Flexibility in Classification Process

Ismaïl Biskri(✉)

Laboratoire de Mathématiques et Informatique Appliquées,
Université du Québec à Trois-Rivières, Trois-Rivières, Canada
Ismail.Biskri@uqtr.ca

Abstract. A whole classification process is the result of a discovery process that requires constant back and forth between theoretical description of the solution, software implementation, testing and refinement of the theoretical description in the light of the results of experimentation. This process is iterative. It should be, always, under the control of the user according to his subjectivity, his knowledge and the purpose of his analysis. In the last years, several platforms for digging data where classification is the main functionality have emerged. Some of these platforms allow a rapid prototyping and support a re-use of existing "computational modules" from existing "computational tool cases". However, they lack flexibility and sound formal foundations. We propose, in this paper, a formal model with strong logical foundations, based on typed applicative systems. In this model, "computational modules" are considered as operators followed by their operands. A specific processing chain becomes a specific arrangement of a set of modules according to the needs of the user. The model ensures a firm compositionality of this arrangement.

Keywords: Classification · Flexibility · Applicative systems

1 Introduction

Language processing, text processing, social Medias processing, knowledge extraction, applications in education, etc. are a broad field of research including information retrieval, indexation, classification, and information's analysis. As Web is a big source of information, this field can have many implications on several sectors of society. Compared to the quick expansion of data quantity, the evolution of their analysis is too slow and insufficient.

Classification is a major component of many processing chains for language processing, text processing, social Medias processing, knowledge extraction, applications in education, etc. Classification organizes documents, files, data, etc. in such a way that similar documents, files, data, etc. are grouped together. From a computational point of view, classification relies on statistical comparison of content-based descriptors identified by the user according to his needs and goals.

The whole classification process may be divided in three steps. The first one is the features extraction. Extracted features are used as content - based descriptors. The user decides the nature of these descriptors. He may decide to replace them by other descriptors. Indeed, the main difficulty is not extracting the features, over the years,

A. Amine et al. (Eds.): CIIA 2018, IFIP AICT 522, pp. 279–289, 2018.
https://doi.org/10.1007/978-3-319-89743-1_25

many libraries have been developed and shared to facilitate the features extraction; but it is choosing the right features.

The second step is the classification process itself. Different classifiers such as neural network based classifier can be used to perform this step. One of the limitations of these classifiers is that they are opaque and it's not always easy to evaluate the contribution of a specific descriptor and a specific classifier in the classification result.

The third step is the interpretation by the user of the classification results. A user can use different tools to make an enlightened reading of the results. The choice of the tool strongly depends on the purpose of the analysis, or even the knowledge and subjectivity of the user.

Features extraction, classifiers and interpretation tools have an impact on the classification process. Different combinations of features, classifiers and interpretation tools should continue to be explored in order to improve the computer assisted classification process or in some cases better customize the classification process to specific application areas.

In the literature about classification, data-mining, text-mining, and Big-Data, many projects aim to allow the creation of complex processing chains. ALADIN [9], D2K/T2K [5], RapidMiner [8], Knime [11] and WEKA [12] use *processing chains* for language and data engineering, Gate [4] use it for linguistic analysis. The processing chains are widely used, but the solutions previously mentioned suffer from limitations. They are strongly bonded to their specific platforms and programming languages. To take the best advantage of them, the user needs to have knowledge about the developed software and sometimes about programming language. The user is not, always expert, in programming language. A big challenge in these fields is the multitude of disciplines needed to go further. So, experts of different domains need to work together.

In our paper, 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. Typed application systems are widely used in the field of automatic processing of natural language with the current of categorial grammars [1–3] and applicative grammars [10]. They allowed the construction of parsers for many languages like French, Arabic, English, Dutch, Korean, etc. They also allowed logical representations for linguistic or cognitive operators.

Before presenting the formal model itself, we will first introduce, in the next section, typed applicative systems and combinatory logic.

2 Combinatory Logic and Typed Applicative Systems

Combinatory logic was introduced by Moses Schönfinkel in 1924, and extended by Curry and Feys [6, 7]. This logic uses abstract operators called combinators in order to eliminate the need of the variables and, thus, to avoid variable telescoping. Combinators act as functions over argument within a typed operator-operand structure. Their 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. Elementary combinators can be associated to others to create complex combinators. In our paper, we use only the four elementary combinators **B, C, S, W**, whose notations and β-reductions are shown in the table below.

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

The composition combinator **B** combines two operators x and y together 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 composition combinator **S** distributes an operand u to two operators x and y.

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.

We can combine elementary combinators together to construct more complex combinators. For example, we could have an expression such as "**B S C** x y z u v". Its global action is determined by the successive application of its elementary combinators (first **B** secondly **S** and finally **C**).

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

The resulting expression, without combinators, is called a normal form. This form, 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 defined recursively by $\chi^1 = \chi$ and $\chi^n = \mathbf{B}\ \chi\ \chi^{n-1}$. For example, the action of the expression $\mathbf{C}^2$ x y z would be:

$\mathbf{C}^2$ x y z
B C C x y z
C (**C** x) y z
(**C** x) z y
C x z y
x y z

The distance of a combinator, noted by χ_n, represent the number n of steps its action is postponed. It is defined by $\chi_0 = \chi$ and $\chi_n = \mathbf{B}^n \chi$. For example, the action of the expression $\mathbf{C_2}$ x y z u v will be:

$\mathbf{W}_2$ x y z u v
$\mathbf{B}^2$ **W** x y z u v
B B B W x y z u v
B (**B W**) x y z u v
(**B W**) (x y) z u v
B W (x y) z u v
W ((x y) z) u v
((x y) z) u u v
x y z u u v

Applicative systems assign to each operator and to each operand an applicative type to express how they work. The set of applicative types is recursively defined as follows:

1. Basic types are types.
2. If x and y are types, Fxy is a type.

Fxy is the applicative type of an operator whose operand is of type x and the result of its application to its operand is of type y.

An operator function having two x typed operand and returning a y typed result will be of type FxFxy. This can also be read as: an operator taking an x type operand and giving back an operator taking an x type operand and returning an y type result.

3 Formal Model

In our model, we aim at explicitly defining the set of operations contained in programs. In the applicative modeling, these operations are translated into functional 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. Typed modules are organized in series and as such they form processing chains. A typed module acts then like a mathematical function that takes several arguments, process them and return an output. Here, we are not interested in the internal programming of the modules but only in their representation as functions and how they are organized to create processing chains.

A processing chain must be syntactically correct. Its semantic interpretation depends, mainly, on the user's point of view regarding the expected analysis.

Our model tends to answer two questions:

- Given a set of typed modules, what are the allowable arrangements that lead to coherent processing chains?
- Given a coherent processing chain, how can we automate as much as possible its assessment?

Fig. 1. Module schematisation

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 as follow: [M1 : Fxy].

As a general notation, [M1 : $Fx_1 \ldots Fx_n y$] is a module M1 with n inputs of different types, input in place "i" is of type x_i, and an output of type y, [M2 : $(Fx)^n y$] is a module M2 with n inputs of type x and an output of type y.

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. 2). It also can be seen as a module itself as it has inputs and output (Fig. 3).

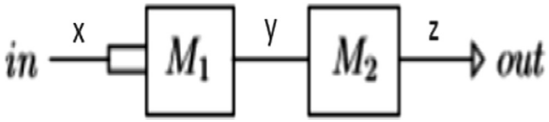


Fig. 2. Valid chain of two modules in series

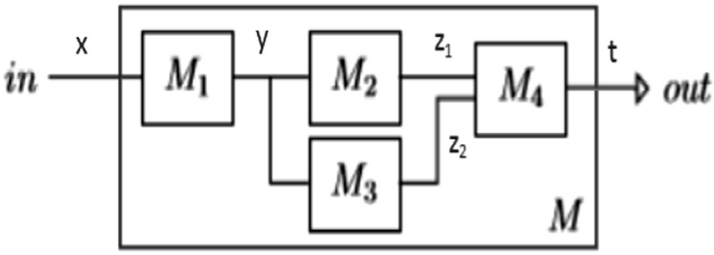


Fig. 3. Processing chain as a new module

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 these rules:

APPLICATIVE RULE	[X : x] + [M1 : Fxy] ---------- [Y : y]
COMPOSITION RULE	[M1 : Fxy] + [M2 : Fyz] ----------**B** [**B** M2 M1 : Fxz]
DISTRIBUTIVE COMPOSITION RULE	[M1 : Fxy] + [M2 : FxFyz] ----------**S** [**S** M2 M1 : Fxy]
PERMUTATION RULE	[M1 : FxFyz] ----------**C** [**C** M1 : FyFxz]
DUPLICATION RULE	[M1 : FxFxy] ----------**W** [**W** M1 : Fxy]

The above rules are only the core set of the model. Extended rules are provided so they can be applied to any number of inputs.

COMPOSITION RULE	[M1 : $Fx_1...Fx_ny$] + [M2 : Fyz] ---------- $\mathbf{B^n}$ [$\mathbf{B^n}$ M2 M1 : $Fx_1...Fx_nz$]
PERMUTATION RULE	[M1 : $Fx_1...Fx_ny$] ---------- $\mathbf{C^n}$ [$\mathbf{C_{p-1}}(\mathbf{C_p}(\ldots(\mathbf{C_{m-2}}M1))) : Fx_1...Fx_{p-1}Fx_mFx_p...Fx_{m-1}Fx_{m+1}...Fx_ny$]
DUPLICATION RULE	[M1 : $(Fx)^ny$] ---------- $\mathbf{W^n}$ [$\mathbf{W^n}$M1 : Fxy]

The composition rule is used when two modules are in series (as in Fig. 2). If M1 has n inputs, the power of the B combinator is n. For these rules, the inputs number of M2 can be more than one. The duplication rule transforms a module with n identical inputs to a module with only one input. It can be applied only if the chain give the same value to each of its inputs (Fig. 4). The permutation rule allows to change 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 the other rules applicable.

Fig. 4. Module getting a single value in its three inputs

4 Application of the Approach

In this section, we will show how the rules given in the previous section are applied and illustrate the reduction of a processing chain with an example.

Let us consider the linear connection of two modules (Fig. 2). The module [M1 : FxFxy] applies on two identical inputs of type x and yield an output of type y. The module [M2 : Fyz] applies on this output to yield an output of type z. This chain is expressed by the expression: [M1 : FxFxy] + [M2 : Fyz]. The composition rule can be applied and returns the complex module [$\mathbf{B^2}$ M2 M1 : FxFxy]. If the type of M1 output and M2 output where not the same, we could not have applied the composition rule. So, the application of the rules is a proof of syntactic correctness of the chain.

The module [$\mathbf{B^2}$ M2 M1 : FxFxy] can be reduced a second time with the duplication rule. It is reduced to the complex module [**W** ($\mathbf{B^2}$ M2 M1) : Fxy].

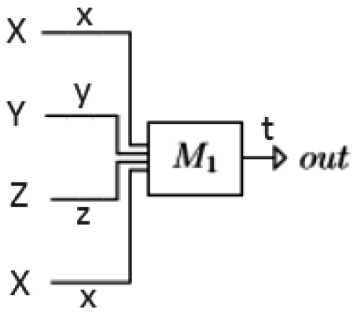
The permutation rule allows reorganising the inputs of a module to apply another rule. Let M be a module with four inputs of types x, y, z and x and an output of type t: [M : FxFyFzFxt]. Let X be the value given to the first and fourth inputs (Fig. 5a). If the fourth was in second position, we could apply the duplication rule to M. So, we want to move the fourth input to second position. The permutation rule returns the complex module [$\mathbf{C_1}$ ($\mathbf{C_2}$ M) : FxFxFyFzt] (Fig. 5b). On this new module, the duplication rule can be applied to get a complex module [**W** ($\mathbf{C_1}$ ($\mathbf{C_2}$ M)) : FxFyFzt] (Fig. 5c).

Let us now give the analysis of a somewhat complex processing chain (Fig. 6). This chain is a combination of five modules.

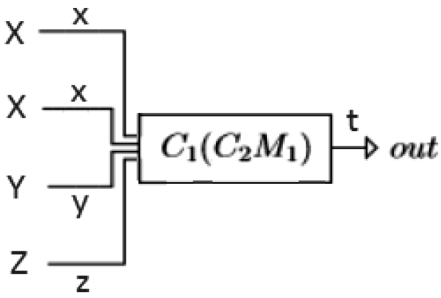
- M1 of type FxFyz
- M2 of type Fzx
- M3 of type Fzx
- M4 of type Fzy
- M5 of type FxFxFyt

To reduce this chain, we will start with the last module and process from left to right. So we start with [M5 : FxFxFyt]. His first input takes the output of [M2 : Fzx]. The composition rule gives a new complex module [**B** M5 M2 : FzFxFyt] (Fig. 7). This new module and [M3 : Fzx] can be reduced with the distributive composition rule to get the module [**S** (**B** M5 M2) M3 : FzFyt] (Fig. 8). The first input of this module can be reduced with the composition rule to get a new module [$\mathbf{B^2}$ (**S** (**B** M5 M2) M3) M1 : FxFyFyt] (Fig. 9).

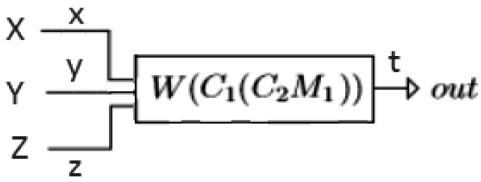
To reduce this module with [M4 : Fzy] we want to use the composition rule. But to apply it, M4 output must be the first input of our module. We use the permutation rule to reorganise the inputs and got a new module [**C** ($\mathbf{C_2}$ ($\mathbf{B^2}$ (**S** (**B** M5 M2) M3) M1)) : FyFxFyt] (Fig. 10). Finally, we can apply the combination rule that returns the module



(a)



(b)



(c)

Fig. 5. Inputs reorganisation

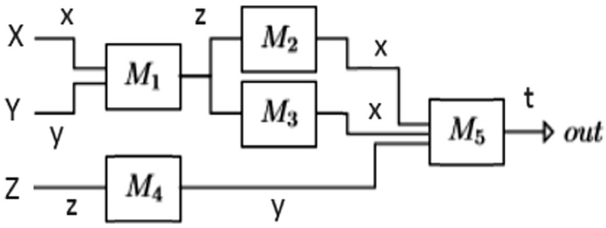


Fig. 6. A complex processing chain

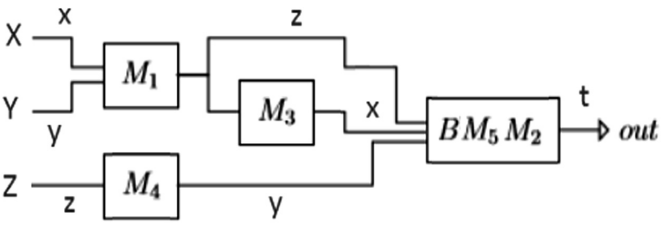


Fig. 7. Reduction step 1

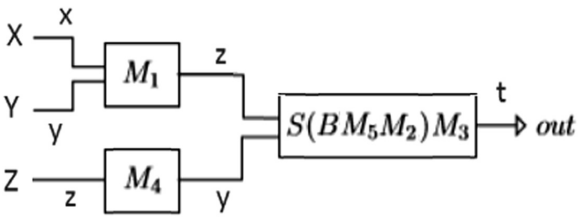


Fig. 8. Reduction step 2

Fig. 9. Reduction step 3

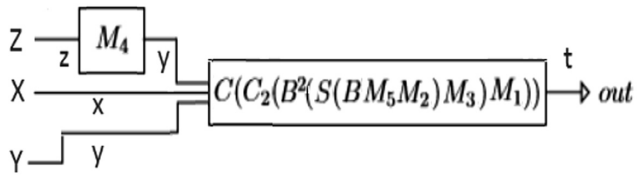


Fig. 10. Reduction step 4

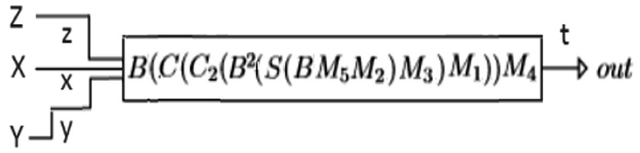


Fig. 11. Reduction last step

[**B** (**C** (**C**$_2$ (**B**^{2} (**S** (**B** M5 M2) M3) M1))) M4 : FzFxFyt]. As we have only one module, and no other rule can be applied, the processing chain is reduced (Fig. 11).

As it has been completely reduced, the processing chain is considered as syntactically correct. Its combinatory expression is: **B** (**C** (**C**$_2$ (**B**^{2} (**S** (**B** M5 M2) M3) M1))) M4. Using combinatory logic reductions, we can get the normal form of this expression.

B (**C** (**C**$_2$ (**B**^{2} (**S** (**B** M5 M2) M3) M1))) M4 Z X Y

C (**C**$_2$ (**B**^{2} (**S** (**B** M5 M2) M3) M1)) (M4 Z) X Y

C$_2$ (**B**^{2} (**S** (**B** M5 M2) M3) M1) X (M4 Z) Y

B^2 (**S** (**B** M5 M2) M3) M1 X Y (M4 Z)

S (**B** M5 M2) M3 (M1 X Y) (M4 Z)

B M5 M2 (M1 X Y) (M3 (M1 X Y)) (M4 Z)

M5 (M2 (M1 X Y)) (M3 (M1 X Y)) (M4 Z)

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

Even if this 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.

The prototype has been tested on 40 different processing chains containing 15 syntactically incorrect chains and 25 correct chains. The results are shown in Table 1. We are, currently, working on the implementation of modules with effective functionalities in the domain of classification.

Table 1. Results of reduction

	Reduced	Not reduced
Valid chain	25	0
Invalid chain	0	15

5 Conclusion

The need for flexible, adaptable, consistent and easy-to-use tools and platforms is essential. But, many challenges are yet to be solved. The user stays in center of its experience and he can change his mind. The flexibility of the tools is really important when it happens. Without it, user needs to, constantly, go back and forth between theoretical description of the solution, software implementation, testing and refinement of the theoretical description in light of experimentation results. The model that we

propose allows rapid prototyping and support a maximal re-use and composition of existing modules. It also ensures a firm compositionality of the different modules in the different processing chains. Moreover, our approach provides a general framework in which users would be able to build multiple language and text analysis processes according to their own objectives.

References

1. Biskri, I., Desclés, J.P.: Applicative and combinatory categorial grammar (from syntax to functional semantics). In: Recent Advances in Natural Language Processing. John Benjamins Publishing Company, Amsterdam (1997)
2. Biskri, I., Anastacio, M., Joly, A., Amar Bensaber, B.: A typed applicative system for a language and text processing engineering. In: International Journal of Innovation in Digital Ecosystems. Elsevier, Amsterdam (2015)
3. Biskri, I., Anastacio, M., Joly, A., Amar Bensaber, B.: Integration of sequence of computational modules dedicated to text analysis: a combinatory typed approach. In: Proceedings of AAAI. FLAIRS 2013, St. Pete Beach (2013)
4. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: a framework and graphical development environment for robust NLP tools and applications. In: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL 2002), Philadelphia (2002)
5. Downie, J.S., Unsworth, J., Yu, B., Tcheng, D., Rockwell, G., Ramsay, S.J.: A revolutionary approach to humanities computing: tools development and the D2K datamining framework. In: Proceedings of the 17th Joint International Conference of ACH/ALLC (2005)
6. Curry, B.H., Feys, R.: Combinatory Logic, vol. I. North Holland, Amsterdam (1958)
7. Hindley, J.R., Seldin, J.P.: Lambda-Calculus and Combinators, an Introduction. Cambridge University Press, Cambridge (2008)
8. Mierswa, I., Wurst, M., Klinkemberg, R., Scholz, M., Euler, T.: YALE: rapid prototyping for complex data mining tasks. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006). ACM Press (2006)
9. Seffah, A., Meunier, J.G.: ALADIN: Un atelier orienté objet pour l'analyse et la lecture de Textes assistée par ordinaleur. In: International Conferencence on Statistics and Texts, Rome (1995)
10. Shaumyan, S.K.: Two paradigms of linguistics: the semiotic versus non-semiotic paradigm. Web J. Formal Comput. Cogn. Linguist. **2**, 1–72 (1998)
11. Warr, A.W.: Integration, Analysis and Collaboration. An Update on Workflow and Pipelining in Cheminformatics. Strand Life Sciences (2007)
12. Witten, I., Frank, E., Hall, M.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers, Burlington (2011)