The Thirty-Second International Florida
Artificial Intelligence Research Society Conference (FLAIRS-32)

Flexible Approach for Computer-Assisted Reading and Analysis of Texts

Ismaïl Biskri, Mohamed Hassani
Université du Québec à Trois-Rivières
{ismail.biskri ; mohamed.hassani}@uqtr.ca

Abstract

A Computer-Assisted Reading and Analysis of Texts (CARAT) process is a complex task that should be, always, under the control of the user according to his subjectivity, his knowledge, his interests, etc. It is, then, important to design flexible platforms to support the implementation of CARAT tools, their management, their adaptation to new needs and the experiments. Even, in the last years, several platforms for digging textual data have emerged, they lack flexibility and sound formal foundations. We propose, in this paper, a formal model with strong logical foundations, based on typed applicative systems.

Introduction

Features extraction, data normalization, classifiers, interpretation tools, etc. have an impact on the result of any complex process in the domain of the Computer Assisted Reading and Analysing of Texts (CARAT). Different combinations of features, data normalization classifiers and interpretation tools are possible and should be explored in order to improve or customize CARAT processes.

There is a real need for ongoing interaction between users and CARAT, due to the dynamic nature of texts, their environment and the different objectives of their reading and analysis. These processes, being subject to multiple adjustments, must demonstrate sufficient flexibility. The achievement of relevant results is highly dependent on the ability of the CARAT process to be readily adapted to the intended objectives of analysis, being previously planned or not. CARAT processes flexibility can be seen as the ability to deal with foreseen and unforeseen cases by adapting parts of a processing chain. In other words, flexibility is as much about what should stay the same in a processing chain as what should be allowed to change (Shonenberg et al. 2008). Let us recall that in technical terms, a processing chain is a specific combination of computational operations; each output of one computational operation can be the input of one or many other computational operations. In what follows, computational operations will be called modules. To ensure flexibility, it is important that:

- The processing chain design should allow various execution alternatives that may arise within the processing chain model. The user can select the most appropriate execution path, by selecting one or more modules from a set of available modules or by changing the order in which modules should run. That is what we call the discovery process.
- When the processing chain is a part of a collaborative work, or known to need to be adjusted at a later stage, its design could be underspecified.
- During the processing chain execution, a new understanding of the goal of the reading and analysis can arise. It may require adding a new module to the processing chain, removing a module from the processing chain, or simply replacing one module with another.
- A specific sequence of modules can be used in several processing chains. It can be considered as a complex module composed of some basic modules.

In the literature about CARAT many projects aim to allow the creation of complex processing chains. ALADIN (Seffah and Meunier 1995), D2K/T2K (Downie et al. 2005), RapidMiner (Mierswa et al. 2006), Knime (Warr 2007) and WEKA (Witten, Frank and Hall 2011) use *processing chains* for language and data engineering, Gate (Cunningham et al. 2002) use it for linguistic analysis and Pipeline-Pilot in the field of industry (Dassault-Systems / BIOVA). The processing chains are widely used, but the solutions previously mentioned suffer from several limitations shown in (Biskri et al. 2015). With the constraint of flexibility, the main question is: How to ensure a systematic syntactic validation of the processing chains? We propose a formal model based on typed applicative systems, in which the validation of the construction of a processing chain is performed by a logical calculation on types in the general framework of Typed Applicative Systems (Biskri and Desclés 1997 ; Shaumyan 1998).

Typed Applicative Systems and Combinatory Logic

Typed Applicative Systems (TAS) postulate a general model in which a construction operation applies an operator to an operand to give a result. The applicative expressions are structured by the simple juxtaposition of two arguments, an operator followed by its operand. If X is an operator and Y its operand then X Y represents the application of X to Y. The set of applicative expressions is recursively constructed by the following rules:

- Basic operators and basic operands are applicative expressions
- If X and Y are applicative expressions then X Y is an applicative expression.

To prove that an application expression is well-formed, TAS assign to each operator and to each operand an applicative type to express how it works. The set of applicative types is recursively defined as follows:

- Basic types are types.
- If α and β are types, $F\alpha\beta$ is a type.

$F\alpha\beta$ is the type of an operator whose operand is of type α and the result of its application to its operand is of type β. One operator X with a type $F\alpha\beta$ is noted $[X : F\alpha\beta]$.

In the scientific literature, there are several typed applicative systems, including Church's lambda-calculus and combinatory logic. Most functional languages like LISP use lambda-calculus as a basis for their modeling. Although used in some functional languages like Haskell, combinatory logic (Curry and Feys 1958 ; Hindley and Seldin 2008) is, particularly, used in approaches of syntactic, semantic and even cognitive analysis of natural languages (Desclés, Guibert and Sauzay 2016). From an extensional point of view, these two systems are considered equivalent. Nevertheless, they are not from an intentional point of view. Unlike lambda-calculus, combinatory logic does not use variables. It uses abstract operators called combinators in order to compose or to transform operators to get more complex operators. Combinators are independent of a restrictive interpretation to a specific use. Combinator’s action is expressed by a unique rule called β-reduction rule; which defines the equivalence between the logical expression without combinator and the one with combinator. In our paper, we will show only five elementary combinators **B, C, S, W, I** (for other combinators, the reader might have a look on (Desclés, Guibert and Sauzay 2016)).

Combinator	β-Reduction
Composition B	**B** x y z ↔ x (y z)
Permutation C	**C** x z y ↔ x y z
Distributive composition S	**S** x y u ↔ x u (y u)
Duplication W	**W** x y ↔ x y y
Identity I	**I** x ↔ x

The composition combinator **B** combines two operators x and y and constructs the complex operator **B** x y that acts on an operand z, z being the operand of y and the result of the application of y to z being the operand of x. The permutation combinator **C** uses an operator x in order to build the complex operator **C** x that acts on the same two operands as x but in reverse order. The distributive composition combinator **S** distributes an operand u to two operators x and y. The result (y u) becomes the operand of the complex operator x u. The duplication combinator **W** takes an operator x that acts on the same operand y twice and constructs the complex operator **W** x that acts on this operand only once. The combinatory **I** expresses the notion of identity.

With elementary combinators, we could construct complex ones, such as "**C B W S** x y z u v". Its global action is determined by the successive application of its elementary combinators (firstly **C,** secondly **B**, then **W** and finally **S**).

C B W S x y z u v
B S W x y z u v
S (**W** x) y z u v
(**W** x) z (y z) u v
W x z (y z) u v
x z (y z) u v

The obtained expression is the normal form, which, according to Church & Rosser theorem, is unique.

Two other forms of complex combinators exist : the power and the distance of a combinator. Let χ be a combinator.

The power of a combinator, noted by χ^n, represents the number n of times its action must be applied. It is recursively defined by: $\chi^0 = \mathbf{I}$; $\chi^1 = \chi$; $\chi^n = \mathbf{B}\, \chi\, \chi^{n-1}$

In other terms, if $\chi = \mathbf{C}$, then:

$\chi^0 = \mathbf{I}$
$\chi^1 = \mathbf{C}$
$\chi^2 = \mathbf{C}^2 = \mathbf{B\ C\ C}$
$\chi^3 = \mathbf{C}^3 = \mathbf{B\ C\ C}^2 = \mathbf{B\ C}\ (\mathbf{B\ C\ C})$
etc.

The distance of a combinator, noted by χ_n, represents the number n of steps its action is postponed. It is recursively defined by: $\chi_0 = \chi$; $\chi_n = \mathbf{B}\ \chi_{n-1}$.

In other terms, if $\chi = \mathbf{C}$, then:

$\chi_0 = \mathbf{C}$
$\chi_1 = \mathbf{C}_1 = \mathbf{B\ C}_0 = \mathbf{B\ C}$
$\chi_2 = \mathbf{C}_2 = \mathbf{B\ C}_1 = \mathbf{B}\ (\mathbf{B\ C})$
$\chi_3 = \mathbf{C}_3 = \mathbf{B\ C}_2 = \mathbf{B}\ (\mathbf{B}\ (\mathbf{B\ C}))$
etc.

Formal Model

In our model, operations contained in programs are translated into applicative terms represented by typed modules. This translation allows a more formal definition of an operation in terms of its internal structure and relation with other operations. Also, this translation allows for a better specification of the processing chain design. A processing chain must be syntactically correct. Thus, given a set of typed modules, what are the allowable arrangements that lead to coherent processing chains?

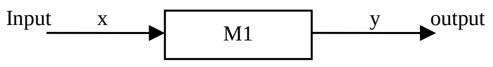

Figure 1. Module schematisation

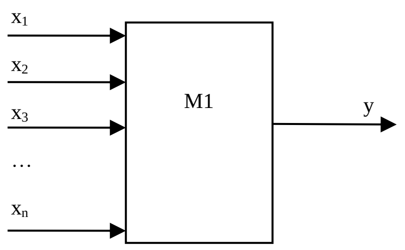

Figure 2. A module with n inputs

To do that, we must, first, assign to each module an applicative type. For example the type Fxy is assigned the module M1 in (Fig. 1) since its input is of type x and its output is of type y. We note the module M1 of type Fxy by [M1 : Fxy]. As a general notation, [M1 : $Fx_1...Fx_ny$] is a module M1 with n inputs of the respective types x_1, x_2, …, x_n, and an output of type y (Fig. 2).

A processing chain is the representation of the order of application of several modules on their inputs. To be valid, the type of an input must be the same as the output linked to it (Fig. 3).

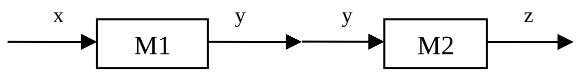

Figure 3. Valid chain of two modules in series

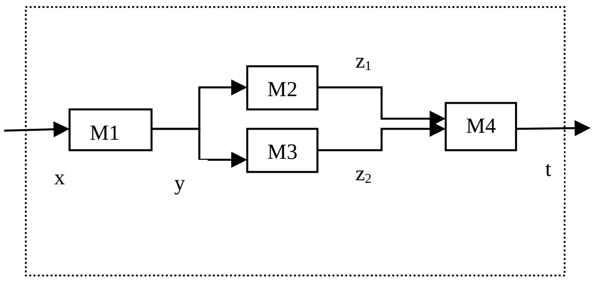

Figure 4. Processing chain as a new module

A processing chain can be seen as a module itself as it has inputs and output (Fig. 4). Our model allows the reduction of a processing chain to this unique module representation. The combinatory logic keeps the execution order and the rules take type in account to check the syntactic correctness. To reduce a chain, we only need the modules list, their type, and their execution order.

Let us show the rules of the model:

APPLICATIVE RULE	$\dfrac{[X : x] + [M1 : Fxy]}{[Y : y]}$
COMPOSITION RULE	$\dfrac{[M1 : Fxy] + [M2 : Fyz]}{[\mathbf{B}\ M2\ M1 : Fxz]}\,\mathbf{B}$
EXTENDED COMPOSITION RULE	$\dfrac{[M1 : Fx_1...Fx_ny] + [M2 : Fyz]}{[\mathbf{B^n}\ M2\ M1 : Fx_1...Fx_nz]}\,\mathbf{B^n}$
DISTRIBUTIVE COMPOSITION RULE	$\dfrac{[M1 : Fxy] + [M2 : FxFyz]}{[\mathbf{S}\ M2\ M1 : Fxz]}\,\mathbf{S}$
PERMUTATION RULE	$\dfrac{[M1 : FxFyz]}{[\mathbf{C}\ M1 : FyFxz]}\,\mathbf{C}$
EXTENDED PERMUTATION RULE	$\dfrac{[M1 : Fx_1...Fx_ny]}{[\mathbf{C_{p-1}}(\mathbf{C_p}(\ldots(\mathbf{C_{m-2}}M1))) : Fx_1...Fx_{p-1}\ Fx_mFx_p...Fx_{m-1}Fx_{m+1}...Fx_ny]}\,\mathbf{C^n}$
DUPLICATION RULE	$\dfrac{[M1 : FxFxy]}{[\mathbf{W}\ M1 : Fxy]}\,\mathbf{W}$
EXTENDED DUPLICATION RULE	$\dfrac{[M1 : (Fx)^ny]}{[\mathbf{W^{n-1}}M1 : Fxy]}\,\mathbf{W^n}$

Extended rules are provided so they can be applied to any number of inputs whereas the others can be applied only to modules with one input. The composition rule is used when two modules are in series (as in Fig. 3). Since M1 is of type Fxy and M2 if type Fyz, the application of the composition rule returns the complex module **B** M2 M1 of type Fxz . If M1 has n inputs, the power of the **B** combinator will be n. We use the extended composition rule as in the (Fig. 5). The module [M1 : $Fx_1Fx_2Fx_3…Fx_ny$] applies on n inputs of different types and yields an output of type y. The module [M2 : Fyz] applies on an input of type y to yield an output of type z. This chain is expressed by the expression: [M1 : $Fx_1Fx_2Fx_3…Fx_ny$] + [M2 : Fyz]. The composition rule can be applied and returns the complex module [$\mathbf{B^n}$ M2 M1 : FxFxy]. If the type of M1 output and M2 input were not the same, we could not have applied the composition rule.

The inputs number of a module can be more than one. The duplication rule (respectively the extended duplication rule) transforms a module with two (respectively n) identical inputs to a module with only one input. This rule can be applied only if the analysis of the processing chain gives the same value to each input (Fig. 6).

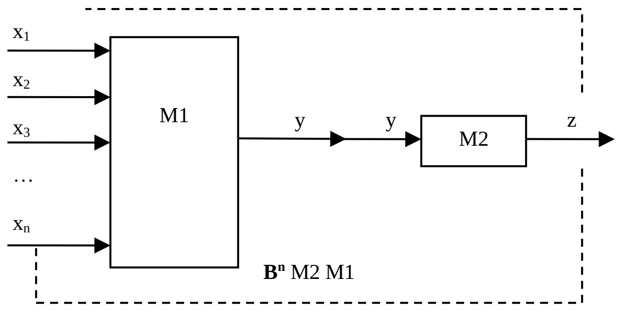

Figure 5. Application of the extended composition rule

The permutation rule allows changing the order of inputs. It takes the input at position m and moves it to the position p, with p<m. It's used to reorganize input to make as much as possible the other rules applicable. Let M be a module with four inputs of types x, y, z and x and an output of type t : [M : FxFyFzFxt]. Let X be the value given to the first and fourth inputs (Fig. 7-a). If the fourth was in second position, we could apply the duplication rule to M. So, we want to move the fourth input to second position. The extended permutation rule returns the complex module [**C₁** (**C₂** M) : FxFxFyFzt] (Fig. 7-b). On this new module, the duplication rule can be applied to get a complex module [**W** (**C₁** (**C₂** M)) : FxFyFzt] (Fig 7-c).

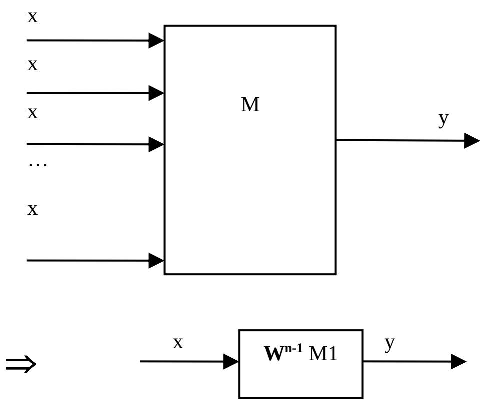

Figure 6. Application of the extended duplication rule

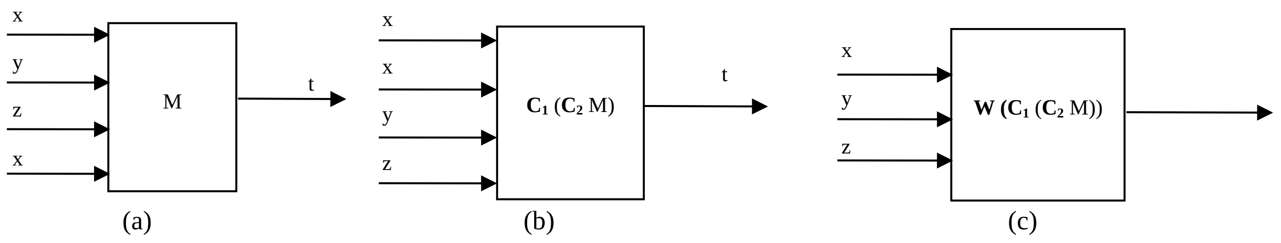

Figure 7. Inputs reorganisation

Implementation

Even if our work is currently at the theoretical stage, a first prototype of the model was implemented. The rules are implemented in a F# library and a testing software in C# language. An open-source library called GraphX is used for graphics visualization.

Two strategies of analysis are considered. The first is from right-to-left (somehow a bottom up analysis). This strategy applies, only, when the whole processing chain is constructed. The second one is from left-to-right (somehow a top down analysis).

The prototype has been tested on 60 different processing chains containing 15 syntactically incorrect chains and 45 correct chains. We wanted to ensure that our approach does not allow undergeneration or overgeneration. The results are shown in table 1. We are, currently, working on the implementation of modules with effective functionalities in the domain of classification. We will present the results of this work in our next publications.

	REDUCED	**NOT REDUCED**
VALID CHAIN	45	0
INVALID CHAIN	0	15

Table 1. Results of reduction

Let us give the analysis of the complex processing chain (Fig. 8) which is a combination of seven modules.

- M1 of type Fyz
- M2 of type Fuy
- M3 of type FzFyt
- M4 of type Fyx
- M5 of type Fxu
- M6 of type FuFxy
- M7 of type Fzu
- X, Y, Z are the inputs of the processing chain with respectively the types x, y and z. T of type t is the output.

To reduce this chain, we will use the left to right strategy. We start by composing M5 and M6, since M6 is of FuFxy and M5 is of type Fxu. The composition rule gives a new complex module [**B** M6 M5 : FxFxy] (Fig. 9). In order to compose this constructed module with the module [M4 : Fyx], we need, first, to reorder its inputs by applying the permutation rule. We get a new module [**C** (**B** M6 M5) : FxFxy] (Fig. 10). We can, now, compose this new module with [M4 : Fyx] and get the module [(**B** (**C** (**B** M6 M5)) M4) : FyFxy] (Fig. 11). Since [M1 : Fyz] takes the output of [(**B** (**C** (**B** M6 M5)) M4) : FyFxy] and [(**B** (**C** (**B** M6 M5)) M4) : FyFxy] has two inputs, we apply the extended composition rule to get the complex modules [($\mathbf{B}^2$ M1 (**B** (**C** (**B** M6 M5)) M4)) : FyFxz] (Fig. 12).

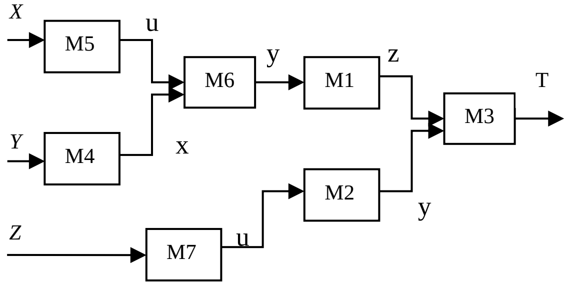

Figure 8. The complex processing chain to be analyzed

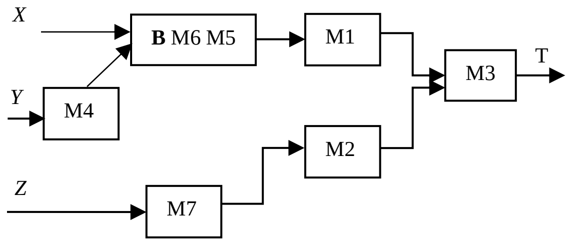

Figure 9. Reduction step 1

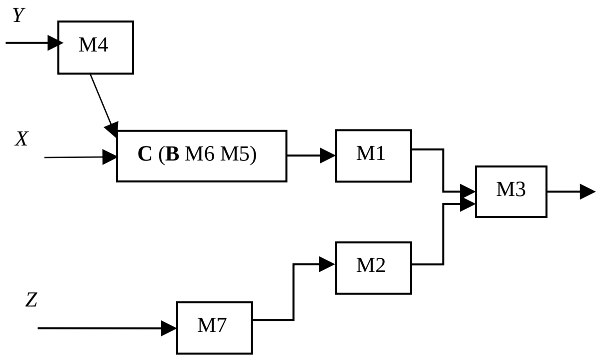

Figure 10. Reduction step 2

The next step allows the composition of [M2 : Fuy] and [M7 : Fzu]. We get the complex module [**B** M2 M7 : Fzy] (Fig. 13). The extended composition rule is applied to [M3 : FzFyt] and [($\mathbf{B}^2$ M1 (**B** (**C** (**B** M6 M5)) M4)) : FyFxz]. It returns the module [($\mathbf{B}^2$ M3 ($\mathbf{B}^2$ M1 (**B** (**C** (**B** M6 M5)) M4))) : FyFxFyt] (Fig. 14). We use the extended permutation rule to reorder the inputs and get the new module [(**C** ($\mathbf{C}_2$ ($\mathbf{B}^2$ M3 ($\mathbf{B}^2$ M1 (**B** (**C** (**B** M6 M5)) M4))))) : FyFxFyt] (Fig. 15). Finally, we can apply the composition rule that returns the module [(**B** (**C** ($\mathbf{C}_2$ ($\mathbf{B}^2$ M3 ($\mathbf{B}^2$ M1 (**B** (**C** (**B** M6 M5)) M4))))) (**B** M2 M7)) : FzFyFxt] (Fig. 16). As we have only one module, and no other rule can be applied, the processing chain is reduced.

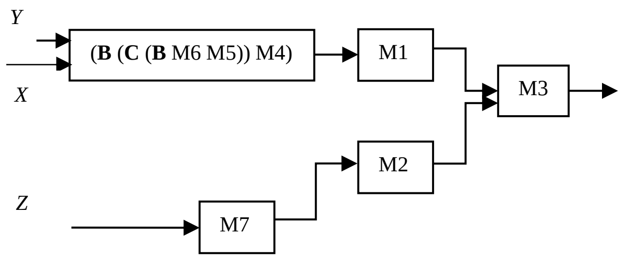

Figure 11. Reduction step 3

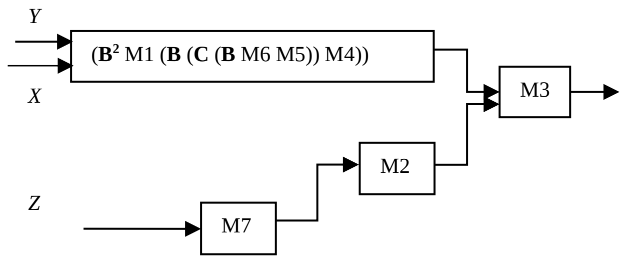

Figure 12. Reduction step 4

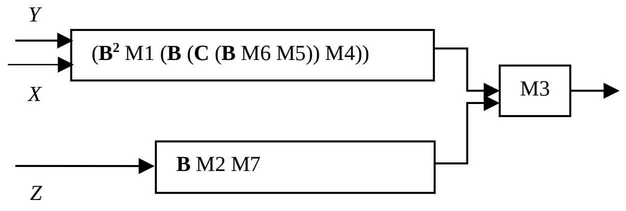

Figure 13. Reduction step 5

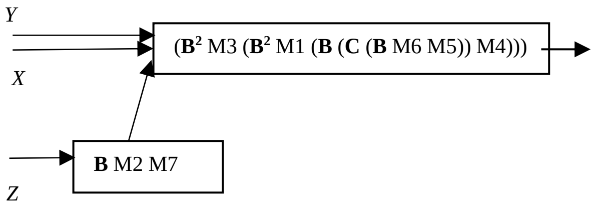
Y
X
Z
($\mathbf{B}^2$ M3 ($\mathbf{B}^2$ M1 (**B** (**C** (**B** M6 M5)) M4)))
B M2 M7

Figure 14. Reduction step 6

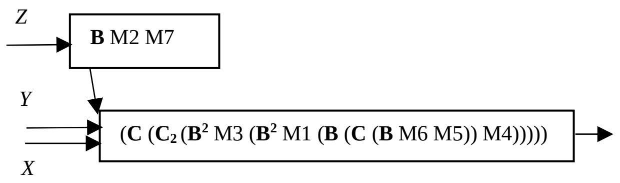
Z
B M2 M7
Y
(**C** ($\mathbf{C_2}$ ($\mathbf{B^2}$ M3 ($\mathbf{B^2}$ M1 (**B** (**C** (**B** M6 M5)) M4)))))
X

Figure 15. Reduction step 7

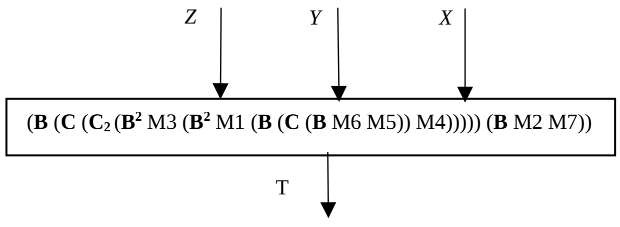
Z
Y
X
(**B** (**C** ($\mathbf{C_2}$ ($\mathbf{B^2}$ M3 ($\mathbf{B^2}$ M1 (**B** (**C** (**B** M6 M5)) M4))))) (**B** M2 M7))
T

Figure 16. Reduction step 8

As it has been completely reduced, the processing chain is considered as syntactically correct. Its combinatory expression is: **B** (**C** ($\mathbf{C_2}$ ($\mathbf{B^2}$ M3 ($\mathbf{B^2}$ M1 (**B** (**C** (**B** M6 M5)) M4))))) (**B** M2 M7). Using combinatory logic reductions, we can get the normal form of this expression.

- **B** (**C** ($\mathbf{C_2}$ ($\mathbf{B^2}$ M3 ($\mathbf{B^2}$ M1 (**B** (**C** (**B** M6 M5)) M4))))) (**B** M2 M7) Z Y X
- **C** ($\mathbf{C_2}$ ($\mathbf{B^2}$ M3 ($\mathbf{B^2}$ M1 (**B** (**C** (**B** M6 M5)) M4)))) ((**B** M2 M7) Z) Y X
- $\mathbf{C_2}$ ($\mathbf{B^2}$ M3 ($\mathbf{B^2}$ M1 (**B** (**C** (**B** M6 M5)) M4))) Y ((**B** M2 M7) Z) X
- $\mathbf{B^2}$ M3 ($\mathbf{B^2}$ M1 (**B** (**C** (**B** M6 M5)) M4)) Y X ((**B** M2 M7) Z)
- M3 (($\mathbf{B^2}$ M1 (**B** (**C** (**B** M6 M5) M4)) Y X) ((**B** M2 M7) Z)
- M3 (M1 ((**B** (**C** (**B** M6 M5)) M4) Y X)) ((**B** M2 M7) Z)
- M3 (M1 ((**C** (**B** M6 M5)) (M4 Y) X)) ((**B** M2 M7) Z)
- M3 (M1 ((**B** M6 M5) X (M4 Y))) ((**B** M2 M7) Z)
- M3 (M1 (M6 (M5 X) (M4 Y))) ((**B** M2 M7) Z)
- M3 (M1 (M6 (M5 X) (M4 Y))) (M2 (M7 Z))

The obtained normal form expresses the order of application of modules on their inputs (Z, Y and X).

It could be possible to use an analysis strategy from right-to-left and the processing chain in Fig. 8 would be reduced to the combinatory expression (**B** (**B** (**C** ($\mathbf{C_2}$ (**B** (**C** (**B** ($\mathbf{B^2}$ (**B** M3 M1) M6) M5)) M4))) M2) M7) *Z Y X*. According to Church-Rosser's theorem this combinatory expression is equivalent to the one obtained in (Fig. 16) since their reduction yields the same normal form.

Conclusion

Several sectors of activity, be they industrial, economic, scientific, cultural, "social networking", etc., are generating more and more textual data than ever before. To know how to use this data, users need flexible, adaptable, consistent and easy-to use tools and platforms that can help to make advanced analytics they need accessible, data processing, data integration, application integration, machine learning, etc. Scientists and even industrialists, like Dassault-Systems and BIOVA for example, have understood the importance of this major issue and what we call *collaborative intelligent science*, in which the user must stay in center of its experience.

With the theoretical model we propose, it would be possible to meet these needs.

References

Biskri, I. and Desclés, J.P. 1997. Applicative and Combinatory Categorial Grammar (from syntax to functional semantics). In *Recent Advances in Natural Language Processing*. John Benjamins Publishing Company.

Biskri, I.; Anastacio, M.; Joly, A.; and Amar Bensaber, B. 2015. A Typed Applicative System for a Language and Text Processing Engineering". *The International Journal of Innovation in Digital Ecosystems*. Elsevier.

Cunningham, H.; Maynard, D.; Bontcheva, K.; and Tablan, V. 2002. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 2002 Association for Computational Linguistics*. Philadelphia.

Curry, B. H. and Feys, R. 1958. *Combinatory logic*, Vol. I, North.Holland.

Desclés, J.P.; Guibert, G.; and Sauzay, B. 2016. *Logique combinatoire et λ-calcul : des logiques d'opérateurs*. Cépaduès.

Downie, J. S.; Unsworth, J.; Yu, B.; Tcheng, D.; Rockwell, G.; and Ramsay, S. J. 2005. A revolutionary approach to humanities computing: tools development and the D2K datamining framework. In *Proceedings of the 17th Joint International Conference of ACH/ALLC*.

Hindley, J. R.; and Seldin, J. P. 2008. *Lambda.calculus and Combinators, an Introduction*. Cambridge University Press.

Mierswa, I.; Wurst, M.; Klinkemberg, R.; Scholz, M.; and Euler, T. 2006. YALE: Rapid Prototyping for Complex Data Mining Tasks. In *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining*.

Seffah, A.; and Meunier, J.G. 1995. ALADIN : Un atelier orienté objet pour l'analyse et la lecture de Textes assistée par ordinaleur. In Proceedings of the 1995 International Conferencence on Statistics and Texts. Rome.

Shaumyan, S. K. 1998. Two Paradigms of Linguistics: The Semiotic Versus Non-Semiotic Paradigm. *Web Journal of Formal, Computational and Cognitive Linguistics*.

Shonenberg, M.H.; Mans, R.S.; Russell, N.C.; Mulyar, N.A.; and Van Der Aalst, W.M.P., 2008. Process Flexibility: a Survey of Contemporary Approaches. In *Advances in Enterprise Engineering*. Lecture Notes in Business Information Processing Book Series. Volume 10.

Warr, A. W. 2007. Integration, analysis and collaboration. An Update on Workflow and Pipelining in cheminformatics. *Strand Life Sciences*.

Witten, I.; Frank, E.; and Hall, M. 2011. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers.