Higher Dimensional Syntax

Martin Wehr
University of Edinburgh, Department of Computer Science (LFCS)
Mayfield Road, Edinburgh EH9 3JZ, Scotland U.K.
email: wehr@dcs.ed.ac.uk web: http://www.ed.ac.uk/home/wehr

May 9, 1999

Abstract

The aim of this paper is to present the notion of higher-dimensional syntax, which is a hierarchy of languages. Each term of a n-dimensional language will be typed by terms of the underlying $n-1$-dimensional language. This is an application of the emerging higher-dimensional category notions.

On the lower dimensions live notions of syntax which are already known. So dimension zero corresponds to words, dimension one corresponds to equation-free algebraic theories and dimension two corresponds to algebraic data types available in programming languages like ML and Haskell. In this paper I show that this hierarchy continues for higher dimensions and contributes new notions of syntax. As expected syntax comes with a structural induction principle which allows definition of several operation. The operations of substitution, unification and type inference have been investigated. Further the operation of recursive iteration schemes like generalised `map` and `fold` algorithms have been presented [Weh98].

This paper concentrates on the issue of a categorical characterisation of higher dimensional syntax using notions which are partly folklore (free monoidal object in a monoidal category) partly are new higher-dimensional category notions. I will characterise higher dimensional syntax as a left globular set having structure of a free Lawvere theory at each dimension with a right action and a right coaction of the lower dimension language. The right action is reminiscent to the substitution property in polymorphic type theories. The right coaction is reminiscent to the minimal typing property of a polymorphic type theory.

1 Introduction

Motivation

The special session on "Higher-order syntax" on this years LICS [FPT99, Hof99, GP99] shows renewed interest on understanding the notion of syntax. Higher-order syntax is concerned with binding. That is, syntax with parameters taking as arguments, syntax with parameters ... and so on. Instead of higher dimensional syntax is concerned with the notion of typing relation. It is a trivial observation that in a type system the terms as well as the types are syntactic items related by typing. In Barendregt's lambda-cube [Bar92] a third syntactic item of *kinds* shows up. In this case the kinds are in typing relation to to types like types are in typing relation to terms. The notion of higher dimensional syntax characterises these hierarchies of typing relation. A syntactic item of dimension n will be in typed by an item of of next lower dimension $n-1$.

The construction

First recall that John Baez "microcosm principle" [BD98] tells that in order to characterise categorically the notion of a monoid we need the structure of a monoidal category. So in a monoidal category $(\mathcal{C}, \otimes, I)$ we get a notion of a monoidal object M with morphisms $m : M \otimes M \to M$ and $u : I \to M$ satisfying the obvious commuting diagrams. Also recall in Dubuc [Dub74] we find that the free monoidal object $A^{*\otimes}$ to a object A exists if the category $\mathcal{C}$ is cocomplete and the tensor preserves colimits in both arguments. Technically we present several comma categories with monoidal structure. We use the notation $\widetilde{A} := \mathcal{C} \downarrow A$ to denote the comma category to an object $A \in \mathcal{C}$.

Dimension zero

Applying the above to the monoidal category $(Set, \times, 1)$ we get to any object S (t.i. a set of letters) the free monoid $S^{*\times}$ (t.i. the words over S with juxtaposition as multiplication and the empty word as neutral element). This will be dimension zero of higher dimensional syntax.

Dimension one

Using the words $S^{*\times}$ we are able to define S-sorted signatures. Fist we define *sorts* as the set $S^{srt} := S^{*\times} \times S$ to get the comma category $\widetilde{S^{srt}} := Set \downarrow S^{srt}$. The objects $\Sigma \in \widetilde{S^{srt}}$ are sorted signatures $\Sigma : C \to S^{srt}$ so any constructor $f \in C$ comes with its sort $\Sigma(f) = (u, s) \in S^{*\times} \times S$. We define a cartesian operad tensor to get the monoidal category $S^{sig} := (\widetilde{S^{srt}}, \overset{co}{\otimes}, I_{op})$ of sorted signatures. So the free monoid object $\Sigma^{*\overset{co}{\otimes}}$ is the set containing all single terms which can be build from a sorted signature Σ. There is another monoidal category with the "left spread" tensor $S^{juxt} := (\widetilde{S^{*\times} \times S^{*\times}}, \overset{\triangleleft}{\otimes}, I_{\triangleleft})$. This monoidal category can be seen as the indexed version of the juxtaposition in dimension zero. Forgetting the monoidal structure, the category of sorted signatures S^{sig} is a full subcategory of S^{juxt}. After applying both free monoid constructions we get $\Sigma^{lw} := \Sigma^{*\overset{co}{\otimes} *\overset{\triangleleft}{\otimes}}$ the free Lawvere theory of a signature. Another characterisation of Σ^{lw} is the free cartesian category with set S as generator for objects and a signature Σ as generator for morphisms. This is the first dimension of higher dimensional syntax. In particular we get a first instance of the typing relation. Terms $t \in \Sigma^{lw}$ which are the syntactic items of dimension one will be typed by words $u, v \in S^{*\times}$, the syntactic items of the next lower dimension (concrete typing will be denoted as $u \vdash t : v$).

Dimension two

To understand the second dimension we will iterate the constructions applied so far. We use the syntax of dimension one to define a notion of parametric sorts Σ^{psrt}. We introduce constructors by the comma category construction $\widetilde{\Sigma^{psrt}} := Set \downarrow \Sigma^{psrt}$. The objects $\Sigma' \in \widetilde{\Sigma^{psrt}}$ of this category will be called parametric signatures.

Note at dimension zero we obtained the language by one free construction (juxtaposition). At dimension one we had to use two constructions (a indexed form of juxtaposition and the cartesian operad tensor). Finally at dimension two we have to use three constructions in order to get the language, namely juxtaposition, the cartesian operad tensor and additionally instantiation.

parametric signature	language of dimension $n-1$
$\Downarrow$ instantiation	$\Downarrow$
sorted signature	parametric sorts of dimension n
$\Downarrow$ composition closure	$\Downarrow$ introduction of constructors
collection of single terms	collection of parametric signatures
$\Downarrow$ juxtaposition	$\Downarrow$ choose a signature
language	
Steps to construct the free language of dimension n	*Steps to iterates the language construction one dimension higher*

Figure 1: Program : Higher dimensional languages

Higher dimensions

After dimension two the construction of higher dimensional syntax stabilises in the sense that we can iterate the constructions without a conceptual change. So to get syntax of dimension n we need a "signature" Σ of dimension n so that the collection of the free terms build over this signature is the language $\Sigma^{th(n)}$ of dimension n. In order to define a signature of dimension n we need a language $\Sigma'^{th(n-1)}$ of dimension $n-1$ to define the notion of parametric sorts Σ'^{psrt}. By a comma category construction $\widetilde{\Sigma'^{psrt}} := Set \downarrow \Sigma'^{psrt}$ we get as objects the the n-dimensional signatures. Conceptually the comma-construction does "introduction of constructors" because of the freedom to choose a set $C \in Set$ of constructors which should be sorted $\Sigma : C \to \Sigma'^{psrt}$. To get the n-dimensional syntactic language $\Sigma^{th(n)}$ we have to close the set under some syntactic operations.

The parametric sorts Σ'^{psrt} allows instantiations with $n-1$-dimensional syntactic items. The collection of all instances of sorts gives a new sort Σ'^{esrt}. We extend the instantiation to n-dimensional signatures to get the explicitly typed signature Σ^{exp} living in the monoidal category of sorted signatures $\Sigma'^{sig} := (\widetilde{\Sigma'^{esrt}}, \overset{co}{\otimes}, I_{op})$. At this level the construction of dimension one apply namely closing under cartesian operad and juxtaposition. We get a free sorted Lawvere theory $\Sigma^{exp\ lw}$ which is the explicitly typed version of the n dimensional language. A final step from explicitly typed system (synonymous: Church-style) to a implicitly typed system (synonymous: Curry-style) gives the free n-dimensional language $\Sigma^{th(n)}$ to a n-dimensional signature Σ.

Punchline

Despite the details of the above construction look rather involved, there is a simple punchline : free monoidal objects on varying monoidal categories does it all. In particular the variation of monoidal categories consists in defining the category of n-dimensional languages on top of the category of $n-1$-dimensional languages. The definition on top of another is done by a comma construction of "introducing constructors". By three closures namely juxtaposition, composition closure (using the cartesian operad) and instantiation the signature freely generates the language

2 Background

This section will present higher dimensional syntax by inductive definitions. The main aim of the paper is to characterise these notions by repeated construction of monoidal objects in varying monoidal categories. So the reader might skip this section and refer to it as he might feel the need for motivation of categorical notions. For those with a solid background in type theory sections 2.1 and 2.2 serves just as an introduction of notation. In section 2.3 the notion of higher dimensional syntax with a type inference algorithm is presented.

We will use pullbacks using the notation $A\,{}_f\!\times_g B$ in in case of given morphisms $f : A \to C$ and $g : B \to C$.

2.1 Fist dimension : signatures, Lawvere theories and unification

A signature is a tuple $\Sigma = (C, a : C \to \mathbb{N})$ consisting of a set C of constructors and an arity function a. Any signature Σ induces a free Lawvere theory Σ^{lw}. The elements $t \in \Sigma^{lw}(n,m)$ (another notation $n \vdash t : m$) are called terms. By convention the variables occuring in a term $n \vdash t$ are named as $\{x_1, \ldots, x_n\}$. Terms $t \in \Sigma^{lw}(n,m)$ are constructed by three cases :

(var)	$m = 1$	and	$t = x_i$	for $i \in \{1, \ldots, n\}$
(constr)	$m = 1$	and	$t = f(t')$	for $f \in C$ and $a(f) = k$ and $t' \in \Sigma^{lw}(n,k)$
(juxt)	$m = k + l$	and	$t = t^1 t^2$	for $t^1 \in \Sigma^{lw}(n,k)$ and $t^2 \in \Sigma^{lw}(n,l)$

So a term $t : m$ is a sequence of "single" terms $t = t_1 \ldots t_m$. We define the collection of single terms by setting $\Sigma^{stm}(n) := \Sigma^{th}(n,1)$. We define two functions length $|\_| : \Sigma^{lw} \to \mathbb{N}$ and available variables $av : \Sigma^{lw} \to \mathbb{N}$ by setting $|t| = m$ and $av(t) = n$ for any term of form $n \vdash t : m$. Composition of terms $t \bullet t'$ is defined if there is a natural number $n \in \mathbb{N}$ with $n \vdash t$ and $t' : n$; by substituting the variables $x_1, \ldots, x_n$ occuring in t by the single terms $t'_1, \ldots, t'_n$ which form t' by juxtaposition.

The instance relation on terms is defined by $t \sqsubseteq^{inst} t'$ iff $\exists t''.t \bullet t'' = t'$. The minimal witness of an instance relation is unique modulo renaming; we will use the notation $t'' = t'/t$ to denote it. Two terms t^1, t^2 are unifiable iff $\exists t.t^1 \bullet t = t^2 \bullet t$ where the term t is called unifier. The above term t is called the most general unifier (short: mgu) iff any other unifier t' for t^1, t^2 is an instance of t (t.i. $t \sqsubseteq^{inst} t'$).

Proposition 1 *(Existence of mgu) If the terms t^1 and t^2 are unifiable then there exists a most general unifier.*

Even better, unifiability is decidable and a most general unifier can be computed by the algorithm in figure 2. The algorithm hun takes as argument a triple (t^1, t^2, t) where terms t^1, t^2 are to be unified and t is a first guess of the unifying substitution. If the computation $hun(t^1, t^2, t)$ stops successfully with result (t', w) then w is an unifier and the resulting unified term is $t' = t^1 \bullet w = t^2 \bullet w$.

The properties of the unification algorithm are characterised by the following proposition.

Proposition 2 *(unification algorithm sound, minimal and complete)*

(sound) *If computation $hun(t^1, t^2, t)$ successfully results (t', w) then w is an unifier of the terms t^1, t^2 ...*

(minimal) *... any other other unifier t'' of t^1, t^2 satisfies the implication $t \sqsubseteq^{inst} t'' \implies w \sqsubseteq^{inst} t''$.*

$$
\begin{array}{lcl}
hun(x_i, t, w) & = & \text{if } w_i \neq x_i \text{ then return } hun(w_i, t, w) \\
& & \quad \text{else if } t = x_i \text{ then return } (x_i, w) \\
& & \quad \text{else if } x_i \text{ occurs in } t \text{ then } error(\textit{“occurrence failure”}) \\
& & \quad \text{else return } (t, (w|i \mapsto t)) \\
hun(t, x_i, w) & = & \text{return } hun(x_i, t, w) \\
hun(f(t^1), f(t^2), w) & = & \text{return } hun(t^1, t^2, w) \\
hun(t^1t^2, t^3t^4, w) & = & \text{such that } |t^1| = |t^3| \text{ and } |t^2| = |t^4| \\
& & \quad (t, w') := hun(t^1, t^3, w); \\
& & \quad (t', w'') := hun(t^2, t^4, w'); \\
& & \quad \text{return } ((t \bullet w'')t', w'') \\
\text{otherwise} & = & error(\textit{“not unifiable”}) \\
& & \quad - \text{only if length or top constructor of terms differ}
\end{array}
$$

where $(w|i \mapsto t)$ substitutes all occurrences of variable x_i in w by the single term t

$$
unify(t^1, t^2) \quad = \quad \text{for } n = |t^1| = |t^2| \ \text{ return } hun(t^1, t^2, x_1 \ldots x_n)
$$

Figure 2: unification algorithm

$$
sep\_unify : \Sigma^{lw} \times \Sigma^{lw} \rightharpoonup \Sigma^{lw} \times \Sigma^{lw}
$$

$$
\begin{array}{lcl}
sep\_unify(t^1, t^2) & = & n := av(t^1); \\
& & (\_, w) := unify(t^1, n \rhd t^2); \\
& & \text{return } w@n;
\end{array}
$$

Figure 3: separated unification algorithm

(complete) *If there is an unifier of term t^1, t^2 which is an instance of t the computation $hun(t^1, t^2, t)$ succeeds.*

Given two terms t^1, t^2 the solution of the *separated* unification problem is given by two witnesses t, t' satisfying $t^1 \bullet t = t^2 \bullet t'$. An algorithm solving this problem is presented in figure 3 taking t^1, t^2 as arguments and resulting with t, t' in case of success.

The following two functions are used to define the algorithm for separated unification. For $l \in \mathbb{N}$ variable shift $l \rhd \_ : \forall n, m. \Sigma^{lw}(n, m) \to \Sigma^{lw}(l + n, m)$ is defined by renaming any variable x_i occuring in t to x_{i+l} in $l \rhd t$. For any $l < m$ split $\_@l : \forall n, m. \Sigma^{lw}(n, m) \to \Sigma^{lw}(n, l) \times \Sigma^{lw}(n, m - l)$ is defined by splitting any term $t = t_1 \ldots t_m$ in a tuple of two pieces $t@l = (t_1 \ldots t_l, t_{l+1} \ldots t_m)$.

2.2 Second dimension : parametric signature, types, pre-values and associated type system

To a set S of sorts we get S^* the free monoid of strings and $S^{srt} := S^* \times S$ as the *sorts* to S. A S-sorted signature $\Sigma = (C, a : C \to S^{srt})$ comes with a sorted arity function.

To a signature $\Sigma\mathcal{T} = (E, at : E \to \mathbb{N})$ we get $\Sigma\mathcal{T}^{lw}$ the free Lawvere theory and $\Sigma\mathcal{T}^{psrt} := \sum_{n \in \mathbb{N}} \Sigma\mathcal{T}^{lw}(n, \_) \times at^{-1}(n)$ as the *parametric sorts* to $\Sigma\mathcal{T}$. A parametric signature $\Sigma\mathcal{D} = (D, ad : D \to \Sigma\mathcal{T}^{psrt})$ comes with a *typed* arity function.

The reader is encouraged to check that the algebraic data type definition in ML or Haskell are exactly parametric signatures. We call the elements $\tau \in \Sigma\mathcal{T}^{lw}$ the *types* of the parametric signature and the constructors $T \in E$ the type-constructors. The elements $C \in D$ are the data-constructors and $ad(C) = (\tau, T)$ is the sorting of C, in this case we define the *argument type* $ag_{\Sigma\mathcal{D}} : D \to \Sigma\mathcal{T}^{lw}$ of a data-constructor by setting $ag_{\Sigma\mathcal{D}}(C) = \tau$.

2.2.1 Explicit typing (Church style)

By definition of the parametric sorts $\Sigma\mathcal{T}^{psrt}$ we get the coincidence of arities $av(\tau) = at(T)$ for any sort element $(\tau, T) \in \Sigma\mathcal{T}^{psrt}$. We use this fact to define the parameter function $par : E \to \mathbb{N}$ by the map $C \mapsto av(ag_{\Sigma\mathcal{D}}(C))$. So we can build the explicit sorts $\Sigma\mathcal{T}^{esrt} := \{(\tau \bullet \tau', T\tau') \,|\, |\tau'| = par(C)\}$ by instantiation with composable types $\tau' \in \Sigma\mathcal{T}^{lw}$. The set of explicit sorts is a subset of the indexed set of non-parametric sorts $\Sigma\mathcal{T}^{esrt} \subseteq \Sigma\mathcal{T}^{stm\ srt} := \sum_{n \in \mathbb{N}} (\Sigma\mathcal{T}^{stm}(n))^* \times \Sigma\mathcal{T}^{stm}(n)$. So we define the explicit sorted signature which is an instance of a one dimensional $\Sigma\mathcal{T}^{stm}$-sorted signature $\Sigma\mathcal{D}^{exp} = (D\,{}_{part}\!\times_{|\_|} \Sigma\mathcal{T}^{lw}, ae)$ with sorting $ae : (D\,{}_{part}\!\times_{|\_|} \Sigma\mathcal{T}^{lw} \to \Sigma\mathcal{T}^{exp\ srt}$ defined by the map $(C, \tau') \mapsto (\tau \bullet \tau', T(\tau'))$ in case of sorting $ad(C) = (\tau, T)$. We can apply the construction of the free Lawvere theory on the explicit signature to get the $\Sigma\mathcal{T}^{stm}$-sorted theory $\Sigma\mathcal{D}^{exp\ lw}$. Note that the parameter function extends by structural induction to the explicitly typed theory so we get $par : \Sigma\mathcal{D}^{exp\ lw} \to \mathbb{N}$. By structural induction we define the right action of types on well sorted terms $\tau \vdash v : \tau'$ as the following map :

$$\_ \lhd \_ : \Sigma\mathcal{D}^{exp\ lw}\,{}_{par}\!\times_{|\_|} \Sigma\mathcal{T}^{lw} \to \Sigma\mathcal{D}^{exp\ lw}$$

$$v \lhd \tau'' = \begin{cases} \textbf{(var)} & x_i & \text{if } v = x_i \\ \textbf{(constr)} & (C, \tau^1 \bullet \tau'')(v' \lhd \tau'') & \text{if } v = (C, \tau^1)v' \\ \textbf{(juxt)} & (v^1 \lhd \tau'')(v^2 \lhd \tau'') & \text{if } v = v^1 v^2 \end{cases}$$

A simple check tells that $v \lhd \tau''$ is well-sorted iff $par(v) = av(\tau'')$ so we get a map of typings $(\tau \vdash v : \tau') \mapsto (\tau \bullet \tau'' \vdash v \lhd \tau'' : \tau' \bullet \tau'')$. Further the laws $v \lhd (\tau \bullet \tau') = (v \lhd \tau) \lhd \tau'$ and $(v \bullet v') \lhd \tau = (v \lhd \tau) \bullet (v' \lhd \tau)$ are valid.

2.2.2 Implicit typing (Curry style)

By mapping $C \mapsto |ag_{\Sigma\mathcal{D}}(C)|$ we get an arity function $ap : D \to \mathbb{N}$ and define the pre-value signature $\Sigma\mathcal{D}^{pre} = (D, ap : D \to \mathbb{N})$ of the parametric signature. The terms of the free theory $v \in \Sigma\mathcal{D}^{pre\ lw}$ are called the pre-values. In the following we will assume the convention that the variables of a pre-value $v \in \Sigma\mathcal{D}^{pre\ lw}(n, \_)$ are named like $x_1, \ldots, x_n$ and the *type variables* of a type $\tau \in \Sigma\mathcal{T}^{lw}(m, \_)$ are named like $a_1, \ldots, a_m$.

Any parametric signature induces a type system which extracts the well-typed *values* from the merely syntactic well-formed pre-values.

$$\text{(var)}\ \frac{}{\tau \vdash_{\Sigma\mathcal{D}} x_i : \tau_i} \qquad \text{(constr)}\ \frac{\tau \vdash_{\Sigma\mathcal{D}} v' : \tau' \quad \tau' = ag_{\Sigma\mathcal{D}}(C) \bullet \tau''}{\tau \vdash_{\Sigma\mathcal{D}} C(v') : T(\tau'')}$$

$$\text{(juxt)}\ \frac{\tau \vdash_{\Sigma\mathcal{D}} v^1 : \tau^1 \quad \tau \vdash_{\Sigma\mathcal{D}} v^2 : \tau^2}{\tau \vdash_{\Sigma\mathcal{D}} v^1 v^2 : \tau^1 \tau^2}$$

So we define the free implicitly typed theory generated by the parametric signature as the collection of well typed pre-values $\Sigma\mathcal{D}^{pth}(\tau, \tau') := \{v \in \Sigma\mathcal{D}^{pre\ lw} \,|\, \tau \vdash_{\Sigma\mathcal{D}} v : \tau' \text{ is derivable }\}$. A simple check confirms that the set $\Sigma\mathcal{D}^{pth}$ is closed under composition and juxtaposition.

As an alternative definition we can also extract the implicitly typed theory from the explicitly typed theory by erasure of type annotations. The erasure $\| \_ \|$: $\Sigma\mathcal{D}^{exp\ lw} \to \Sigma\mathcal{D}^{pre\ lw}$ is define by structural induction :

$$\| v \| = \begin{cases} \textbf{(var)} & x_i & \text{if } v = x_i \\ \textbf{(constr)} & C \| v' \| & \text{if } v = (C, \tau)v' \\ \textbf{(juxt)} & \| v^1 \| \| v^2 \| & \text{if } v = v^1 v^2 \end{cases}$$

The operation of erasure is to erase all type annotations τ from constructors (C, τ) occuring in a value. So the second definition is $\Sigma\mathcal{D}^{pth}(\tau, \tau') := \{\| v \| \ | v \in \Sigma\mathcal{D}^{exp\ lw}(\tau, \tau')\}$. So the right action $\lhd$ carries over to the implicit theory by setting $v \lhd \tau := \| v \lhd \tau \|$. This fact is known in type theory as the following proposition.

Proposition 3 *(substitution property) Given a derivation $\tau \vdash v : \tau'$ then for any composable type τ'' the judgement $\tau \bullet \tau'' \vdash v : \tau' \bullet \tau''$ is derivable.*

2.2.3 Type inference

Type inference computes a typing to a given pre-value. In figure 4 we present an algorithm for type inference to a given parametric signature $\Sigma\mathcal{D}$. Core of that algorithm is a function $hatp_{\Sigma\mathcal{D}} : \Sigma\mathcal{T}^{lw} \times \Sigma\mathcal{D}^{pre\ lw} \rightharpoonup \Sigma\mathcal{T}^{lw} \times \Sigma\mathcal{T}^{lw}$ which takes a tuple (τ, v) where v is the pre-value to be typed and τ is a first guess of the type of the variables occuring in v. The result $(\tau', \tau'') = hatp_{\Sigma\mathcal{D}}(\tau, v)$ forms a typing $\tau' \vdash_{\Sigma\mathcal{D}} v : \tau''$ where τ' is an instance of the guess τ. The function $hatp_{\Sigma\mathcal{D}}$ considers three cases corresponding to the rules (var), $(constr)$ and $(juxt)$ of the type system. The constructor case makes use of separated unification.

The properties of the type inference algorithm are stated by the following proposition.

Proposition 4 *(type inference sound, minimal type and complete)*

(sound) *If computation $hatp_{\Sigma\mathcal{D}}(\tau, v)$ succeeds with result $(\tau'\tau'')$ then type judgement $\tau' \vdash_{\Sigma\mathcal{D}} v : \tau''$ is derivable ...*

(minimal type) *... and for any derivation $\tau^1 \vdash_{\Sigma\mathcal{D}} v : \tau^2$ holds the implication $\tau \sqsubseteq^{inst} \tau^1 \implies \tau'' \sqsubseteq^{inst} \tau^2$.*

(complete) *If pre-value v can be typed so that the variables of v are typed by an instance of τ then computation $hatp_{\Sigma\mathcal{D}}(\tau, v)$ succeeds.*

The idea to prove the above proposition for the type inference algorithm goes as follows. The type inference algorithm makes use of the unification algorithm. The proof of soundness, minimality and completeness therefore makes use of the corresponding properties of the unification algorithm. Soundness is proved by constructing a derivation from a successful computation (again the three cases are to consider). By juggling with instance-inequations the soundness-proof is refined to prove minimality. The proof of completeness goes the other way round; take a derivation and construct a successful computation from it (again the three cases are to consider).

2.2.4 Minimal typ*ing*

Above statement can be strengthened to the minimal typ*ing* property. This property states that any derivation $\tau^1 \vdash v : \tau^2$ is an instance of the minimal typing $\tau \vdash v : \tau'$ by a substitution. To formalise this it is necessary to consider only those term variables which occur in the value. For a value $\tau \vdash v$ we define the following set of indices $occ(v) := \{i \in [|\tau|] \,|\, x_i \text{ occurs in } v\}$. Let $n = |\tau|$ and $O \subset [n]$ and

$$
\begin{array}{lcl}
hatp : \Sigma\mathcal{T}^{lw} \times \Sigma\mathcal{D}^{pre\ lw} \rightharpoonup \Sigma\mathcal{T}^{lw} \times \Sigma\mathcal{T}^{lw} & & \\
hatp(\tau, x_1) & = & \mathsf{return}\ (\tau, \tau_i); \\
hatp(\tau, Cv) & = & (\tau', \tau^v) := hatp(\tau, v); \\
& & (T, \tau^{da}) := srt_{\Sigma\mathcal{D}}(C); \\
& & (\tau^{vw}, \tau^{dw}) := sep\_unify(\tau^v, \tau^{da}); \\
& & \mathsf{return}\ (\tau' \bullet \tau^{vw}, T\tau^{dw}); \\
hatp(\tau, v^1 v^2) & = & (\tau', \tau^1) := hatp(\tau, v^1); \\
& & (\tau'', \tau^2) := hatp(\tau', v^2); \\
& & \mathsf{return}\ (\tau'', (\tau^1 \bullet (\tau''/\tau'))\tau^2);
\end{array}
$$

Figure 4: type inference algorithm algorithm

$m = n - |O|$ with the unique monotone bijection $f : [n] \to [m]$ then we define the restricted type by its single term components as :

$$\text{for } i \in [n] \ \ (\tau\rceil_O)_i := \begin{cases} \tau_i & \text{if } i \in O \\ x_{n+f(i)} & \text{if } i \in [n] \setminus O \end{cases}$$

By a simple property of derivations we see that the minimal type property implies the minimal typing property in $\vdash_{\Sigma\mathcal{D}}$.

Proposition 5 *(transitivity of typing instances) Given a pre-value v and two derivations $\tau \vdash v : \tau'$ and $\tau^1 \vdash v : \tau^2$ then a witness τ^w for the typing instance is already given by a witness for the context or the result type. Formally*

$$(\forall i \in occ(v).\tau_i \bullet \tau^w = \tau_i^1) \ \Leftrightarrow\ \tau' \bullet \tau^w = \tau^2$$

The proof is given by induction on the derivation considering the three cases (var), (constr) and (juxt).

Corollary 1 *(minimal typing) Transitivity of typing instance means that the minimal type property implies the stronger minimal typing property. Assume given a derivation $\tau \vdash v : \tau'$ so that type τ' is minimal. This implies for any other derivation $\tau^1 \vdash v : \tau^2$ that the result type is an instance $\tau' \sqsubseteq^{inst} \tau^2$. Therefor exists an instance witness τ^w such that $\tau' \bullet \tau^w = \tau^2$. This implies by transitivity $\forall i \in occ(v).\tau_i \bullet \tau^w = \tau^1$. Therefore by simple restriction we get a minimal typing*

$$\boxed{\tau\rceil_{occ(v)} \vdash v : \tau'}$$

Therefore type inference gives an inverse to the right action $\lhd$ by computing a minimal typing $\tau \vdash v : \tau'$ to a given pre-value. We get a function $tp : \Sigma\mathcal{D}^{th} \to \Sigma\mathcal{D}^{th}{}_{par} \times_{av} \Sigma\mathcal{T}^{lw}$ by the map $\tau^1 \vdash v : \tau^2 \mapsto (\tau \vdash v : \tau', \tau^w = \tau^2/\tau' = \tau^1/\tau)$. So we get the two valid implication to function applications of tp :

result $tp(\tau^1 \vdash v : \tau^2) = (\tau \vdash v : \tau', \tau^w)$ implies
(witness) $(\tau \vdash v : \tau') \lhd \tau^w = \tau^1 \vdash v : \tau^2$
(right action) $tp((\tau^1 \vdash v : \tau^2) \lhd \tau'') = (\tau \vdash v : \tau', \tau^w \bullet \tau'')$

So we have an effective function $mtw_{\Sigma\mathcal{D}} : \Sigma\mathcal{T}^{lw} \times \Sigma\mathcal{D}^{pre\ lw} \times \Sigma\mathcal{T}^{lw} \rightharpoonup \Sigma\mathcal{T}^{lw}$ computing the instance witness τ^w to a minimal typing if input pre-typing $\tau^1 \vdash$

$v : \tau^2$ is derivable (t.i. $mtw_{\Sigma\mathcal{T}}(\tau^1, v, \tau^2) = \tau^w$ if $\tau \vdash v : \tau'$ minimal tying and $(\tau \vdash v : \tau') \lhd \tau^w = (\tau^1 \vdash v : \tau^2)$). So we are lead to define the carrier of the implicitly typed terms by

$$C_2 := \{(v, \tau^w) \mid \exists \tau^1, \tau^2 \in \Sigma\mathcal{T}^{lw}.mtw_{\Sigma\mathcal{T}}(\tau^1, v, \tau^2) = \tau^w\} \subseteq \Sigma\mathcal{D}^{pre\ lw} \times \Sigma\mathcal{T}^{lw}$$

Note that this two dimensional object is defined purely by use of one dimensional free Lawvere theories and an effective function.

2.3 Higher dimensional language: The inductive definition

A language of dimension n comes with a carrier C_n of terms. Each term is typed by a terms of the lower dimensional language so to term $t \in C_n$ exist terms $t^1, t^2 \in C_{n-1}$ with a valid typing relation $t^1 \vdash_n t : t^2$. Therefor we have two functions (codomain and domain) $c, d : C_n \to C_{n-1}$. Let's assume in the following that carriers C_n with a negative index are singletons $C_n = \{\bullet\}$ iff $n < 0$. We deal with freely generated languages so there is a signature G_n which generates the language. Our goal is to understand the free construction $\_^{th(n)}$ which leads to the definitional equation $C_n := G_n^{th(n)}$ telling how a signature generates the language.

The free construction consists of three steps namely : indexed juxtaposition, composition closure and instantiation. A n-dimensional generator is a parametric signature G_n. This datum is the departure point of a our journey to the n-dimensional language. Instantiation takes the parametric signature G_n and produces a sorted signature eG_n. The composition closure takes the sorted signature eG_n and produces the collection of single terms sC_n. Juxtaposition of the single terms finally produces the language C_n.

Note that each construction induces an operation which is present in the language C_n. Instantiation induces the right action $\lhd$. Composition closure induces composition $\bullet$ and juxtaposition induces pairing.

2.3.1 Indexed sets

To understand the operations of juxtaposition, composition closure and instantiation in detail, we have to examine the indexed structure of the set sets G_n, eG_n and sC_n. A parametric signature comes with parametric sorting $psrt : G_n \to pS_n$. The sorted signature comes with a simple sorting $srt : eG_n \to S_n$. Likewise the single term collection comes with simple sorting $tsrt : sC_n \to S_n$. We can define the notion of a hom-sorts by a pullback $H_n := C_{n-1\,c}\times_c C_{n-1}$. This allows the viewpoint that each language comes with a hom-sorting $cd : C_n \to H_n$.

To understand the nature of the parametric sorts pS_n and simple sorts S_n we look a the hom-sorts H_n which have similar structure. Because hom-sorts are defined by a pullback we can split the hom-sorting cd into its two familiar parts $c, d : C_n \to C_{n-1}$ of codomain and domain function. Further the pullback implies the validity of the left globularity condition $c; c = d; c$. So we get another indexing to a n-dimensional language which we call the parameters by setting $par := c; c = d; c : C_n \to C_{n-2}$. Given a n-dimensional typing $t^1 \vdash_n t : t^2$ the parameters $par(t)$ tell type of the variables occuring in the terms t^1 and t^2. Finally by giving the pullback definition for the sorts pS_n and S_n we understand how to split the sortings into two parts c and d and definition of a parameter function par for the sets G_n, eG_n and sC_n. We define the sorts by setting :

$$\begin{array}{lcl} pS_n & := & C_{n-1\,c}\times_c G_{n-1} \\ S_n & := & C_{n-1\,c}\times_c sC_{n-1} \end{array}$$

In the following we will denote terms by $t \in C_n$ single terms by $s \in sC_n$ and constructors by $f \in G_n$ or $f \in eG_n$. By use of the sortings we define the following indexed sets :

$$\begin{array}{lcll} \text{set of typed terms} & C_n(t,t') & := & \{t'' \in C_n \mid cd(t'') = (t,t')\} \\ \text{set of typed single terms} & sC_n(t,s) & := & \{t' \in sC_n \mid tsrt(t') = (t,s)\} \\ \text{set of sorted constructors} & eG_n(t,s) & := & \{f \in eG_n \mid srt(f) = (t,s)\} \\ \text{set of parametric constructors} & G_n(t,f) & := & \{f' \in G_n \mid psrt(f') = (t,f)\} \end{array}$$

Note that all indexes t, t', s and f used in the definitions above are elements of dimension $n-1$.

We are now prepared to give the inductive definitions for instantiation, composition closure and juxtaposition.

2.3.2 Instantiation

The construction of instantiation consists in transforming a parametric signature $(G_n, psrt)$ into a sorted signature (eG_n, srt). The definition of parametric sorts by the pullback $pS_n := C_{n-1\,c}\times_c G_{n-1}$ gives a canonical split of the parametric sorting function $psrt$ to $c : G_n \to C_{n-1}$ and $d : G_n \to G_{n-1}$. Further by the pullback property we define the parameter function $par := c; c = d; c : G_n \to C_{n-2}$. We construct the set of sorted constructors by the following pullback $eG_n := G_{n\ par}\times_d C_{n-1}$. The fine structure of any simple constructor $f \in eG_n$ is therefor a tuple (f', t) consisting of a parametric constructor $f' \in G_n$ and a term $t \in C_{n-1}$. To complete the description of instantiation we define the sorting $srt : eG_n \to S_n$ by the following map :

$$(f,t) \mapsto (c(f) \overset{n-1}{\bullet} t, d(f)(t))$$

In order to define the result tuple of the map we made use of $n-1$ dimensional structure. The left component of the tuple uses the composition $\overset{n-1}{\bullet}$ defined in the language C_{n-1}. The right component of the tuple uses the fact that the set of single terms sC_{n-1} is compositional closed.

2.3.3 Composition closure

The construction of composition closure consists in transforming a a sorted signature (eG_n, srt) into a collection of single terms $(sC_n, tsrt)$ The definition of simple sorts by the pullback $S_n := C_{n-1\,c}\times_c sC_{n-1}$ gives a canonical split of the simple sorting function srt to $c : eG_n \to C_{n-1}$ and $d : eG_n \to sC_{n-1}$. By a two clause inductive definition we define the notion of a n-dimensional single term $s \in sC_n(t, s')$:

$$\begin{array}{lll} \textbf{(var)} & s = x_i^n & \text{for } i \in [|t|] \\ \textbf{(comp)} & s = f(t') & \text{for } f \in eG_n(t'', s') \text{ and } t' \in C_n(t, t'') \end{array}$$

Note that in a term of dimension n may occur variables x_i^n of dimension n. Note further by freeness any language comes with a notion length of a term $|\_| : C_n \to \mathbb{N}$. Finally we chose to present the definition of single term collection with use of the full language C_n in case **(comp)** .

2.3.4 Indexed juxtaposition

The construction of indexed juxtaposition consists in transforming a a collection of single terms $(sC_n, tsrt)$ into the language (C_n, cd) of dimension n. To define juxtaposition of dimension n we make use of juxtaposition of dimension $n-1$. Suppose given the single terms $s_1, \ldots, s_m \in sC_{n-1}$ so that the juxtaposition $s_1 \ldots s_m \in C_{n-1}$ is defined in dimension $n-1$. We define for any term $t \in C_{n-1}$ juxtaposable with term $s_1 \ldots s_m$ the indexed set :

$$C_n(t, s_1 \ldots s_m) := sC(t, s_1) \times \cdots \times sC_n(t, s_m)$$

$$
\begin{array}{lcl}
tinf : \Sigma_n^{lw} \rightharpoonup C_1 \times \cdots \times C_{n-1} \; \times \; C_{n-1} \times \cdots \times C_1 & & \\[1em]
tinf(v) & = & (t^c, t^d) := tinf_n(t); \\
 & & (t^{cc}, t^{cd} t^{dd}) := tinf_{n-1}(t^c t^d); \\
 & & \quad \vdots \\
 & & (t^{c^{n-1}}, t^{c^{n-2}d} t^{c^{n-3}dd} \ldots t^{d^{n-1}}) := tinf_2(t^{c^{n-2}} t^{c^{n-3}d} \ldots t^{d^{n-2}}); \\
 & & \mathsf{return}\ (t^{c^{n-1}}, \ldots, t^c, t^d, \ldots, t^{d^{n-1}});
\end{array}
$$

Figure 5: n dimensional type inference algorithm algorithm

Note further that by freeness *all* terms of C_{n-1} are represented by an unique juxtaposition of single terms.

Having said this we have achieved the definition of the n dimensional language.

2.3.5 Type inference

The parametric signature $(G_n, psrt : G_n \to pS_n)$ can be reduced to a one dimensional pre-value signature $\Sigma_n := (G_n, a : G_n \to \mathbb{N})$ by defining the arity as $a := psrt; c; |\_|$ with use of the length function $|\_| : C_{n-1} \to \mathbb{N}$. So the elements $v \in \Sigma_n^{lw}$ are the prevalues. Collapsing the signature G_{n-1} to the simple signature Σ_{n-1} implies that we reduce the n dimensional signature G_n to a 2 dimensional parametric signature $\Sigma\mathcal{D}_n := (G_n, rpsrt : G_n \to \Sigma_{n-1}^{psrt})$. In section 2.2.3 we presented a type inference algorithm computing a minimal typing to any value in the two dimensional case. So we get an effective function

$$tinf_n : \Sigma_n^{lw} \rightharpoonup \Sigma_{n-1}^{lw} \times \Sigma_{n-1}^{lw}$$

which computes the minimal typing for any prevalue $v \in \Sigma_n^{lw}$ typable in the type system $\vdash_{\Sigma\mathcal{D}_n}$. Suppose we have a successful computation $tinf_n(v) = (\tau, \tau')$ so we have a derivation $\tau \vdash_{\Sigma\mathcal{D}_n} v : \tau'$. But we want to know if v is an element of the n dimensional language C_n. That holds iff the prevalue $\tau\tau' \in \Sigma_{n-1}^{lw}$ is a value of C_{n-1}. So we have reduced the n dimensional typing problem to a $n-1$ dimensional typing problem. So by a sequence of $n-1$ type inference algorithms for $1 < i \leq n$:

$$tinf_i : \Sigma_i^{lw} \rightharpoonup \Sigma_{i-1}^{lw} \times \Sigma_{i-1}^{lw}$$
for the two dimensional parametric signature $\Sigma\mathcal{D}_i := (G_i, rpsrt : G_i \to \Sigma_{i-1}^{psrt})$

We get a type inference algorithm for dimension n computing a minimal typing see figure 5.

The minimal typing property gives to any derivation $d : t^1; \ldots; t^{n-1} \vdash_{C_n} t : t'^{n-1} : \cdots : t'^1$ with $t^i, t'^i \in C_i$ a unique witness $q^w := t^{w1}, \ldots, t^{w(n-1)}$ such that the minimal typing m represents by right action through the witness the given derivation :

$m \lhd q^w = d$ explicitly:
$d = t^{c^{n-1}} \overset{1}{\bullet} t^{w1}; \ldots; t^c \overset{n-1}{\bullet} t^{w(n-1)} \vdash_{C_n} t : t^d \overset{n-1}{\bullet} t^{w(n-1)} : \cdots : t^{d^{n-1}} \overset{1}{\bullet} t^{w1}$

So we get a representation of the n dimensional carrier expressed in one dimensional terms and a effective function :

$$C_n := \{(t, t^{w(n-1)}, \ldots, t^{w1}) \Sigma_n^{lw} \times \cdots \times \Sigma_1^{lw} \mid q^w \text{ is the witness for a typing of } t\}$$

3 Categorical Machinery

3.1 Free monoid objects

The words to a given set of letters S form the free monoid S^* to that set. Set theoretically we construct $S^* := \sum_{n\in\mathbb{N}} S^n$ involving a geometric series. Categorically this means we define a monoid object in the monoidal category $(Set, \times, 1)$ taking the cartesian product as tensor. The categorical extension of this construction is folklore. For instance Dubuc [Dub74] mentions that the construction of a free monoid object is possible in any monoidal category $(\mathcal{C}, \otimes, I)$ if the category enumerable cocomplete and the tensor preserves colimits in both arguments. In the following we will always assume the category in question is enumerable cocomplete. The constructions will make use of cartesian product and pullbacks so we will assume finite completeness. We will present a sequence of comma-categories with monoidal structure. Note that the comma construction preserves finitary completeness and enumerable cocompleteness.

3.2 Notations

We will use the following notations. To a object A of a monoidal category $(\mathcal{C}, \otimes, I)$ we will denote the free monoidal object as $A^{*\otimes}$. The multiplication morphism will be $m : A^{*\otimes} \otimes A^{*\otimes} \to A^{*\otimes}$ and the unit morphism will be $u : I \to A^{*\otimes}$. The operation of constructing the free monoid object is a monad on $\mathcal{C}$ given by a functor $\_^{*\otimes} : \mathcal{C} \to \mathcal{C}$ and two natural transformations $\eta : Id \dot{\to} \_^{*\otimes}$ and $\mu : \_^{*\otimes} \circ \_^{*\otimes} \dot{\to} \_^{*\otimes}$.

We denote the projections to the cartesian product as $\pi^1 : A \times B \to A$ and $\pi^2 : A \times B \to B$. For any two morphisms $f : A \to B$ and $g : A \to C$ we denote the pairing by $< f, g >: A \to B \times C$.

For any two morphisms $f : A \to C$ and $g : B \to C$ we will denote the pullback object by $A\,{}_f{\times}_g\,B$ and the left and right projection by $p^1 : A\,{}_f{\times}_g\,B \to A$ and $p^2 : A\,{}_f{\times}_g\,B \to B$. The diagonal arrow will be denoted by $f \overset{i}{\times} g := p^1; f = p^2; g : A\,{}_f{\times}_g\,B \to C$.

Given an object of a category $A \in \mathcal{C}$ we will denote the comma category by $\widetilde{A} := \mathcal{C} \downarrow A$. The objects of the comma category $f \in \widetilde{A}$ are morphisms $f : C \to A$ of the category $\mathcal{C}$. To any morphism $g : A \to B$ we get two functors between comma categories $\_ \blacktriangleleft g : \widetilde{A} \to \widetilde{B}$ by composition and $g \blacktriangleright \_ : \widetilde{B} \to \widetilde{A}$ by pullback. We will call the operation $\_ \blacktriangleleft g$ the right change of base and $g \blacktriangleright \_$ the left change of base.

3.3 Monoidal comma categories

3.3.1 Cartesian Product

Given two objects $f, g \in \widetilde{A}$ of a comma category we have morphisms $f : C \to A$ and $g : C' \to A$ of category $\mathcal{C}$. By finitary completeness of $\mathcal{C}$ we get the pullback $f \overset{i}{\times} g : C\,{}_f{\times}_g\,C' \to A$. So we get the monoidal structure $A^\times := (\widetilde{A}, \overset{i}{\times}, I_i)$ by setting $I_i := id_A \in \mathcal{C}(A, A)$. In fact this is a cartesian product on the comma category and the indexed pairing for morphisms $h : f \to g$ and $h' : f' \to g$ is defined by the morphism $< h, h' >^i :=< p^1, p^2 >; < h, h' >: f \overset{i}{\times} f' \to g$. The projections of the pullback p^i are the projections π^i of the cartesian product in the comma category.

3.3.2 Arrow product

Given a comma category objects $f \in \widetilde{A \times B}$ we introduce two notations $f_c := f \blacktriangleleft \pi^1 \in \widetilde{A}$ and $f_d := f \blacktriangleleft \pi^2 \in \widetilde{B}$. Given two objects $f \in \widetilde{A \times B}$ and $g \in \widetilde{C \times D}$ we will consider one of the following coincidences $A = C, A = D, B = C$ or $B = D$.

Lets take the case $A = D$ so we have $f \in \widetilde{A \times B}$ and $g \in \widetilde{C \times A}$ in this case we can define $f\,{}_c{\times}_d\, g := < f_c \overset{i}{\times} g_d, p^1; f_d, p^2; g_c >: p(f_c \overset{i}{\times} g_d) = p(f)\,{}_{f_c}{\times}_{g_d}\, p(g) \to A \times B \times C$. So we get a bifunctor $\_\,{}_c{\times}_d\,\_ : \widetilde{A \times B} \times \widetilde{C \times A} \to \widetilde{A \times B \times C}$. The definitions of ${}_c{\times}_c$, ${}_d{\times}_d$ and ${}_d{\times}_c$ should be obvious.

By defining the bifunctor $\_ \overset{\circ}{\times} \_ := \_\,{}_c{\times}_d; \_ \blacktriangleleft < \pi^3, \pi^2 > : \widetilde{A \times B} \times \widetilde{C \times A} \to \widetilde{C \times B}$ we get a monoidal category $A^\circ := (\widetilde{A \times A}, \overset{\circ}{\times}, I_\circ)$ by setting $I_\circ := \delta_A =< id_A, id_A > \in \mathcal{C}(A, A \times A)$. In case of the underlying category $\mathcal{C} = Set$ we see that any object $G \in \widetilde{V \times V}$ is a graph. The set V is the set of vertices and $G : C \to V \times V$ is a colouring of edges with colours $c \in C$. The tensor product $G \overset{\circ}{\times} G'$ can be seen as composition of edges with juxtaposition of colours. The free monoid object $G^{*\overset{\circ}{\times}}$ is the free category generated by the graph G.

3.3.3 Sprout product

We denote by A^* the free monoid object in the monoidal category $(\mathcal{C}, \times, 1)$. So we get the structure maps $m : A^* \times A^* \to A^*$ and $u : 1 \to A^*$.The construction of the free monoid object is a monad so we get the endofunctor $\_^* : \mathcal{C} \to \mathcal{C}$ with natural transformations $\eta : Id \dot{\to} \_^*$ and $\mu : \_^* \circ \_^* \dot{\to} \_^*$. The monad functor $\_^*$ gives a functor on comma categories for any objects $A \in \mathcal{C}$ we get a functor $\widetilde{A} \to \widetilde{A^*}$ by the map $f : C \to A \mapsto f^* : C^* \to A^*$.

We define the bifunctor "left sprout" $\_ \overset{\triangleright}{\times} \_ := \_\,{}_d{\times}_d\,\_; \_ \blacktriangleleft << \pi^2, \pi^3 >; m, \pi^1 > : \widetilde{A^* \times B} \times \widetilde{A^* \times B} \to \widetilde{A^* \times B}$. The definition of the "right sprout" bifunctor $\overset{\triangleleft}{\times}$ should be obvious. So we get a monoidal category $(\widetilde{A^* \times B}, \_ \overset{\triangleright}{\times} \_, I_\triangleright)$ by setting $I_\triangleright :=< \eta_A, id_B > \in \mathcal{C}(1 \times B, A^* \times B)$

4 Two dimensional languages: The categorical axiomatisation

4.1 Sorted signatures and operad tensors

4.1.1 Juxtaposing constructors

We call the category $A^{sig} := \widetilde{A^* \times A}$ the category of A-sorted signatures. The category $A^{cgrph} := \widetilde{A^* \times A^*}$ is called the category of A-homsorted cartesian graphs. Note the following morphism $asrt := (\pi^{1\,*}; \mu_a) \times \pi^{2\,*} : (A^* \times A)^* \to A^* \times A^*$ which "adjusts the sorting". We get a functor $jc : A^{sig} \to A^{cgrph}$ by the following equivalent definitions $\_^*; \_ \blacktriangleleft asrt = \_ \blacktriangleleft < id_{A^*}, \eta_A >; \_^{*\overset{\triangleleft}{\times}}$.

4.1.2 Operad tensors

We define the planar sorted operad tensor by $\_ \overset{po}{\otimes} \_ := (Id_{A^{sig}} \times jc); \_ \overset{\circ}{\times} \_ : A^{sig} \times A^{sig} \to A^{sig}$. So $(A^{sig}, \overset{po}{\otimes}, I_{op})$ is a monoidal category by setting the operad unit to $I_{op} = I_\triangleright =< \eta_A, id_A > \in \mathcal{C}(1 \times A, A^* \times A)$.

Any morphism in $FinSet$ is a function $f : [n] \to [m]$ for natural number intervals $[n] := \{1, \ldots, n\}$. To any objects $A \in \mathcal{C}$ we get a morphism $A^f :< \pi^{f(1)}, \ldots, \pi^{f(n)} >: A^m \to A^n$. By definition of the free monoid object $A^* := \sum_{n \in \mathbb{N}} A^n$ we get for any natural number $n \in \mathbb{N}$ a mono $in_n : A^n \to A^*$. So we get a comma category object $A^f; in_n \in \widetilde{A^*}$. Given two objects $g \in \widetilde{A^* \times B}$ and $h \in \widetilde{A^*}$ we get a new object by use of indexed product $g_c \overset{i}{\times} h \in \widetilde{A^*}$ so we define

$g\,_c{\times}\,h := < g_c \overset{i}{\times} h, p^1; g_d > \in \widetilde{A^* \times B}$. Putting things together we get a definition of "combing" on top of an operad (see Baez [BD98]). For object $g \in \widetilde{A^* \times B}$ we define :

$$FunCmb\, g := \sum_{m \in \mathbb{N}} \sum_{n \in \mathbb{N}} \sum_{f \in FinSet(n,m)} g\,_c{\times}(A^f; in_n) \in \widetilde{A^* \times B}$$

Note that we used copairing $[f, g] : C + C' \to A$ in category $\mathcal{C}$ to get a indexed coproduct $f + g \in \widetilde{A}$ in the comma category. Further we used cocompleteness of $\mathcal{C}$ for the existence of the colimit. This definition can be used for any subcategory of $FinSet$. If we take the subcategory of finite sets and bijections $FinBij$ we get the functor $BijCmb : \widetilde{A^* \times B} \to \widetilde{A^* \times B}$.

Finally we define the classical sorted operad tensor by :

$$\_ \overset{so}{\otimes} \_ : BijCmb \times BijCmb; \_ \overset{po}{\otimes} \_ : A^{sig} \times A^{sig} \to A^{sig}$$

And we define the cartesian sorted operad tensor by :

$$\_ \overset{co}{\otimes} \_ : FunCmb \times FunCmb; \_ \overset{po}{\otimes} \_ : A^{sig} \times A^{sig} \to A^{sig}$$

4.2 Free Lawvere theory

Applying the free monoid construction twice we get the construction of the free Lawvere theory to a given signature. $\_^{lw} := \_^{*\overset{co}{\otimes}}; \_ \blacktriangleleft < id_{A^*}, \eta_A >; \_^{*\overset{\triangleleft}{\times}} : A^{sig} \to \widetilde{A^* \times A^*}$. The first free monoid construction is closure relative to composing of constructors. The second free monoid construction is closure relative to juxtaposition.

So to any signature $\Sigma \in A^{sig}$ we get the free Lawvere theory $\Sigma^{lw} \in \widetilde{A^* \times A^*}$. Note that Σ^{lw} has double monoid objects structure $\Sigma^{lw} \in Mon(\widetilde{A^* \times A^*}, \overset{\circ}{\times}, I_\circ)$ (i) and by definition $\Sigma^{lw} \in Mon(\widetilde{A^* \times A^*}, \overset{\triangleleft}{\times}, I_\triangleleft)$ (ii). So $p(\Sigma^{lw})$ is a internal cartesian category object in $\mathcal{C}$. By setting $C_1 := p(\Sigma^{lw})$ and $C_0 := A^*$ we get morphisms for codomain and domain $c := (\Sigma^{lw})_c : C_1 \to C_0$ and $d := (\Sigma^{lw})_d : C_1 \to C_0$ By (i) we get a composition morphism $comp : C_1\, {}_c{\times}_d\, C_1 \to C_1$ as $comp := p(m_\circ)$ for monoid multiplication $m_\circ : \Sigma^{lw} \overset{\circ}{\times} \Sigma^{lw} \to \Sigma^{lw}$. The cartesian product $prod : C_0 \times C_0 \to C_0$ is given by $prod := m_A$ for monoid multiplication $m_A : A^* \times A^* \to A^*$. And pairing $pair : C_1\, {}_d{\times}_d\, C_1 \to C_1$ is given as $pair := p(m_\triangleleft)$ for multiplication $m_\triangleleft : \Sigma^{lw} \overset{\triangleleft}{\times} \Sigma^{lw} \to \Sigma^{lw}$ by (ii). The commuting diagrams are easily checked.

5 Higher dimensional languages : The categorical axiomatisation

The construction of higher dimensional syntax follows a simple principle. Each dimension comes with its notion of language $\mathcal{L}_n$, signature $\mathcal{S}_n$ and free construction $\_^{th(n)} : \mathcal{S}_n \to \mathcal{L}_n$. The notion of n-dimensional language is defined by its internal structure and by a right action of a $n-1$-dimensional language on it. In order to define a n-dimensional language L_n the underlying hierarchy of languages $L_0, \ldots, L_{n-1}$ up to dimension $n-1$ have to be constructed first. The free construction $\_^{th(n)}$ is composed of two steps. First the signature of dimension n is reduced to a signature of dimension $n-1$ by the process of "instantiation" $inst_n : \mathcal{S}_n \to \mathcal{S}_{n-1}$. This instantiation process generates by structural induction the the right action of a $n-1$-dimensional language on the language to be constructed. In the second step the free construction $\_^{th(n-1)}$ of dimension $n-1$ is applied to the instantiated signature.

Lets collect the machinery we have so far. To a monoidal category $(\mathcal{B}, \otimes, I)$ with a free monoid construction $\_^{*\otimes}$ we define two functors :

signature sorts	$\_^{srt\,\otimes}$	$: \mathcal{B} \to \mathcal{B}$	by the map $A \mapsto A^{*\otimes} \otimes A$
hom sorts	$\_^{homs\,\otimes}$	$: \mathcal{B} \to \mathcal{B}$	by the map $A \mapsto A^{*\otimes} \otimes A^{*\otimes}$

Above we constructed to each objects $A \in \mathcal{B}$ three monoidal categories :

signature category	$A^{sig} := (\widetilde{A^{srt\,\otimes}}, \overset{co}{\otimes}, I_{co})$
juxtaposition category	$A^{juxt} := (\widetilde{A^{homs\,\otimes}}, \overset{\triangleleft}{\times}, I_{\triangleleft})$
composition category	$A^{comp} := (\widetilde{A^{homs\,\otimes}}, \overset{\circ}{\times}, I_{\circ})$

Forgetting the monoidal structure we see that A^{juxt} and A^{comp} are the same categories and A^{sig} is a full subcategory of them via right change of base $\_^{s2h} := \_ \blacktriangleleft < id_{A^{*\otimes}}, \eta_A > : \widetilde{A^{srt\,\otimes}} \to \widetilde{A^{homs\,\otimes}}$. We define the category of Lawvere theories A^{lwth} which are monoid objects L in the both categories $L \in A^{juxt}$ and $L \in A^{comp}$. In the section above we defined a functor $\_^{lw} : A^{sig} \to A^{lwth}$.

5.1 Iteration

To any comma category $\widetilde{A} := \mathcal{C} \downarrow A$ we get a functor $p : \widetilde{A} \to \mathcal{C}$ by taking the codomain part of the comma category structure. Functor p will be used to iterate the comma construction. To a comma category object $f \in \widetilde{A}$ we define $\widetilde{f} := \widetilde{p(f)} = \mathcal{C} \downarrow p(f)$.

The goal of the following section is to extend then functor $\_^{lw}$ by iteration. Let's define the single term object $\Sigma^{stm} := \Sigma^{*\overset{co}{\otimes}\,s2h}$ to a signature $\Sigma \in A^{sig}$. Note that the single term object Σ^{stm} is an object of the monoidal category A^{juxt}. So by setting the $\mathcal{B} := A^{juxt}$ and $A := \Sigma^{stm}$ we can iterate the above constructions.

We define the parametric sort object $\Sigma^{psrt} := \Sigma^{lw}\ {}_c{\times}_c\ \Sigma^{s2h}$ in category $A^{homs\,\otimes}$. Of course the category of parametric signatures is defined by "introduction of constructors" $\Sigma^{psig} := \widetilde{\Sigma^{psrt}}$. By use of the unit of the free cartesian operad monoid $u : \Sigma \to \Sigma^{*\overset{co}{\otimes}}$ therefor $u^{s2h} : \Sigma^{s2h} \to \Sigma^{*\overset{co}{\otimes}\,s2h} = \Sigma^{stm}$ we get a morphism

$$inp : \Sigma^{psrt} \to (\Sigma^{stm})^{srt\overset{\triangleleft}{\times}} = (\Sigma^{stm})^{*\overset{\triangleleft}{\times}} \overset{\triangleleft}{\times} \Sigma^{stm} = \Sigma^{lw}\ {}_c{\times}_c\ \Sigma^{stm}$$

So we get a witness $\_ \blacktriangleleft inp : \Sigma^{psig} \to (\Sigma^{stm})^{sig}$ for the fact that the category of parametric signatures is a full subcategory of the iterated signature category $(\Sigma^{stm})^{sig}$.

Below we will define in categorical terms instantiation as a functor $\_ \star \Sigma^{lw} : \Sigma^{stm\ sig} \to \Sigma^{stm\ sig}$ So we can apply the free Lawvere theory construction to achieve a free parametric theory construction $\_^{pth} := \_ \blacktriangleleft inp; \_ \star \Sigma^{lw}; \_^{lw} : \Sigma^{psig} \to \Sigma^{stm\ lw}$

5.2 Instantiation

Instantiation is the crucial concept to understand dimension two and higher. The idea of instantiation is to fill given parameters with all composable arguments. Applying this to a collections of parametrised items produces the collection of all instances. The generators of a language of of dimension two and higher are parametric signatures. So a generator is a collection of parametrised constructors. The instantiation of a parametrised signature produces a sorted signature to which the free Lawvere theory construction applies. So we reduce a higher dimension generator to a one dimensional generator and apply the one dimensional construction.

5.2.1 Categorical instantiation

Given a monoid object A in the monoidal category $B^\circ := (\widetilde{B \times B}, \overset{\circ}{\times}, I_\circ)$. We want to define the instantiation $f \star A \in \widetilde{A}$ of an object $f \in \widetilde{A}$ in the iterated comma category. We will assume that the objects A and f are morphisms $A : C \to B \times B$ and $f : C' \to C$ in category $\mathcal{C}$. Further we will make use of the components $A_c, A_d \in \widetilde{B}$. The following pullback defines the codomain $p(f \star A)$ of the instantiation.

$$\begin{tikzcd}
& & C \arrow[dr, "A_c"] & \\
C'{}_{f;A_d}\times_{A_c} C \arrow[urr, "p^2"] \arrow[dr, "p^1"'] & & pb & B \\
& C' \arrow[r, "f"] & C \arrow[ur, "A_d"'] &
\end{tikzcd}$$

Recall the definition of the multiplication morphism $m_\circ : A \overset{\circ}{\times} A \to A$ in the monoidal category B° by the following diagram :

$$\begin{tikzcd}
& & & B \\
& & C \arrow[ur, "A_d"] \arrow[dr, "A_c"] & \\
C \arrow[urrr, bend left, "A_d"] \arrow[drrr, bend right, "A_c"'] & p(A \overset{\circ}{\times} A) \arrow[l, "p(m_\circ)"'] \arrow[ur] \arrow[dr] & pb & B \\
& & C \arrow[ur, "A_d"] \arrow[dr, "A_c"] & \\
& & & B
\end{tikzcd}$$

So the span build by the two legs $p^1; f : p(f \star A) \to C$ and $p^2 : p(f \star A) \to C$ induces the unique mediating arrow to the above pullback $k : p(f \star A) \to p(A \overset{\circ}{\times} A)$. So we use the morphism $k; p(m_\circ) : p(f \star A) \to C$ as definition of the instantiated object $f \star A \in \widetilde{A}$ in the iterated comma category. Note further by a second use of multiplication $m_\circ$ we get the right action $ract : (f \star A) \star A \to f \star A$ as morphism in category $\widetilde{A}$. By use of associativity of the monoid object A we should get an iso $(f \star A) \star A \cong f \star (A \overset{\circ}{\times} A)$.

5.3 Structure of a language

Each language of dimension n is a quintuple $L_n = (C_n, L_{n-1}, <\_,\_>^n, \overset{n}{\bullet}, \overset{n}{\lhd})$ where C_n is the carrier of the language and L_{n-1} is a language of dimension $n-1$. There are codomain and domain morphisms to the carrier of the lower dimension $c_n, d_n : C_n \to C_{n-1}$. These morphisms satisfy the left globularity condition $c_n; c_{n-1} = d_n; c_{n-1}$ we call the resulting morphism the parameter morphism $par : C_n \to C_{n-2}$. The three operations are :

$$\begin{array}{lll}
\textbf{(pairing)} & <_,_>^n & : C_n\, {}_c\times_c\, C_n \to C_n \\
\textbf{(composition)} & _\overset{n}{\bullet}_ & : C_n\, {}_c\times_d\, C_n \to C_n \\
\textbf{(right action)} & _\overset{n}{\lhd}_ & : C_n\, {}_{par}\times_d\, C_{n-1} \to C_n
\end{array}$$

C_n	language object
G_n	parametric signature object
pS_n	parametric sort object (to define $G_n \in \widetilde{pS_n}$)
eS_n	simple sort object (with mono $inj : eS_n \rightarrowtail sC_{n-1}^{srt\overset{\triangleleft}{\times}}$)
eG_n	sorted signature object (with property $eG_n \in \widetilde{eS_n}$)
sC_n	"single term object"

... their definitions and where they live.

$$
\begin{array}{lcll}
C_n & := & eG_n^{lw} = sC_n^{*\overset{\triangleleft}{\times}} & \in sC_{n-1}^{lwth} \\
sC_n & := & eG_n^{*\overset{co}{\otimes}} & \text{monoid object in } sC_{n-1}^{sig} \\
pS_n & := & C_{n-1} \overset{\triangleleft}{\times} G_{n-1} & \in sC_{n-2}^{cgrph} \\
eS_n & := & pS_n \overset{\circ}{\times} C_{n-1} & \in sC_{n-2}^{cgrph} \\
eG_n & := & G_n \star C_{n-1} & \in sC_{n-1}^{sig}
\end{array}
$$

Figure 6: objects of dimension n in higher dimensional syntax

(composition)	monoid object in monoidal category $(sC_{n-1})^{comp}$
(juxtaposition)	monoid object in monoidal category $(sC_{n-1})^{juxt}$
(right action)	A right action map $ract : C_n \star C_{n-1} \to C_n$ in category $(sC_{n-1})^{lwth}$

Figure 7: structure of a n-dimensional language object C_n

Satisfying the following laws (in case of sets C_n, C_{n-1} with elements $t, t' \in C_n$ and $u, v \in C_{n-1}$) :

$$
\begin{array}{ccccccc}
d < t, t' >^n & = & < dt, dt' >^{n-1} & & & & \\
d(t \overset{n}{\bullet} t') & = & dt' & & c(t \overset{n}{\bullet} t') & = & ct \\
c(t \overset{n}{\triangleleft} u) & = & c(t) \overset{n-1}{\bullet} u & & d(t \overset{n}{\triangleleft} u) & = & d(t) \overset{n-1}{\bullet} u \\
 & & & & & & \\
< t, t' > \bullet t'' & = & < t \bullet t'', t' \bullet t'' > & & < t, t' > \triangleleft u & = & < t \triangleleft u, t' \triangleleft u > \\
(t \bullet t') \triangleleft u & = & (t \triangleleft u) \bullet (t' \triangleleft u) & & t \overset{n}{\triangleleft} (u \overset{n-1}{\bullet} v) & = & (t \overset{n}{\triangleleft} u) \overset{n}{\triangleleft} v
\end{array}
$$

The reader may feel free to present the above laws as commuting diagrams. By this definition we get internal language objects in the category Set. Any category $\mathcal{C}$ which is enumerable cocomplete and finitely complete would suffice too. Note that the comma category construction preserves these completeness properties.

To lay a foundation for this sequence of languages notice the trivial language 1 given by setting $C_n = C_{n-1} = C_{n-2} = 1$. Therefore the codomain, domain and parameter morphisms are trivial and the pairing, composition and right action too. We define that languages L_n with negative index n are trivial.

6 Conclusion

This paper presents the formal foundation of higher dimensional syntax. The contribution consists in two characterisations. One by inductive definitions and one categorical axiomatisation, bringing together the two sources of inspiration namely type theory and higher dimensional categories. The first definition is inspired by type theoretical notions which culminates in a minimal typing property with a type inference algorithm for an effective computation of this existence property. The categorical definition is inspired by the new emerging notions of higher dimensional category theory culminating the following one sentence characterisation. **A higher dimensional language is a left globular object which is a cartesian category object at each level and has right actions of each lower dimensional language**. Further the free construction of those languages from a parametric signature is presented.

Future work

The higher dimensional syntax comes equipped with a type inference algorithm, pattern matching, map and fold algorithms [Weh98]. This suggests that higher dimensional syntax can be considered as programming language with a very rich type structure. An implementation with experimental results should evaluate the usefulness of this language.

Actual research of the author is concerned with deriving logic from syntax. At the time being a sequence of constructions have been discovered starting with syntax, constructing a logic, constructing proof nets to this logic, constructing a geometry of interaction (short: GoI) on the proof nets, finally constructing game semantics from the GoI. This constructions are reversible leading to full completeness results. Details should appear soon in printed form. At the moment this analysis of full completeness works at dimension two. An extension of these results to higher dimensions might lead to full completeness results for variants of Barendregts λ-cube and pure type systems.

Related work

The work is inspired by the ongoing research discovering higher dimensional categorical structures (termed "post-modern algebra" by John Baez). The notion of Lawvere theories and its inductive definition is present in Wagners work [Wag94]. The notion of "combing" a operad tensor is inspired by Baez, Dolan [BD95, BD98] and Schmitt [Sch93]. The notion of left globularity is inspired by the Batanin, Street globular approach [Bat98, Str98]. The work on the multicategory approach by Hermida, Makkai, Power [HMP98, Her97] and Leinster [Lei98a, Lei98b, Lei99] bears a striking resemblance. One might be tempted to call the constructions in this paper iterated free cartesian multicategory constructions from cartesian multigraphs. Further investigation to pin down this similarity would be rewarding.

On the type theory side the Barendregts notion of the λ-cube and pure type systems [Bar92] are related. But these type theories run into undecidability of typing [Wel94]. Opposed to that all higher dimensional languages are predicative higher dimensional polymorphic type theories with decidable typing problem. A first type inference algorithm has been given by Damas, Milner [DM82] To the knowledge of the author the higher dimensional type inference algorithm is new.

Acknowledgement

I want to thank Samson Abramksy for his patience and encouraging discussions. I want to thank Daniele Turi who helped to improve the presentation.

References

[Bar92] H. P. Barendregt. *Handbook of Logic in Computer Science*, volume 2, chapter Lambda Calculi with Types, pages 118–310. Oxford Science Publications, May 1992.

[Bat98] M.A. Batanin. Monoidal globular categories as a natural environment for the theory of weak n-categories. *Adv. Math.*, 136:39–103, 1998.

[BD95] J. Baez and J. Dolan. Higher-dimensional algebra and topological quantum field theory. *Jour. Math. Phys.*, 36:6073–6105, 1995.

[BD98] J. Baez and J. Dolan. Higher dimensional algebra III: n-categories and the algebra of opetopes. *Adv. Math.*, 135:145–206, 1998.

[DM82] L. Damas and R. Milner. Principal type-schemes for functional programs. In *POPL' 82*, pages 207–211. ACM, 1982.

[Dub74] Eduartdo J. Dubuc. Free monoids. *Journal of Algebra*, 29:208–228, 1974.

[FPT99] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *LICS*, 1999.

[GP99] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax involving binders. In *14th LICS*, Trento, Italy, July 1999.

[Her97] Claudio Hermida. Spans and higher dimesnional multicategories. lecture notes of talk at CT97 and AMS meeting, 1997.

[HMP98] C. Hermida, M. Makkai, and J. Power. Higher dimensional multigraphs. In *LICS '98 Proceedings*, volume 13, pages 207–213, Indianapolis, Indiana, June 1998. IEEE, ieee press.

[Hof99] Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *LICS*, 1999.

[Lei98a] Tom Leinster. General operads and multicategories. xxx preprint archive as math.CT/9810053, 1998.

[Lei98b] Tom Leinster. Structures in higher-dimesnsional category theory. available at `http://www.dpmms.cam.ac.uk/ leinster/`, 1998.

[Lei99] Tom Leinster. Generalised enrichement for categories and multicategories. xxx preprint server as math.CT/9901139, 1999.

[Sch93] Wiliam R. Schmitt. Hopf algebras of combinatorical structures. *Canad. J. Math.*, 45(2):412–428, 1993.

[Str98] Ross Street. The role of Michael Batanin's monoidal globular categories. In E. Getzler and M. Kapranov, editors, *Higher Category Theory*, number 230 in Contemporary Mathematics, pages 99–116. A.M.S., 1998.

[Wag94] Eric G. Wagner. *Handbook of Logic in Computer Science*, volume 3, chapter Algebraic Semantics, pages 323–393. 1994.

[Weh98] Martin Wehr. Higher order algebraic data types (full paper). Technical report, University of Edinburgh, `URL http://www.dcs.ed.ac.uk/home/wehr`, July 1998.

[Wel94] J. B. Wells. Typability and type checking in the second-order λ-calculus are equivalent and undecidable. In *Proc. 9th Ann. IEEE Symp. Logic in Computer Science*, 1994.