert R. Meyer. The Semantics of the Second Order Polymorphic Lambda Calculus. In *Semantics of Data Types, LNCS 173*, pages 131 - 144. Springer-Verlag, June 1984.

[Bou86] N. Bourbaki. *The Theory of Sets.* Elements of Mathematics. Hermann Publishers in Arts and Sciences, 1986.

[BT88] V. Breazu-Tannen. Combining Algebra and Higher Order Types. In *Logics in Computer Science*, pages 82 - 90. Computer Society Press, July 1988.

[BV85] C. Beierle and A. Voss. Implementation Specifications. In H. J. Kreowski, editor, *Recent Trends in Data Type Specification*, pages 39 - 53. Springer-Verlag, 1985.

[BW85] M. Barr and C. Wells. *Toposes, Triples and Theories.* Springer Verlag, 1985.

[CF68] H. P. Curry and R. Feys. *Combinatory Logic.* North Holland, 1968.

[Coh81] Paul M. Cohn. *Universal Algebra*, volume 6 of *Mathematics and Its Applications.* D. Reidel Publishing Company, second edition, 1981.

[Col87] P. A. Collier. Type inference in the presence of a basic type hierachy. In *Actus de la VII, Conferencia de la Chilena de Ciencia de la Computation*, 1987.

[Cut80] N. J. Cutland. *Computability : An Introduction to Recursive Function Theory.* Cambridge University Press, 1980.

[CW85] Luca Cardelli and Peter Wegner. On understanding types, data abstraction and polymorphism. *Computing Surveys*, 17(4):471 - 522, December 1985.

[DM82] Luis Damas and Robin Milner. Principle Type Schemes and Functional Programs. In *Principles of Programming Languages*, pages 207 - 212, 1982.

[Ehr78] H. D. Ehrich. Extensions and implementations of abstract data types. In *Mathematical Foundations of Computer Science, LNCS 64.* Springer Verlag, 1978.

[Ehr82] H. D. Ehrich. On the theory of specification, implementation and parameterisation of abstract data types. *Journal of the A.C.M*, 29(1):206 - 227, January 1982.

[EK83] H. Ehrig and H. J. Kreowski. Compatibility of parameter passing and implementation of abstract data types. *Theoretical Computer Science*, 27:255 - 286, 1983.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1.* EATCS: Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[EM90] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 2 : Module Specifications and Constraints.* EATCS: Monographs on Theoretical Computer Science. Springer-Verlag, 1990.

[Foo85] N. Foo. Algebraic specifications as solutions of implementation equations. Technical Report Technical Report 263, Basser Dept. of Computer Science, University of Sydney, May 1985.

[Gal] J. H. Gallier. On Girard's Candidates de Reductibilite. Draft Manuscript.

[GB79] J. A. Goguen and R. M. Burstall. The semantics of clear, a specification language. In *Abstract Software Specifications, LNCS 86.* Springer-Verlag, 1979.

[GJ82] Huet G. and Hullot J. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25:239 – 266, 1982.

[GM81] J. A. Goguen and J. Meseguer. Completeness of many sorted equational logic. *Sigplan Notices*, 16(7):24 – 32, July 1981.

[GM82] J. A. Goguen and J. Meseguer. Universal realization, persistent interconnection and implementation of abstract data types. In *ICALP '82, LNCS 140*, pages 265 - 281, July 1982.

[GM86] Joseph A. Goguen and Jose Meseguer. Remarks on remarks on many sorted equational logic. *Bulletin of the EATCS*, 30:66 - 73, October 1986.

[GM87] Joseph A. Goguen and Jose Meseguer. Remarks on many sorted equational logic. *Sigplan Notices*, 22(4):41 – 48, April 1987.

[Gog78] Joseph A. Goguen. Abstract errors for abstract data types. In E. J. Newhold, editor, *Formal Description of Programming Concepts.* North Holland Publishing Co., 1978.

[GW88] J. A. Goguen and T. Winkler. Introducing OBJ3. Technical Report SRI-CLS-88-9, SRI International, August 1988.

[HEP80a] B. Mahr H. Ehrig, H. J. Kreowski and P. Padawitz. Compound algebraic implementations. In *Mathematica Foundations of Computer Science, LNCS 88*, pages 231 – 245. Springer-Verlag, 1980.

[HEP80b] H. J. Kreowski H. Ehrig and P. Padawitz. Algebraic implementation of abstract data types : Concepts, syntax, semantics and correctness. In *Automata, Languages and Programming 80, LNCS 85.* Springer-Verlag, July 1980.

[HMT90] R. Harper, R. Milner, and M. Tofte. *The Definition of Standard ML.* The MIT Press, 1990.

[HO80] Gërard Huet and Derek C. Oppen. Equations and Rewrite Rules. In Ronald V. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349 – 405. Academic Press, 1980.

[HO82] Christopher M. Hoffman and Michael J. O'Donnell. Programming with Equations. *A.C.M Transactions on Programming Languages and Systems*, 4(1):83 – 112, January 1982.

[HS79] Horst Herrlich and George E. Strecker. *Category Theory.* Sigma Series in Pure Mathematics. Heldermann-Verlag, 1979.

[HS86] R. J. Hindley and J. P. Seldin. *Introduction to Combinators and Lambda-Calculus.* Cambridge University Press, 1986.

[Hup80] Ulrich L. Hupbach. Abstract Implementation of Abstract Data Types. In *Mathematical Foundations of Computer Science, LNCS 88*, pages 291 – 304. Springer-Verlag, 1980.

[JABW81] J. V. Tucker J. A. Bergstra, M. Broy and M. Wirsing. On the Power of Algebraic Specifications. In *Mathematical Foundations of Computer Science, LNCS 118.* Springer-Verlag, 1981.

[JAG75] E. G. Wagner J. B. Wright J. A. Goguen, J. W. Thatcher. An introduction to categories, algebraic theories and algebras. Technical Report 23477, IBM Thomas J. Watson Research Centre Mathematical Sciences Department, 1975.

[JAGW78] J. W. Thatcher J. A. Goguen and E. G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In Yeh, editor, *Current Trends in Programming Methodology IV*, pages 80 – 144. Prentice Hall, New York, 1978.

[JRHS72] B. Lercher J. R. Hindley and J. P. Seldin. *Introduction to Combinatory Logic.* London Mathematical Society Lecture Notes Series 7. Cambridge University Press, 1972.

[KA84] Samuel Kamin and Myla Archer. Partial implementations of abstract data types: A dissenting view on errors. In *LNCS 174: Semantics of Data Types.* Springer-Verlag, 1984.

[KB70] D. E. Knuth and P. B. Bendix. Simple problems in universal algebra. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 203 – 297. Pergammon Press, 1970.

[Klo80] J.W Klop. *Combinatory Reduction Systems.* Tracts in Computer Science. CWI, 1980.

[Koy82] C. P. J. Koymans. Models of the lambda-calculus. *Information and Control*, 52:306 – 332, 1982.

[KR89] A. Kock and G.E Reyes. Doctrines in categorical logic. In John Barwise, editor, *Hanbook of Mathematical Logic*, pages 283 – 313. Elsevier Science Publishers, fifth edition, 1989.

[KS81] Herbert A Klaeren and Martin Schulz. Computable algebras, word problems and canonical term algebras. In *LNCS 104: 5th GI Conference*, Karlsruhe, March 1981.

[KS85] Deepak Kapur and Mandayam Srivas. A rewrite rule based approach for synthesising abstract data types. In *LNCS 185: Mathematical Foundations of Software Development*. Springer-Verlag, 1985.

[Kur65] A.G. Kurosh. *General Algebra*. Pergamon, 1965.

[Lam74] Joachim Lambek. Functional completeness of cartesian closed categories. *Annals of Mathematical Logic*, 6:259 –292, 1974.

[Lam80a] Joachim Lambek. From lambda calculus to cartesian closed categories. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 376 – 402. Academic Press, 1980.

[Lam80b] Joachim Lambek. From type to sets. *Advances in Mathematics*, 36:113 – 164, 1980.

[Law63] F.W. Lawvere. *Functorial Semantics of Algebraic Theories*. PhD thesis, University of Columbia, 1963.

[LG85] Bloom S. L. and Wagner E. G. Many-sorted Theories and their Algebras with some Applications to Data Types. In Maurice Nivat and John C. Reynolds, editors, *Algebraic Methods in Semantics*, pages 133 – 168. Cambridge University Press, 1985.

[LS81] D.J. Lehmann and M.B. Smyth. Algebraic specifications of data types: A synthetic approach. *Mathematical Systems Theory*, 14:97 – 139, 1981.

[LS86] J. Lambek and P.J. Scott. *An Introduction to Higher Order Categorical Logic*. Cambridge Studies in Advanced Mathematics 7. Cambridge University Press, 1986.

[MA86] Ernest G. Manes and Michael A. Arbib. *Algebraic Approaches to Program Semantics.* Springer-Verlag, Berlin, 1986.

[Mac71] Saunders Maclane. *Categories for the Working Mathematician.* Number 5 in Graduate Texts in Mathematics. Springer-Verlag, 1971.

[MB67] Saunders Maclane and Garrett Birkhoff. *Algebra.* Macmillan Company, 1967.

[Mey82] Albert R. Meyer. What is a Model of the Lambda-calculus? *Information and Control*, 52:87 - 122, 1982.

[Mil77] Robin Milner. A Theory of Type Polymorphism in Programming. Internal Report CSR-9-77, University of Edinburgh, Department of Computer Science, September 1977.

[Mil78] George J. Milne. *A Mathematical Model of Concurrent Computation.* PhD thesis, University of Edinburgh, March 1978.

[MPS86] David MacQueen, Gordon Plotkin, and Ravi Sethi. An Ideal Model for Recursive Polymorphic Types. *Information and Control*, 71:0 - 0, 1986.

[MT90] R. Milner and M. Tofte. *Commentary on Standard ML.* MIT press, 1990.

[MW85] Zohar Manna and Richard Waldinger. *The Logical Basis for Computer Programming.* Addison Wesley, 1985.

[New42] M.H.A Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2), April 1942.

[Nou80] C. Farshid Nourani. A model-theoretic approach to specification, extension and implementation. In *International Symposium on Programming, LNCS 83*, pages 282 - 297. Springer-Verlag, 1980.

[Nou83] C. Farshid Nourani. Abstract implementations and their correctness proofs. *Jouranl of the A.C.M*, 30(2):343 - 359, April 1983.

[O'D77] Michael J. O'Donnell. *Computing Systems Described by Equations.* Springer-Verlag, 1977. LNCS 58.

[PG81] Kamran Parsaye-Ghomi. *Higher Order Abstract Data Types.* PhD thesis, University of California, Los Angeles, 1981.

[Plo77] G. D. Plotkin. Lcf considered as a programming language. *Theoretical Computer Science*, 5:223 - 255, 1977.

[Poi86] Axel Poigné. On specifications, theories and models with higher types. *Information and Control*, 68(1):1 - 43, March 1986.

[PV85] Axel Poigné and Josef Voss. On the implementation of abstract data types by programming languages. In *Mathematical Foundations of Software Development, LNCS 185*, volume 1. Springer-Verlag, 1985.

[Rey85] John C. Reynolds. Three approaches to typing. In *Mathematical Foundations of Software Development, LNCS 185*, volume 1, pages 97 - 138. Springer-Verlag, March 1985.

[Sco71] D. Scott. Continuous lattices. Technical Monograph PRG-7, Princeton University, August 1971.

[Sco76] D. Scott. Data types as lattices. *SIAM Journal of Computing*, 5(3):552 - 587, September 1976.

[Sco80] D. Scott. Relating Theories of the Lambda Calculus. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 402 - 450. Academic Press, 1980.

[SP82] M.B. Smyth and G. D. Plotkin. The category theoretic solution of recursive domain equations. *SIAM Journal of Computing*, 11(4):761 - 783, November 1982.

[ST86] D. Sannella and A. Tarlecki. Toward Formal Development of Programs from Algebraic Specifications: Implementations Revisited. Technical Monograph ECS-LFCS-86-17, Laboratory for the Foundations of Computer Science, December 1986.

[ST89] D. Sannella and A. Tarlecki. Toward Formal Development of ML Programs: Foundations and Methodology. Technical Monograph ECS-LFCS-89-71, Laboratory for the Foundations of Computer Science, February 1989.

[Sto77] J. E. Stoy. *Denotational Semantics: The Scott Strachey Approach to Programming Language Theory.* MIT Press, 1977.

[SW81] D. Sannella and M. Wirsing. Implementation of Parameterised Specifications. In *Automata Languages and Programming, LNCS 140*, pages 473 - 488. Springer-Verlag, July 1981.

[Tai67] W.W. Tait. Intensional Interpretation of Functionals of Finite Type. *Journal of Symbolic Logic*, 32:198 - 212, 1967.

[TJWB77] Wagner E. G. Thatcher J. W. and Wright J. B. Initial algebra semantics and continuous algebras. *Journal of the A.C.M*, 24(1):68 - 95, January 1977.

[Tur] D. A. Turner. Miranda : A non-strict functional language with polymorphic types. To appear in Proceedings IFIP International Conference on Functional Programming Languages and Computer Architecture, Nancy September 1985.

[VG78] Guttag J. V. and Horning G. The algebraic specification of abstract data types. *Acta Informatica*, 10:27 - 54, 1978.

[Wad76] C. Wadsworth. The relation between computational and denotational properties for scott's d infinity models of the lambda calculus. *Siam Journal of Computing*, 5(3), September 1976.

[Zil79] N. Zilles. An introduction to data algebras. In *Abstract Software Specifications, LNCS 86*, 1979.