Category theory for optimization

Heng Zhao*, Laurent Thiry* and Michel Hassenforder*
*MIPS, University of Haute-Alsace, Mulhouse, France
Email: {heng.zhao, laurent.thiry, michel.hassenforder}@uha.fr

***Abstract*—This paper shows how concepts coming from category theory can help to improve the algorithms dealing with large set of data. Data structures can be modeled by functors that are related by natural transformations usable both to reduce data size or to shift an algorithm applicable to a particular data structure to an equivalent algorithm for another data structure, ie. results are the same but time required to get it can be different. As an illustration, the paper takes the example of queries on graph databases used by semantic web and big data communities.**

I. Introduction

The performance of an algorithm mostly depend on the data structure considered, and a well-known optimization technic is "memoization" that stores a result (rather than to compute it) and directly returns it when needed. An other common optimization consists in splitting data and use concurrent computations ("divide and conquer" principle). The question is then: how to transform an existing algorithm to integrate such an optimization principle and get better performance ? The paper proposes to profit of the fundamental concepts from category theory to formalize both data structures (with functors F_i), relations between this structures (with natural transformations/isomorphisms $\eta_{ij} : Fi \to F_j$) and algorithms using a particular structure (with catamorphisms $c_i : F_i \to a$). An optimization is then a combination of these elements (i.e. $c_j = c_i \circ \eta_{ji}$ that can be simplified by using cut-fusion principle). As an illustration, the paper consider graph structures - the (F_i) then represents various representations of a graph, and graph queries (see c_i) that consists in finding a specific subgraph in a graph. It then gives the times requires for each model to show optimizations. The paper is divided into two parts. The first one presents the categorical concepts used in the proposition. The second one gives an application to graph databases, and discusses the performances obtained.

II. Categorical concepts

A. Foundations

Category theory is a field of mathematics dealing with "structures". A category is composed with objects and morphisms/arrows about them [1]. There is an identity morphism for each object and a composition operator $(\circ)$, that is associative and has id as neutral element. An example of a category is $\mathcal{S}et$ that has sets s_i as objects and functions $f_j : s_k \to s_l$ as arrows. $(\circ)$ is then the functional composition. This category is also cartesian closed: all its products $s_i \times s_j$ is also an object. A functor is a structure preserving map that transforms objects and arrows by respecting identities and composition. An example is the powerset functor $\mathcal{P} : \mathcal{S}et \to \mathcal{S}et$ such as $\mathcal{P}(s_i)$ is a subset of s_i, and $\mathcal{P}(f)\{x_1...x_n\} = \{f(x_1)...f(x_n)\}$. Another example is the product functor obtained by defining the product of two functions: $(f_i \times f_j)(x_i, x_j) = (f_i(x_i), f_j(x_j))$. These functors can be composed to obtain new ones, e.g. $\mathcal{P}(s_i \times s_j)$, $\mathcal{P}(s_i \times \mathcal{P}(s_j))$, etc. A natural transformation is a functor transformation that preserve the structure (ie. composition). An example is $\eta : \mathcal{P}(s_i \times s_j) \to \mathcal{P}(s_i \times \mathcal{P}(s_j))$ with $\eta(s) = \{(x, \{y \mid (x, y) \in s\}) \mid x \in \mathcal{P}(\pi_1)(s)\}$ and $\pi_1(x_1, x_2) = x_1$. This transformation is invertible, ie. $\exists\eta', \forall s, \eta'(\eta(s)) = s$, and the functors are said to be naturally isomorphic (unformally, there is no lost of information) what is written $\mathcal{P}(s_i \times s_j) \cong_\eta \mathcal{P}(s_i \times \mathcal{P}(s_j))$.

B. Graph databases (and semantic web)

A directed graph $G = (N, E)$ is defined by a set of nodes N and a set of edges $E \subseteq N \times N$, or equivalently $E \in \mathcal{P}(N \times N)$. With the natural transformation presented above the last expression is equivalent to $E \in \mathcal{P}(N \times \mathcal{P}(N))$. A graph morhism $m : G(N, E) \to G'(N', E')$ is a mapping $f : N \to N'$ that preserves the structure, ie. if $(x_1, x_2) \in E$ then $(f(x_1), f(x_2)) \in E'$. An example is presented in the Figure 1.

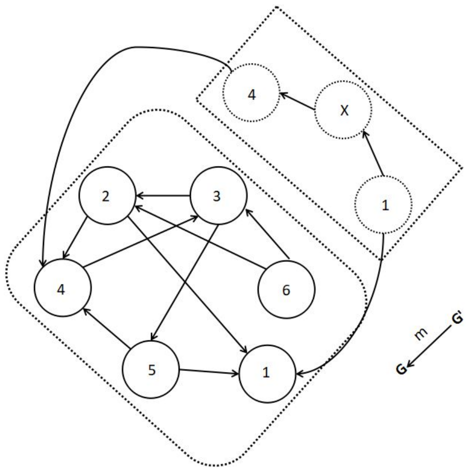

Fig. 1. Sample graphs and graph morphism.

A graph can represent a relation and $(x_1, x_2) \in G$ can be interpreted as "x_1 is a friend of x_2" for instance. Graph morphisms can then serve to answer queries and find a precise information such as "who are the common friends to 1 and 4 ?". This can be represented by the logical expression $\exists X, (1, X) \in G \wedge (4, X) \in G$ that is graphically represented by the graph G'. These concepts are a central part of the

0-7695-6360-0/18/$31.00 ©2018 IEEE
DOI 10.1109/ICSC.2018.00055

semantic web [2] and involves complex algorithms due to the size of data to be treated (size of the graph) and the problem to be solved, ie. finding a morphism m.

III. Algorithmic optimizations

A. Functorial data structures

The concept of "functor" is usefull to model data structures. Products $(x, y) \in X \times Y$ corresponds to records with two accessors functions $\pi_1 : X \times Y \to X$ and $\pi_2 : X \times Y \to Y$. Powerset models the well known datatype "set of Xs", i.e. $\{x, y, z\} \in \mathcal{P}(X)$. These functors can be composed to model relations, e.g. $\{(1, x), (2, y), (3, z)\} \in \mathcal{P}(\mathbb{N} \times X)$, or graph as illustrated in the previous part. As a remark, the previous example can be used to model the "list of Xs" datatype, i.e. $[x, y, z] : L(X)$. The use of lists, rather than sets, is more interesting in the sense their are naturally found in most of the functional programming languages. In particular, the following functors have been encoded in Haskell to compare the performances (time required) in returning the result of a query for various database structures.

Common data structures used by information systems can now be modeled by four functors. Relational databases use a (indexed) set of tables that can be modeled by binary relations, and will be represented in a simplified version by $F_1(X) = L(X \times L(X, X))$ - the first X corresponds to the table name associated to a list of records (the second X corresponds to the first column and the third X to the second one). Graph databases use labeled graphs that can be represented by $F_2(X) = L(X \times (X \times X))$ where the first X corresponds to the source of an edge, the second to the label and the third to the destination. Document databases are composed by objects having a list of properties, and can be represented by $F_3(X) = L(X \times L(X \times X))$ where the first X correspond to a object, the second to a property name, and the third to the property value. Finally, distributed databases with (two computers) can be represented by a couple of databases, i.e. $F_4(X) = F_2(X) \times F_2(X)$ if the local databases are graph databases. The change of a data structure to another one can be formalized by a natural transformation $\eta_{ij} : F_i(X) \to F_j(X)$ and is, with the previous definitions, invertible.

As explained in section II-B, a precise search of an information can use a logical expression that can be represented itself by a graph with variables. An example is, if q="(?x married sophie)", "query2 q graphdb" that will return [(?x,laurent)] if the graphdb contains the edge (laurent married sophie). The signature of the function[1] is then $(X \times X \times X) \to F_2(X) \to L(X \times X)$. Having this algorithm, one can shift the program to query other database structures by using natural transformations. For instance, "query1 p relationnaldb = query2 p (η_{12} relationnaldb)" that has type $(X \times X \times X) \to F_1(X) \to L(X \times X)$, By using pointwise representation, the program can be expressed by "query1 p = (query2 p) η_{12}". Now, the short cut-fusion can be used to suppress the composition ($\circ$) by using the definition of query2 and η_{12} to eliminate unecessary computations and get efficient code for query1. This can be applied to the other database models to get query3 on F_3 and query4 for F_4.

[1]The code is detailled in [3]

B. Results

The performances obtained to get the results of each $query_i$ has been measured by finding some informations from the titanic dataset[2]. More precisely, the queries considered are the following ones:

$q1 = (?X\ Sex\ male)$
$q2 = (?X\ Sex\ male)\ and\ (?X\ Survived\ 1)$
$q3 = (?X\ ?Y\ ?Z)$

The times obtained to get the result according to a percentage of the whole database are:

Size \ Query	q1	q2	q3
0.1%	0.00084s	0.00047s	0.00044s
1%	0.00891s	0.00389s	0.00637s
10%	0.04412s	0.04378s	0.04432s
50%	0.17314s	0.18080s	0.17075s
100%	0.34306s	0.36010s	0.35782s

TABLE I
Performance of differents size of database

An analysis of the results is: the time required is approximatively proportional to the size of the database. The performance of this data model is depended on the number of edges. The edges are similar with the example in III-A. However, if it has used the distributed databases with two computers , the time required could be reallly improved. For example, there is a transformation below: $trf'\ [(x, v1), (x, v2), (x, v3)] == [(x, [v1, v2, v3])]$ if the results set has $n\ (key, value)$ elements, corresponding to the possible values of the variables in a query, then $trf'\ (result)/result'$ has approximately n/2 (key,values) elements.

Size\Query	q1	q2	q3
100%	0.00365s	0.00071s	0.16356s

TABLE II
Performance for new data structure

The distributed version of this new application, except for q3, 10x more faster than the original one.

IV. Conclusion

As an useful tool, Category theory could be used to formalize data structure, relations between these structures. By using this approach, the results are the same but time required to get it can be different. There are other capabilities offered by the system such as graph rewrting that are currently studied and will be presented in the future.

References

[1] J.A.Goguen. A categorical manifesto. In Mathematical Strucutre in Computer Science, pages 49-67,1991;

[2] T.Segaran, C.Evans ans J.Taylor. Programming the Semantic Web. O'Reilly Media, Inc., 1st edition, 2009.

[3] L.Thiry, M.Manfoudh, and M.Hassenforder. A functional inference system for the web. IJWA, 6(1):1-13,2014.

[2]Available at https://www.kaggle.com/c/titanic/data