DIAGRAMMES

YVES LAFONT

Primitive recursive categories and machines

Diagrammes, tome 22 (1989), p. 7-13

<http://www.numdam.org/item?id=DIA_1989__22__7_0>

© Université Paris 7, UER math., 1989, tous droits réservés.

L'accès aux archives de la revue « Diagrammes » implique l'accord avec les conditions générales d'utilisation (http://www.numdam.org/legal.php). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

NUMDAM

Article numérisé dans le cadre du programme
Numérisation de documents anciens mathématiques
http://www.numdam.org/

DIAGRAMMES VOLUME 22 , 1989

ACTES DES JOURNEES E.L.I.T.

(UNIV. PARIS 7, 27 JUIN-2 JUILLET 1988)

Primitive Recursive Categories and Machines

Yves Lafont
CNRS
Laboratoire d'Informatique de l'Ecole Normale Supérieure,
45 rue d'Ulm,
F 75230 Paris Cedex 05

Résumé

Nous étudions les relations entre des résultats catégoriques [Burroni] et des machines abstraites pour les algorithmes récursifs primitifs, dans la lignée de la *Machine Catégorique Abstraite* [CouCurMau] et de la *Machine Linéaire Abstraite* [Lafont88,Lafont88a].

Abstract

We investigate connections between categorical results of [Burroni], and abstract machines for primitive recursive algorithms, in the lineage of the *Categorical Abstract Machine* [CouCurMau] and the *Linear Abstract Machine* [Lafont88a].

reçu le 21/05/1989

1 Primitive Recursive Functions

The *primitive recursive functions* are traditionally defined as the members of the smallest class of functions $\mathrm{N}^n \to \mathrm{N}$ ($n \in \mathrm{N}$) containing some basic ones, and closed under substitution and primitive recursion:

- The projections $\Pi^1, \ldots, \Pi^n : \mathrm{N}^n \to \mathrm{N}$ are primitive recursive.
- If $f_1, \ldots, f_p : \mathrm{N}^n \to \mathrm{N}$ and $g : \mathrm{N}^p \to \mathrm{N}$ are primitive recursive, then the function $h : \mathrm{N}^n \to \mathrm{N}$ such that
$$h(u) = g(f_1(u), \ldots, f_p(u))$$
is also primitive recursive. Notation: $h = g \circ \langle f_1, \ldots, f_p \rangle$ (*substitution*).
- The constant zero $0 : \mathrm{N}^0 \to \mathrm{N}$ and the successor function $\mathsf{S} : \mathrm{N} \to \mathrm{N}$ are primitive recursive.
- If $f : \mathrm{N}^n \to \mathrm{N}$ et $g : \mathrm{N}^{2+n} \to \mathrm{N}$ are primitive recursive, then the function $h : \mathrm{N}^{1+n} \to \mathrm{N}$ such that
$$h(0, u) = f(u) \qquad h(\mathsf{S}x, u) = g(x, h(x, u), u)$$
is also primitive recursive. Notation: $h = \mathsf{Rec}(f, g)$ (*primitive recursion*).

This class is very big indeed: the reader can check that all usual arithmetical functions are primitive recursive, for example:

$$\begin{array}{rclrcl} \mathsf{plus}(0, y) & = & y & \mathsf{plus}(\mathsf{S}x, y) & = & \mathsf{S}(\mathsf{plus}(x, y)) \\ \mathsf{times}(0, y) & = & 0 & \mathsf{times}(\mathsf{S}x, y) & = & \mathsf{plus}(\mathsf{times}(x, y), y) \\ \mathsf{fact}(0) & = & \mathsf{S}0 & \mathsf{fact}(\mathsf{S}x) & = & \mathsf{times}(\mathsf{fact}(x), \mathsf{S}x) \end{array}$$

Lemma 1 *(encoding tuples) For all $n \geq 0$, there are primitive recursive functions $\gamma : \mathrm{N}^n \to \mathrm{N}$ and $\pi^1, \ldots, \pi^n : \mathrm{N} \to \mathrm{N}$ such that $\pi^i \circ \gamma = \Pi^i$ for $i = 1, \ldots, n$.*

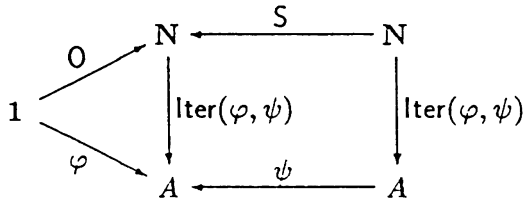
Define $\gamma(x_1, x_2, x_3, \ldots) = 2^{x_1} 3^{x_2} 5^{x_3} \ldots$ which is obviously primitive recursive. Proving that the corresponding $\pi^1, \ldots, \pi^n : \mathrm{N} \to \mathrm{N}$ are also primitive recursive is a bit harder (see [Kleene]).

Primitive recursive functions are traditionally introduced to define the *general recursive functions*, *i.e.* the ones that are computable, at least in theory. But we shall not consider them from this viewpoint.

2 Primitive Recursive Categories

In the category of sets, and more generally in a topos, the *object of natural numbers* N is characterised by the axiom of Lawvere:

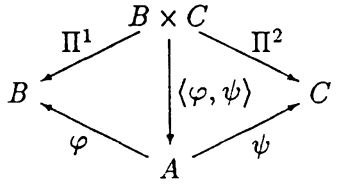
$1 \xrightarrow{0} \mathrm{N} \xleftarrow{S} \mathrm{N}$ is initial among the diagrams $1 \xrightarrow{\varphi} A \xleftarrow{\psi} A$ (where 1 is the terminal object). In other words, for any such diagram, there is a unique $\mathsf{Iter}(\varphi, \psi) : \mathrm{N} \to A$ such that the following diagram commutes:



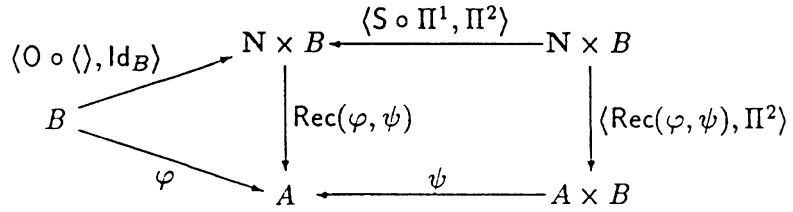
However, this notion, which is a special case of Kan extension, is to weak to capture primitive recursion in categories which are not Cartesian closed, and since primitive recursion has nothing to do with functionality (*i.e.* existence of exponentials), we need a stronger axiom:

A *primitive recursive category* is a category with *finite products* and a *parametrised object of natural numbers* $1 \xrightarrow{0} \mathrm{N} \xleftarrow{S} \mathrm{N}$.

So for each object A, there is a unique $\langle\rangle : X \to 1$, and for each diagram $B \xleftarrow{\varphi} A \xrightarrow{\psi} C$ there is a unique $\langle \varphi, \psi \rangle : A \to B \times C$ such that the following diagram commutes:



We also require that for any diagram $B \xrightarrow{\varphi} A \xleftarrow{\psi} A \times B$, there is a unique $\mathsf{Rec}(\varphi, \psi) : \mathrm{N} \times B \to A$ such that the following diagram commutes:



In particular $B = 1$ gives the axiom of Lawvere.

The structure of primitive recursive category with explicit choice of products and object of natural numbers is obviously (projectively) sketchable (although not algebraic over graphs, see [CopLair]). Consequently, there is a *free* primitive recursive

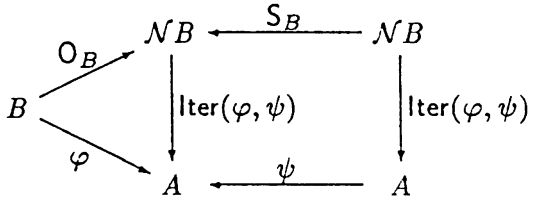
category $\mathcal{P}$, and a canonical functor $\mathcal{F} : \mathcal{P} \to \mathcal{S}et$, since $\mathcal{S}et$ is clearly a primitive recursive category. Using lemma 1, it is quite easy to see that the $\mathcal{F}(\varphi)$ for $\varphi : \mathrm{N}^n \to \mathrm{N}$ in $\mathcal{P}$ are precisely the primitive recursive functions.

Note that, by standard diagonalisation arguments, $\mathcal{F}$ is neither full (for example, Akermannn's function is not primitive recursive) nor faithful[1] (some primitive recursive algorithms are extensionally equal although distinct in $\mathcal{P}$).

3 Getting rid of Cartesian products

Although primitive recursive functions are explicitly defined in terms of projections and substitution, there are good reasons (see [Burroni]) to interpret this notion in categories that are not necessarily Cartesian.

> A *Burroni category*[2] is just a category with an *object of natural numbers* $B \overset{0_B}{\to} \mathcal{N}B \overset{S_B}{\leftarrow} \mathcal{N}B$ over each object B: for any diagram $B \overset{\varphi}{\to} A \overset{\psi}{\leftarrow} A$, there is a unique $\mathrm{Iter}(\varphi, \psi) : \mathcal{N}B \to A$ such that the following diagram commutes:



A primitive recursive category is clearly a Burroni category with $\mathcal{N}B = \mathrm{N} \times B$. The converse is false, since nothing forces a Burroni category to be Cartesian (consider for example a discrete category with two points), and even if it is, $\mathcal{N}A$ may have nothing to do with $\mathcal{N}1 \times A$ (as it is the case in the dual category of $\mathcal{S}et$). However, we have the quite surprising fact:

Theorem 1 *[Burroni] The free Burroni category $\mathcal{B}[X]$ over one object X admits finite products.*

Since the objects of $\mathcal{B}[X]$ are of the form $\mathcal{N}^n X$ for $n \in \mathrm{N}$, we guess that the terminal object is $\mathcal{N}^0 X = X$ and $\mathcal{N}^p X \times \mathcal{N}^q X = \mathcal{N}^{p+q} X$. But of course, the main point of the proof is to construct all required morphisms.

Corollary 1 *The category $\mathcal{B}[X]$ is equivalent to $\mathcal{P}$.*

[1] There is an elegant proof of this in [Burroni], using Matijasevič's Theorem.

[2] This is called a *Peano-Lawvere category* in [Burroni].

There are indeed two canonical functors $\Phi : \mathcal{B}[X] \to \mathcal{P}$ such that $\Phi X = \mathbf{1}$ and $\Psi : \mathcal{P} \to \mathcal{B}[X]$. It is easy to see that $\Psi \circ \Phi$ is the identity functor on $\mathcal{B}[X]$, and $\Psi \circ \Phi$ is naturally isomorphic to the identity on $\mathcal{P}$.

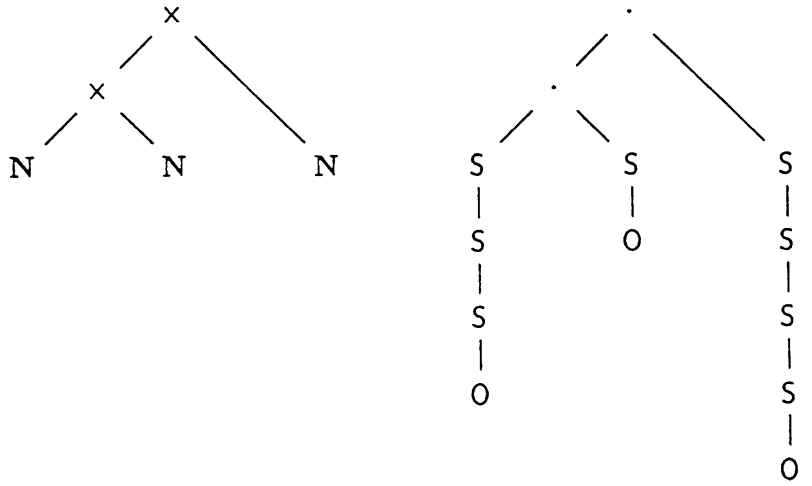
Corollary 2 *If $\mathcal{G}$ is the canonical functor $\mathcal{B}[X] \to \mathcal{S}et$ such that $\mathcal{G}X = \mathbf{1}$, the $\mathcal{G}(\varphi)$ for $\varphi : \mathcal{N}^p X \to \mathcal{N}X$ in $\mathcal{B}[X]$ are precisely the primitive recursive functions.*

These results express that $\mathcal{B}[X]$ is the category of *formal primitive recursive functions.*

4 From categories to machines

Following Lambek's terminology, morphisms in the free categories $\mathcal{P}$ and $\mathcal{B}[X]$ are syntacticly described as equivalence classes of *categorical combinators.* These combinators are truly programs written in a very low level language for a machine we shall describe precisely, at least in the case of $\mathcal{B}[X]$.

One major difference between $\mathcal{P}$ and $\mathcal{B}[X]$ is the structure of types (or objects). In $\mathcal{P}$, types are *trees* and data should be represented similarly:



In $\mathcal{B}[X]$, the corresponding type would be $\mathcal{N}\mathcal{N}\mathcal{N}X$ and the datum SSSOSOSSSSO, which can be implemented as a *stack* on a computer, whereas trees would require *pointers*, with usual problems of *memory allocation* and *garbage collection.*

Theorem 2 *[Burroni] Every morphism $X \to \mathcal{N}X$ in $\mathcal{B}[X]$ is of the form $\mathsf{S}^n \circ \mathsf{O}$ for some $n \in \mathbf{N}$. More generally, every morphism $X \to \mathcal{N}^p X$ is of the form*

$$\underbrace{\mathsf{S} \circ \ldots \circ \mathsf{S}}_{n_1 \text{ times}} \circ \mathsf{O} \circ \ldots \circ \underbrace{\mathsf{S} \circ \ldots \circ \mathsf{S}}_{n_p \text{ times}} \circ \mathsf{O} \qquad \textit{(standard tuple)}$$

for some $(n_1, \ldots, n_p)$ in $\mathbf{N}^p$.

This result can be proved elegantly by applying the universal property of $\mathcal{B}[X]$ to the *gluing* of $\mathcal{B}[X]$ and $\mathcal{S}et$ along the functor $A \mapsto \mathsf{Hom}(X, A)$. Alternatively, it is enough to check:

Lemma 2 *If $\varphi : \mathcal{N}^p X \to \mathcal{N}^q X$ is a morphism in $\mathcal{B}[X]$, then for every standard p-tuple σ, $\varphi \circ \sigma$ is a standard q-tuple.*

This is proved by easy induction on (the length of) a combinator representing φ.

The proof of this lemma can be interpreted as an *operational semantics* for primitive recursive algorithms. Assume indeed that a standard tuple is implemented as a stack:

S
⋮
S
O
⋮
S
⋮
S
O

Combinators are executed as follows:

- Id : do nothing,
- $\varphi \circ \psi$: execute ψ, then φ,
- O : push O,
- S : push S,
- Iter(φ, ψ) : pop S until you reach O, pop O, execute φ, and execute ψ as many times as you popped S.

On usual computers, the code is a sequence of instructions followed by Return. Hence Id will be compiled into an empty sequence, $\varphi \circ \psi$ into a concatenation of two sequences, and O, S into primitive instructions. For Iter(φ, ψ), we need a loop instruction and a supplementary stack to store code addresses:

The Iterative Abstract Machine					
Before			After		
code	tuple stack	code stack	code	tuple stack	code stack
O; c	T	C	c	O/T	C
S; c	T	C	c	S/T	C
Iter(c', c''); c	T	C	Loop(c', c'')	T	c/C
Loop(c', c'')	S/T	C	Loop(c', c'')	T	c''/C
Loop(c', c'')	O/T	C	c'	T	C
Return	T	c/C	c	T	C

Lemma 2 can be reformulated as follows:

Theorem 3 *If the machine starts with the code of a combinator* $\varphi : \mathcal{N}^p X \to \mathcal{N}^q X$, *a p-tuple* σ, *and an empty code stack, it eventually stops on the instruction* Return *with a q-tuple* τ *and an empty code stack.* τ *is the result of applying the algorithm* φ *to* σ.

5 Conclusion

Our iterative machine is certainly not relevant for computer scientists since unary integers are notoriously inefficient. However, all this material extends straightforwardly to finite Kan extensions, such as $B \to \mathcal{W}B \leftleftarrows \mathcal{W}B$ (with *two* successor functions instead of one) corresponding to binary notation.

It is also interesting to consider the dual of $\mathcal{B}[X]$, and more generally, finite right Kan extensions, because the corresponding machines look quite different (lemma 2 is asymmetrical indeed): data are not *tuples* but *automata.* Perhaps we shall discuss this point in another paper. .

Anyhow this illustrates a new interpretation of category theory which has already been initiated in [CouCurMau] and extended in [Lafont88].

References

[Burroni] A. Burroni, Récursivité graphique (1^e partie): catégorie des fonctions récursives primitives formelles, *Cahiers de topologie et géométrie différentielle catégorique* **XXVII-1** (1986).

[CopLair] L. Coppey & C. Lair, Algébricité, monadicité, esquissabilité et non-algébricité, *Diagramme* **13** (1985) 1-112.

[CouCurMau] G. Cousineau, P.L. Curien & M. Mauny, The Categorical Abstract Machine, J. P. Jouannaud, ed., *Functional Programming Languages and Computer Architecture*, LNCS **201** (Springer-Verlag, 1985) 50-64.

[Goodstein] R.L. Goodstein, *Constructive Formalism*, Essays on the foundations of mathematics (University College, Leicester, 1951).

[Kleene] S.C. Kleene, *Introduction to Meta-mathematics* (North Holland, 1952).

[Lafont88] Y. Lafont, Logiques, Catégories et Machines, Thèse de doctorat (Université Paris VII, 1988).

[Lafont88a] Y. Lafont, The Linear Abstract Machine, *Theoretical Computer Science* **59** (1988) 157-180.

[Lambek] J. Lambek, Deductive systems and categories, *Math. Systems Theory* (1968).