Available online at [www.sciencedirect.com](http://www.sciencedirect.com/)

[Electronic Notes in Theoretical Computer Science 298 (2013) 283–307](http://dx.doi.org/10.1016/j.entcs.2013.09.018)

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

A Categorical Theory of Patches

Samuel Mimram and Cinzia Di Giusto

*CEA, LIST*

Abstract

When working with distant collaborators on the same documents, one often uses a version control system, which is a program tracking the history of files and helping importing modifications brought by others as patches. The implementation of such a system requires to handle lots of situations depending on the operations performed by users on files, and it is thus difficult to ensure that all the corner cases have been correctly addressed. Here, instead of verifying the implementation of such a system, we adopt a complementary approach: we introduce a theoretical model, which is defined abstractly by the universal property that it should satisfy, and work out a concrete description of it. We begin by defining a category of files and patches, where the operation of merging the effect of two coinitial patches is defined by pushout. Since two patches can be incompatible, such a pushout does not necessarily exist in the category, which raises the question of which is the correct category to represent and manipulate files in conflicting state. We provide an answer by investigating the free completion of the category of files under finite colimits, and give an explicit description of this category: its objects are finite sets labeled by lines equipped with a transitive relation and morphisms are partial functions respecting labeling and relations.

*Keywords:* Category theory, patch, coinitial patch

# Introduction

It is common nowadays, when working with distant collaborators on the same files (multiple authors writing an article together for instance), to use a program which will track the history of files and handle the operation of importing modifications of other participants. These software called *version control systems* (vcs for short), like git or Darcs, implement two main operations. When a user is happy with the changes it has brought to the files it can record those changes in a *patch* (a file coding the differences between the current version and the last recorded version) and *commit* them to a server, called a *repository*. The user can also *update* its current version of the file by importing new patches added by other users to the repository and applying the corresponding modifications to the files. One of the main difficulties to address here is that there is no global notion of “time”: patches are only partially ordered. For instance consider a repository with one file *A* and

1 This work was partially supported by the French project ANR-11-INSE-0007 REVER.

1571-0661 © 2013 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

<http://dx.doi.org/10.1016/j.entcs.2013.09.018>

two users *u*1 and *u*2. Suppose that *u*1 modifies file *A* into *B* by committing a patch *f* , which is then imported by *u*2, and then *u*1 and *u*2 concurrently modify the file *B* into *C* (resp. *D*) by committing a patch *g* (resp. *h*). The evolution of the file is depicted on the left and the partial ordering of patches in the middle:

h/g *E* ,g*¸*/h

*C* ,*¸*

g

*D,*

h

*g* ,*¸*  *h ,*

*C* ,*¸*

*D,*

*B*,*,* g h

*B*

f *f* ,*,*

*A* f

*A*

Now, suppose that *u*2 imports the patch *g* or that *u*1 imports the patch *h*. Clearly, this file resulting from the *merging* of the two patches should be the same in both cases, call it *E*. One way to compute this file, is to say that there should be a patch *h/g*, the *residual* of *h* after *g*, which transforms *C* into *E* and has the “same effect” as *h* once *g* has been applied, and similarly there should be a patch *g/h* transforming *D* into *E*. Thus, after each user has imported changes from the other, the evolution of the file is as pictured on the right above. In this article, we introduce a category *L* whose objects are files and morphisms are patches. Since residuals should be computed in the most general way, we formally define them as the arrows of pushout cocones, i.e. the square in the figure on the right should be a pushout.

However, as expected, not every pair of coinitial morphisms have a pushout in the category *L*: this reflects the fact that two patches can be conflicting (for instance if two users modify the same line of a file). Representing and handling such conflicts in a coherent way is one of the most difficult part of implementing a vcs (as witnessed for instance by the various proposals for Darcs: mergers, conflictors, graphictors, etc. [[10](#_bookmark35)]). In order to be able to have a representation for all conflicting files, we investigate the free completion of the category *L* under all pushouts, this category being denoted *P*, which corresponds to adding all conflicting files to the category, in the most general way as possible. This category can easily be shown to exist for general abstract reasons, and one of the main contributions of this work is to provide an explicit description by applying the theory of presheaves. This approach paves the way towards the implementation of a vcs whose correctness is deduced from universal categorical properties.

*Related work.* The Darcs community has investigated a formalization of patches based on commutation properties [[10](#_bookmark35)]. *Operational transformations* tackle essen- tially the same issues by axiomatizing the notion of residual patches [[9](#_bookmark33)]. In both cases, the fact that residual should form a pushout cocone is never explicitly stated, excepting in informal sentences saying that “*g/f* should have the same effect as *g* once *f* has been applied”. We should also mention another interesting approach to the problem using inverse semigroups in [[4](#_bookmark29)]. Finally, Houston has proposed a category with pushouts, similar to ours, in order to model conflicting files [[3](#_bookmark27)], see Section [6](#_bookmark23).

*Plan of the paper.* We begin by defining a category *L* of files and patches in Sec-

tion [2](#_bookmark0). Then, in Section [3](#_bookmark4), we abstractly define the category *P* of conflicting files obtained by free finite cocompletion. Section [4](#_bookmark8) provides a concrete description of the construction in the simpler case where patches can only insert lines. We give some concrete examples in Section [5](#_bookmark21) and adapt the framework to the general case in Section [6](#_bookmark23). We conclude in Section [7](#_bookmark25).

# Categories of files and patches

In this section, we investigate a model for a simplified vcs: it handles only one file and the only allowed operations are insertion and deletion of lines (modification of a line can be encoded by a deletion followed by an insertion). We suppose fixed a set *L* = *{a, b, .. .}* of *lines* (typically words over an alphabet of characters). A *ﬁle A* is a finite sequence of lines, which will be seen as a function *A* : [*n*] *→ L* for some number of lines *n ∈* N, where the set [*n*] = *{*0*,* 1*,...,n −* 1*}* indexes the lines of the files. For instance, a file *A* with three lines such that *A*(0) = *a*, *A*(1) = *b* and *A*(2) = *c* models the file *abc*. Given *a ∈ L*, we sometimes simply write *a* for the file *A* : [1] *→ L* such that *A*(0) = *a*. A morphism between two files *A* : [*m*] *→ L* and *B* : [*n*] *→ L* is an injective increasing partial function *f* : [*m*] *→* [*n*] such that

*∀i ∈* [*m*]*,B ◦ f* (*i*) = *A*(*i*) whenever *f* (*i*) is defined. Such a morphism is called a

*patch*.

**Definition 2.1** The category *L* has files as objects and patches as morphisms.

Notice that the category *L* is strictly monoidal with [*m*] *⊗* [*n*] = [*m* + *n*] and for every file *A* : [*m*] *→ L* and *B* : [*n*] *→ L*, (*A ⊗ B*)(*i*) = *A*(*i*) if *i < m* and (*A ⊗ B*)(*i*) = *B*(*i − m*) otherwise, the unit being the empty file *I* : [0] *→ L*, and tensor being defined on morphisms in the obvious way. The following proposition shows that patches are generated by the operations of inserting and deleting a line:

**Proposition 2.2** *The category L is the free monoidal category containing L as objects and containing, for every line a ∈ L, morphisms η*a : *I → a (insertion of a line a) and ε*a : *a → I (deletion of a line a) such that ε*a *◦ η*a = idI *(deleting an inserted line amounts to do nothing).*

**Example 2.3** The patch corresponding to transforming the file *abc* into *dadeb*, by deleting the line *c* and inserting the lines labeled by *d* and *e*, is modeled by the partial function *f* : [3] *→* [5] such that *f* (0) = 1 and *f* (1) = 4 and *f* (2) is undefined. Graphically,

*a d*

*b a*

*c d*

*e*

*b*

The deleted line is the one on which *f* is not defined and the inserted lines are those which are not in the image of *f* . In other words, *f* keeps track of the unchanged lines.

In order to increase readability, we shall consider the particular case where *L* is reduced to a single element. In this *unlabeled* case, the objects of *L* can be identified with integers (the labeling function is trivial), and Proposition [2.2](#_bookmark2) can be adapted to achieve the following description of the category, see also [[6](#_bookmark30)].

**Proposition 2.4** *If L is reduced to a singleton, the category L is the free category*

*whose objects are integers and morphisms are generated by s*n

i

: *n → n* +1 *and*

n : *n* +1 *→ n for every n ∈* N *and i ∈* [*n* + 1] *(respectively corresponding to*

*d*

i

*insertion and deletion of a line at i-th position), subject to the relations*

*s*n+1*s*n = *s*n+1*s*n

*d*n*s*n = idn *d*n*d*n+1 = *d*n*d*n+1

(1)

i j j+1 i i i

i j j

i+1

*whenever* 0 *≤ i ≤ j < n.*

We will also consider the subcategory *L*+ of *L*, with same objects, and *total* injective increasing functions as morphisms. This category models patches where the only possible operation is the insertion of lines: Proposition [2.2](#_bookmark2) can be adapted to show that *L*+ is the free monoidal category containing morphisms *η*a : *I → a* and, in the unlabeled case, Proposition [2.4](#_bookmark3) can be similarly adapted to show that it is the free category generated by morphisms *s*n : *n → n* + 1 satisfying *s*n+1*s*n = *s*n+1*s*n with

i

0 *≤ i ≤ j < n*.

i j j+1 i

# Towards a category of conflicting files

Suppose that *A* is a file which is edited by two users, respectively applying patches

*f*1 : *A → A*1 and *f*2 : *A → A*2 to the file. For instance,

*a c c b*

f1

*→− a b*

f2

*−→ a b c d* (2)

Now, each of the two users imports the modification from the other one. The result- ing file, after the import, should be the smallest file containing both modifications on the original file: *accbcd*. It is thus natural to state that it should be a pushout of the diagram ([2](#_bookmark5)). Now, it can be noticed that not every diagram in *L* has a pushout. For instance, the diagram

*a c b*

f1

*→− a b*

f2

*−→ a d b* (3)

does not admit a pushout in *L*. In this case, the two patches *f*1 and *f*2 are said to be *conflicting*.

In order to represent the state of files after applying two conflicting patches, we investigate the definition of a category *P* which is obtained by completing the category *L* under all pushouts. Since, this completion should also contain an initial object (i.e. the empty file), we are actually defining the category *P* as the free completion of *L* under finite colimits: recall that a category is finitely cocomplete (has all finite colimits) if and only if it has an initial object and is closed under

pushouts [[6](#_bookmark30)]. Intuitively, this category is obtained by adding files whose lines are not linearly ordered, but only partially ordered, such as on the left of

*a*

*c* z*d z*

z *z*

*b*

a

<<<<<<< HEAD

c

=======

d

>>>>>>> 5c55...

b

(4)

which would intuitively model the pushout of the diagram ([3](#_bookmark6)) if it existed, indi- cating that the user has to choose between *c* and *d* for the second line. Notice the similarities with the corresponding textual notation in git on the right. The name of the category *L* reflects the facts that its objects are files whose lines are linearly ordered, whereas the objects of *P* can be thought as files whose lines are only partially ordered. More formally, the category is defined as follows.

**Definition 3.1** The category *P* is the *free ﬁnite conservative cocompletion* of *L*: it is (up to equivalence of categories) the unique finitely cocomplete category together with an embedding functor *y* : *L→P* preserving finite colimits, such that for every finitely cocomplete category *C* and functor *F* : *L → C* preserving finite colimits,

there exists, up to unique isomorphism, a unique functor *F*˜

finite colimits and satisfying *F*˜ *◦ y* = *F* :

: *P → C* preserving

*P*

*C ¸,*

F

y J˜

*L*  F

Above, the term *conservative* refers to the fact that we preserve colimits which already exist in *L* (we will only consider such completions here). The “standard” way to characterize the category *P*, which always exists, is to use the following folklore theorem, often attributed to Kelly [[5](#_bookmark31),[1](#_bookmark26)]:

**Theorem 3.2** *The conservative cocompletion of the category L is equivalent to the full subcategory of L*ˆ *whose objects are presheaves which preserve ﬁnite limits,*

*i.e. the image of a limit in L*op *(or equivalently a colimit in L) is a limit in* **Set** *(and limiting cones are transported to limiting cones). The ﬁnite conservative co- completion P can be obtained by further restricting to presheaves which are ﬁnite colimits of representables.*

**Example 3.3** The category **FinSet** of finite sets and functions is the conservative cocompletion of the terminal category **1**.

We recall that the category *L*ˆ of *presheaves* over *L*, is the category of functors *L*op *→* **Set** and natural transformations between them. The *Yoneda functor y* : *L→ L*ˆ defined on objects *n ∈ L* by *yn* = *L*(*−, n*), and on morphisms by postcomposition,

provides a full and faithful embedding of *L* into the corresponding presheaf cate- gory, and can be shown to corestrict into a functor *y* : *L→P* [[1](#_bookmark26)]. A presheaf of the form *yn* for some *n ∈L* is called *representable*.

Extracting a concrete description of the category *P* from the above proposition is a challenging task, because we a priori need to characterize firstly all diagrams admitting a colimit in *L*, and secondly all presheaves in *L*ˆ which preserve those diagrams. This paper introduces a general methodology to build such a category.

In particular, perhaps a bit surprisingly, it turns out that we have to “allow cycles” in the objects of the category *P*, which will be described as *the category whose objects are ﬁnite sets labeled by lines together with a transitive relation and morphisms are partial functions respecting labels and relations*.

# A cocompletion of files and insertions of lines

In order to make our presentation clearer, we shall begin our investigation of the category *P* in a simpler case, which will be generalized in Section [6](#_bookmark23): we compute the free finite cocompletion of the category *L*+ (patches can only insert lines) in the case where the set of labels is a singleton. *To further lighten notations, in this section, we simply write L for this category*.

We sometimes characterize the objects in *L* as finite colimits of objects in a subcategory *G* of *L*. This category *G* is the full subcategory of *L* whose objects are1 and 2: it is the free category on the graph 1 2 ,*¸*the two arrows being *s*1 and *s*1.

0 1

The category *G*ˆ of presheaves over *G* is the category of *graphs*: a presheaf *P ∈ G*ˆ is a

graph with *P* (1) as vertices, *P* (2) as edges, the functions *P* (*s*1) and *P* (*s*1) associate

1 0

to a vertex its source and target respectively, and morphisms correspond to usual

morphisms of graphs. We denote by *x* → *y* a path going from a vertex *x* to a vertex *y* in such a graph. The inclusion functor *I* : *G → L* induces, by precomposition, a functor *I∗* : *L*ˆ *→ G*ˆ. The image of a presheaf in *L*ˆ under this functor is called its *underlying graph*. By well known results about presheaves categories, this functor admits a right adjoint *I∗* : *G*ˆ *→ L*ˆ: given a graph *G ∈ G*ˆ, its image under the right adjoint is the presheaf *G∗ ∈ L*ˆ such that for every *n ∈* N, *G∗*(*n* + 1) is the set of paths of length *n* in the graph *G*, with the expected source maps, and *G∗*(0) is reduced to one element.

Recall that every functor *F* : *C → D* induces a *nerve functor N*F : *D → C*ˆ defined

on an object *A ∈ C* by *N*F (*A*) = *D*(*F−, A*) [[7](#_bookmark32)]. Here, we will consider the nerve *N*I : *L → G*ˆ associated to the inclusion functor *I* : *G → L*. An easy computation shows that the image *N*I(*n*) of *n ∈L* is a graph with *n* vertices, so that its objects are isomorphic to [*n*], and there is an arrow *i → j* for every *i, j ∈* [*n*] such that

*i < j*. For instance,

*N*I(3) = 0 1 *¸* 2 *¸*

*N*I(4) = 0 1 *¸* v2 *¸˛*v 3 *˛¸¸*

It is, therefore, easy to check that this embedding is full and faithful, i.e. morphisms in *L* correspond to natural transformations in *G*ˆ. Moreover, since *N*I(1) is the graph reduced to a vertex and *N*I(2) is the graph reduced to two vertices and one arrow between them, every graph can be obtained as a finite colimit of the graphs *N*I(1)

and *N*I(2) by “gluing arrows along vertices”. For instance, the initial graph *N*I(0) is

the colimit of the empty diagram, and the graph *N*I(3) is the colimit of the diagram

N*I* (s1) *N¸*I (2) ¸N*I¸*(s0) N*I* (s1) *N¸*I (2) ¸N*I¸*(s0)

*N*I(1)

*N*I(1)

z*N* (2) ¸*7* N*I* (s0)

N*I* (s1)

I

*N*I(1)

which may also be drawn as on the left of

2 ,*¸* 2 ,*¸*

1 1 1



*,*¸*¸* ,*¸*

z *c*

z2

by drawing the graphs *N*I(0) and *N*I(1). Notice, that the object 3 is the colimit of the corresponding diagram in *L* (on the right), and this is generally true for all

objects of *L*, moreover this diagram is described by the functor El(*N*I(3)) *−→*π *L*.

The notation El(*P* ) refers to the *category of elements* of a presheaf *P ∈ C*ˆ, whose

objects are pairs (*A, p*) with *A ∈C* and *p ∈ P* (*A*) and morphisms *f* : (*A, p*) *→* (*B, q*) are morphisms *f* : *A → B* in *C* such that *P* (*f* )(*q*)= *p*, and *π* is the first projection functor. The functor *I* : *G → L* is thus a dense functor in the sense of Definition [4.2](#_bookmark10) below, see [[7](#_bookmark32)] for details.

**Proposition 4.1** *Given a functor F* : *C → D, with D cocomplete, the associated nerve N*F : *D → C*ˆ *admits a left adjoint R*F : *C*ˆ *→ D called the* realization *along F. This functor is deﬁned on objects P ∈ C*ˆ *by*

*R*F (*P* ) = colim(El(*P* ) *−→*π *C −→*F *D*)

**Proof** Given a presheaf *P ∈ C*ˆ and an object *D*, it can be checked directly that

*−→*

morphisms *P → N*F

*D* in *C*ˆ with cocones from El(*P* ) *D*

to *D*, which in turn are in

bijection with morphisms *R*F (*P* ) *→ D* in *D*, see [[7](#_bookmark32)]. *2*

**Definition 4.2** A functor *F* : *C → D* is *dense* if it satisfies one of the two equivalent conditions:

1. the associated nerve functor *N*F : *D → C*ˆ is full and faithful,
2. every object of *D* is canonically a colimit of objects in *C*: for every *D ∈ D*,

*D ∼*= colim(El(*N*F *D*) *−→*π

*C −→*F

1. (5)

Since the functor *I* is dense, every object of *L* is a finite colimit of objects in *G*, and *G* does not have any non-trivial colimit. One could expect the free conservative finite cocompletion of *L* to be the free finite cocompletion *P* of *G*. We will see that this is not the case because the image in *L* of a non-trivial diagram in *G* might still

have a colimit. By Theorem [3.2](#_bookmark7), the category *P* is the full subcategory of *L*ˆ of

presheaves preserving limits, which we now describe explicitly. This category will

turn out to be equivalent to a full subcategory of *G*ˆ (Theorem [4.8](#_bookmark19)). We should first remark that those presheaves satisfy the following properties:

**Proposition 4.3** *Given a presheaf P ∈ L*ˆ *which is an object of P,*

* 1. *the underlying graph of P is ﬁnite,*
  2. *for each non-empty path x* → *y there exists exactly one edge x → y (in partic- ular there is at most one edge between two vertices),*
  3. *P* (*n* + 1) *is the set of paths of length n in the underlying graph of P, and P* (0)

*is reduced to one element.*

**Proof** We suppose given a presheaf *P ∈ P*, it preserves limits by Theorem [3.2](#_bookmark7). The diagram on the left

s2 3 ,s2*¸*

P (s2) *P* (3) P (s2)

2 0

2 ,*¸* 2 *,*

2

*P* (2)

z0

*P* (2)

s1 1 s1

P (s1) z

P (s1)

0 1 0

*P* (1) 1

is a pushout in *L*, or equivalently the dual diagram is a pullback in *L*op. There-

s1 s1

fore, writing *D* for the diagram 2 ¸0 *,*1 1 2 *¸*in *L*, a presheaf *P ∈ P* should satisfy

*P* ((colim *D*)op) *∼*= lim *P* (*D*op), i.e. the above pushout diagram in *L* should be trans-

ported by *P* into the pullback diagram in **Set** depicted on the right of the above figure. This condition can be summarized by saying that *P* should satisfy the iso- morphism *P* (3) *∼*= *P* (2) *×*P (1) *P* (2) (and this isomorphism should respect obvious source and target maps given by the fact that the functor *P* should send a limiting cone to a limiting cone). From this fact, one can deduce that the elements *α* of *P* (3) are in bijection with the paths *x → y → z* of length 2 in the underlying graph

of *P* going from *x* = *P* (*s*2*s*1)(*α*) to *z* = *P* (*s*2*s*1)(*α*). In particular, this implies that

2 1 0 0

for any path *α* = *x → y → z* of length 2 in the underlying graph of *P* , there exists

an edge *x → z*, which is *P* (*s*2)(*α*). More generally, given any integer *n >* 1, the object *n* + 1 is the colimit in *L* of the diagram

1

s1 2 ,s*¸*1 s1

2 ,s1*¸*

"2 ,s*¸*1 s1 2 ,s*¸*1

(6)

1 0 1

1 1

0

*...*

0 1 0

1 1

s1

1

with *n* + 1 occurrences of the object 1, and *n* occurrences of the object 2. Therefore, for every *n ∈* N, *P* (*n* + 1) is isomorphic to the set of paths of length *n* in the underlying graph. Moreover, since the diagram

s1 2 ,s1*¸*s1

2 ,s1*¸*

s1 "2 ,s1*¸*s1

2 ,s1*¸*

(7)

1 0 1

1 1

0 1 0 1 0

*...* 1 1

1 z2 ¸ *7* 1

s

s

1

0

with *n* + 1 occurrences of the object 1 also admits the object *n* + 1 as colimit, we should have *P* (*n* + 1) *∼*= *P* (*n* + 1) *× P* (2) between any two vertices *x* and *y*, i.e. for every non-empty path *x* → *y* there exists exactly one edge *x → y*. Also, since the

object 0 is initial in *L*, it is the colimit of the empty diagram. The set *P* (0) should thus be the terminal set, i.e. reduced to one element. Finally, since *I* is dense, *P* should be a finite colimit of the representables *N*I(1) and *N*I(2), the set *P* (1) is necessarily finite, as well as the set *P* (2) since there is at most one edge between two vertices. *2*

Conversely, we wish to show that the conditions mentioned in the above propo- sition exactly characterize the presheaves in *P* among those in *L*ˆ. In order to prove so, by Theorem [3.2](#_bookmark7), we have to show that presheaves *P* satisfying these conditions preserve finite limits in *L*, i.e. that for every finite diagram *D* : *J → L* admitting a colimit we have *P* (colim *D*) *∼*= lim(*P ◦ D*op). It seems quite difficult to characterize the diagrams admitting a colimit in *L*, however the following lemma shows that it

is enough to check diagrams “generated” by a graph which admits a colimit.

**Lemma 4.4** *A presheaf P ∈ L*ˆ *preserves ﬁnite limits if and only if it sends the colimits of diagrams of the form*

El(*G*) π*G* I

(8)

*−−→G −→L*

*to limits in* **Set***, where G ∈ G*ˆ *is a ﬁnite graph such that the above diagram admits a colimit. Such a diagram in L is said to be* generated by the graph *G.*

**Proof** In order to check that a presheaf *P ∈ L*ˆ preserves finite limits, we have to check that it sends colimits of finite diagrams in *L which admit a colimit* to limits in **Set**, and therefore we have to characterize diagrams which admit colimits in *L*. Suppose given a diagram *K* : *J → L*. Since *I* is dense, every object of linear is a colimit of a diagram involving only the objects 1 and 2 (see Definition [4.2](#_bookmark10)). We can therefore suppose that this is the case in the diagram *K*. Finally, it can be shown

that diagram *K* admits the same colimits as a diagram containing only *s*1 and *s*1

0 1

as arrows (these are the only non-trivial arrows in *L* whose source and target are 1

or 2), in which every object 2 is the target of exactly one arrow *s*1 and one arrow *s*1. For instance, the diagram in *L* below on the left admits the same colimits as the diagram in the middle.

0

1

2 s2s1 3 2 1

s1 2 s1

s1 2 s1

s1 2 s1

,*¸*2 1 ,s*¸*0s0 1 ,0*¸* 1 ,0*¸* 1 ,0*¸*

1

s

0 1 1 1

1 v *z* 1 s1

1 1 1 1

1

s1

1

0  1 *¸* 2 *¸* 3 *¸*

s1 2 ¸s*¸*0

0

z2s0

Any such diagram *K* is obtained by gluing a finite number of diagrams of the form

s1 s1 π I

1 1 2 ¸0 *,*1 along objects 1, and is therefore of the form El(*G*) *−→ G −→ L* for some

finite graph *G ∈ G*ˆ: the objects of *G* are the objects 1 in *K*, the edges of *G* are

the objects 2 in *K* and the source and target of an edge 2 are respectively given by the sources of the corresponding arrows *s*1 and *s*1 admitting it as target. For

1 0

instance, the diagram in the middle above is generated by the graph on the right.

The fact that every diagram is generated by a presheaf (is a discrete fibration) also

follows more abstractly and generally from the construction of the comprehensive factorization system on **Cat** [[8](#_bookmark34),[11](#_bookmark36)]. *2*

Among diagrams generated by graphs, those admitting a colimit can be charac- terized using the following proposition:

**Lemma 4.5** *Given a graph G ∈ G*ˆ*, the associated diagram* ([8](#_bookmark14)) *admits a colimit in L if and only if there exists n ∈L and a morphism f* : *G → N*I *n in L*ˆ *such that* *every morphism g* : *G → N*I *m in L*ˆ*, with m ∈ L, factorizes uniquely through N*I *n:*

*G*  f *N¸n* ˛*N¸$m*

I I

g

**Proof** Follows from the existence of a partially defined left adjoint to *N*I, in the sense of [[8](#_bookmark34)], given by the fact that *I* is dense (see Definition [4.2](#_bookmark10)). *2*

We finally arrive at the following concrete characterization of diagrams admitting colimits:

**Lemma 4.6** *A ﬁnite graph G ∈*

*G*ˆ *induces a diagram* ([8](#_bookmark14)) *in L which admits a*

*colimit if and only if it is “tree-shaped”, i.e. it is*

1. *acyclic: for any vertex x, the only path x* → *x is the empty path,*
2. *connected: for any pair of vertices x and y there exists a path x* → *y or a path*

*y* → *x.*

**Proof** Given an object *n ∈ L*, recall that *N*I *n* is the graph whose objects are elements of [*n*] and there is an arrow *i → j* if and only if *i < j*. Given a finite graph *G*, morphisms *f* : *G → N*I *n* are therefore in bijection with functions *f* : *V*G *→* [*n*], where *V*G denotes the set of vertices of *G*, such that *f* (*x*) *< f* (*y*) whenever there exists an edge *x → y* (or equivalently, there exists a non-empty path *x* → *y*).

Consider a finite graph *G ∈ G*ˆ, by Lemma [4.5](#_bookmark16), it induces a diagram ([8](#_bookmark14)) admitting

a colimit if there is a universal arrow *f* : *G → N*I *n* with *n ∈ L*. From this it follows that the graph is acyclic: otherwise, we would have a non-empty path *x* → *x* for some vertex *x*, which would imply *f* (*x*) *< f* (*x*). Similarly, suppose that *G* is a graph with vertices *x* and *y* such that there is no path *x* → *y* or *y* → *x*, and there is an universal morphism *f* : *G → N*I *n* for some *n ∈ L*. Suppose that *f* (*x*) *≤ f* (*y*) (the case where *f* (*y*) *≤ f* (*x*) is similar). We can define a morphism *g* : *G → N*I(*n* + 1) by *g*(*z*)= *f* (*z*)+1 if there isa path *x* → *z*, *g*(*y*)= *f* (*x*) and *g*(*z*)= *f* (*z*) otherwise. This morphism is easily checked to be well-defined. Since we always have *f* (*x*) *≤ f* (*y*) and *g*(*x*) *> g*(*y*), there is no morphism *h* : *N*I *n → N*I(*n* + 1) such that *h ◦ f* = *g*.

Conversely, given a finite acyclic connected graph *G*, the relation *≤* defined on morphisms by *x ≤ y* whenever there exists a path *x* → *y* is a total order. Writing *n* for the number of vertices in *G*, the function *f* : *G → N*I *n*, which to a vertex associates the number of vertices strictly below it wrt *≤*, is universal in the sense of Lemma [4.5](#_bookmark16). *2*

**Proposition 4.7** *The free conservative ﬁnite cocompletion P of L is equivalent to the full subcategory of L*ˆ *whose objects are presheaves P satisfying the conditions of Proposition* [*4.3*](#_bookmark11)*.*

**Proof** By Lemma [4.4](#_bookmark15), the category *P* is equivalent to the full subcategory of *L*ˆ whose objects are presheaves preserving limits of diagrams of the form ([8](#_bookmark14)) generated by some graph *G ∈ G*ˆ which admits a colimit, i.e. by Lemma [4.6](#_bookmark17) the finite graphs

which are acyclic and connected. We write *G*n for the graph with [*n*] as vertices and edges *i →* (*i* + 1) for 0 *≤ i < n −* 1. It can be shown that any acyclic and connected finite graph can be obtained from the graph *G*n, for some *n ∈* N, by iteratively adding an edge *x → y* for some vertices *x* and *y* such that there exists a non-empty path *x* → *y*. Namely, suppose given an acyclic and connected finite graph *G*. The relation *≤* on its vertices, defined by *x ≤ y* whenever there exists a path *x* → *y*, is a total order, and therefore the graph *G* contains *G*n, where *n* is the number of edges of *G*. An edge in *G* which is not in *G*n is necessarily of the form *x → y* with *x ≤ y*, otherwise it would not be acyclic. Since by Proposition [4.3](#_bookmark11), see ([7](#_bookmark13)), the diagram generated by a graph of the form



*...*

is preserved by presheaves in *P* (which corresponds to adding an edge between vertices at the source and target of a non-empty path), it is enough to show that presheaves in *P* preserve diagrams generated by graphs *G*n. This follows again by Proposition [4.3](#_bookmark11), see ([6](#_bookmark12)). *2*

One can notice that a presheaf *P ∈ P* is characterized by its underlying graph since *P* (0) is reduced to one element and *P* (*n*) with *n >* 2 is the set of paths of length *n* in this underlying graph: *P ∼*= *I∗*(*I∗P* ). We can therefore simplify the description of the cocompletion of *L* as follows:

**Theorem 4.8** *The free conservative ﬁnite cocompletion P of L is equivalent to the full subcategory of the category G*ˆ *of graphs, whose objects are ﬁnite graphs such that for every non-empty path x* → *y there exists exactly one edge x → y. Equivalently,*

*it can be described as the category whose objects are ﬁnite sets equipped with a transitive relation <, and functions respecting relations.*

In this category, pushouts can be explicitly described as follows:

**Proposition 4.9** *With the last above description, the pushout of a diagram in P*

f g

(*B, <*B) *→−* (*A, <*A) *−→* (*C, <*C) *is B C/ ∼ with B e b ∼ c ∈ C whenever there*

*exists a ∈ A with f* (*a*)= *b and f* (*a*)= *c, equipped with the transitive closure of the relation inherited by <*B *and <*C*.*

*Lines with labels.* The construction can be extended to the labeled case (i.e. *L* is not necessarily a singleton). The forgetful functor *L*ˆ *→* **Set** sending a presheaf *P* to the set *P* (1) admitsa right adjoint! : **Set** *→ L*ˆ. Given *n ∈* N*∗* the elements of !*L*(*n*) are words *u* of length *n* over *L*, with !*L*(*s*n*−*1)(*u*) being the word obtained from *u* by removing the *i*-th letter. The free conservative finite cocompletion *P* of *L* is the slice category *L/*!*L*, whose objects are pairs (*P, l*) consisting of a finite presheaf *P ∈ L*ˆ together with a *labeling* morphism *l* : *P →* !*L* of presheaves. Alternatively, the description of Proposition [4.8](#_bookmark19) can be straightforwardly adapted by labeling the

i

elements of the objects by elements of *L* (labels should be preserved by morphisms), thus justifying the use of labels for the vertices in following examples.

# Examples

In this section, we give some examples of merging (i.e. pushout) of patches.

**Example 5.1** Suppose that starting from a file *ab*, one user inserts a line *a∗* at the beginning and *c* in the middle, while another one inserts a line *d* in the middle. After merging the two patches, the resulting file is the pushout of

*a∗*

* 1. f1 *a*

*c →−*

* 1. *b*

f2 *a*

*−→ d*

*b*

which is *c d*

**Example 5.2** Write *G*1 for the graph with one vertex and no edges, and *G*2 for the graph with two vertices and one edge between them. We write *s, t* : *G*1 *→ G*2 for the two morphisms in *P*. Since *P* is finitely cocomplete, there is a coproduct *G*1 + *G*1 which gives, by universal property, an arrow seq : *G*1 + *G*1 *→ G*2:



*a∗*

*a*

*b*

s %*G*,*%*2 ,*¸*t

,,*,¸*

s t

*¸*¸*,*

seq

seq

*G*1 *G¸*1 + *G*1 ¸*,G*1

or graphically

that we call the *sequentialization morphism*. This morphism corresponds to the following patch: given two possibilities for a line, a user can decide to turn them into two consecutive lines. We also write seq*∗* : *G*1 + *G*1 *→ G*2 for the morphism obtained similarly by exchanging *s* and *t* in the above cocone. Now, the pushout of

seq

*→−−*

seq*′*

*−−→ * is

which illustrates how cyclic graphs appear in *P* during the cocompletion of *L*.

**Example 5.3** With the notations of the previous example, by taking the coproduct of two copies of idG1 : *G*1 *→ G*1, there is a universal morphism *G*1 + *G*1 *→ G*1, which illustrates how two independent lines can be merged by a patch (in order to resolve conflicts).

id*•*

,,*,,¸*

id*•*

*¸*¸*,*

merge

# Handling deletions of lines

All the steps performed in previous sections in order to compute the free conservative finite cocompletion of the category *L*+ can be adapted in order to compute the cocompletion *P* of the category *L* as introduced in Definition [2.1](#_bookmark1), thus adding

support for deletion of lines in patches. In particular, the generalization of the description given by Theorem [4.8](#_bookmark19) turns out to be as follows.

**Theorem 6.1** *The free conservative ﬁnite cocompletion P of the category L is the category whose objects are triples* (*A, <, l*) *where A is a ﬁnite set of lines, < is a transitive relation on A and l* : *A → L associates a label to each line, and morphisms ƒ* : (*A, <*A*, l*A) *→* (*B, <*B*, l*B) *are partial functions ƒ* : *A → B such that for every a, a∗ ∈ A both admitting an image under ƒ, we have l*B(*ƒ* (*a*)) = *l*A(*a*)*, and a <*A *a∗ implies ƒ* (*a*) *<*B *ƒ* (*a∗*)*.*

Similarly, pushouts in this category can be computed as described in Proposition [4.9](#_bookmark20), generalized in the obvious way to partial functions.

**Example 6.2** Suppose that starting from a file *abc*, one user inserts a line *d* after *a* and the other one deletes the line *b*. The merging of the two patches (in *P∗*) is the pushout of

* 1. *a a*

f1 f2

*d*

*→− −→*

* 1. *b*
  2. *c c*

*a*

which is *d c*

i.e. the file *adc*. Notice that the morphism *ƒ*2 is partial: *b* has no image.

Interestingly, a category very similar to the one we have described in Theorem [6.1](#_bookmark24) was independently proposed by Houston [[3](#_bookmark27)] based on a construction performed in [[2](#_bookmark28)] for modeling asynchronous processes. This category is not equivalent to ours because morphisms are reversed partial functions: it is thus not the most general model (in the sense of being the free finite cocompletion). As a simplified explanation for this, consider the category **FinSet** which is the finite cocompletion of **1**. This category is finitely complete (in addition to cocomplete), thus **FinSet**op is finitely cocomplete and **1** embeds fully and faithfully in it. However, **FinSet**op is not the finite cocompletion of **1**. Another way to see this is that this category does not contain the “merging” morphism of Example [5.3](#_bookmark22), but it contains a dual morphism “duplicating” lines.

# Concluding remarks and future works

In this paper, we have detailed how we could derive from universal constructions a category which suitably models files resulting from conflicting modifications. It is finitely cocomplete, thus the merging of any modifications of the file is well-defined. We believe that the interest of our methodology lies in the fact that it adapts easily to other more complicated base categories *L* than the two investigated here: in future works, we should explain how to extend the model in order to cope with multiple files (which can be moved, deleted, etc.), different file types (containing text, or more structured data such as xml trees). Also, the structure of repositories (partially ordered sets of patches) is naturally modeled by event structures labeled by morphisms in *P*, which will be detailed in future works, as well as how to

model usual operations on repositories: cherry-picking (importing only one patch from another repository), using branches, removing a patch, etc. It would also be interesting to explore axiomatically the addition of inverses for patches, following other works hinted at in the introduction.

Once the theoretical setting is clearly established, we plan to investigate algorith- mic issues (in particular, how to efficiently represent and manipulate the conflicting files, which are objects in *P*). This should eventually serve as a basis for the imple- mentation of a theoretically sound and complete distributed version control system (no unhandled corner-cases as in most current implementations of vcs).

# Acknowledgement

The authors would like to thank P.-A. Melli`es, E. Haucourt, T. Heindel, T. Hirschowitz and the anonymous reviewers for their enlightening comments and suggestions.

# References

1. J. Ad´amek and J. Rosicky. *Locally presentable and accessible categories*, volume 189. Cambridge Univ. Press, 1994.
2. R. Cockett and D. Spooner. Categories for synchrony and asynchrony. *Electronic Notes in Theoretical* *Computer Science*, 1:66–90, 1995.
3. R. Houston. *On editing text*. <http://bosker.wordpress.com/2012/05/10/on-editing-text>.
4. J. Jacobson. A formalization of darcs patch theory using inverse semigroups. Technical report, CAM report 09-83, UCLA, 2009.
5. M. Kelly. *Basic concepts of enriched category theory*, volume 64. Cambridge Univ. Press, 1982.
6. S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer Verlag, 1971.
7. S. Mac Lane and I. Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer, 1992.
8. R. Par´e. Connected components and colimits. *Journal of Pure and Applied Algebra*, 3(1):21–42, 1973.
9. M. Ressel, D. Nitsche-Ruhland, and R. Gunzenha¨user. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 288–297. ACM, 1996.
10. D. Roundy and al. *The Darcs Theory*. <http://darcs.net/Theory>.
11. R. Street and R. Walters. The comprehensive factorization of a functor. *Bull. Amer. Math. Soc*, 79(2):936–941, 1973.

# A geometric interpretation of presheaves on *L*+

Since presheaf categories are sometimes a bit difficult to grasp, we recall here the geometric interpretation that can be done for presheaves in *L*ˆ+. We forget about labels of lines and for simplicity suppose that the empty file is not allowed (the

objects are strictly positive integers). In this section, we denote this category by *L*. The same reasoning can be performed on the usual category *L*+, and even *L*, but the geometrical explanation is a bit more involved to describe.

In this case, the presheaves in *L*ˆ can easily be described in geometrical terms: the elements *P* of *L*ˆ are *presimplicial sets*. Recall from Proposition [2.4](#_bookmark3) that the

category *L* is the free category whose objects are strictly positive natural integers, containing for every integers *n ∈* N*∗* and *i ∈* [*n*+1] morphisms *s*n : *n → n*+1, subject

i

to the relations *s*n+1*s*n = *s*n+1*s*n whenever 0 *≤ i ≤ j < n*. Writing *y* : *L → L*ˆ for

i j j+1 i

the Yoneda embedding, a representable presheaf *y*(*n* + 1) *∈ L*ˆfin can be pictured

+

geometrically as an *n*-simplex: a 0-simplex is a point, a 1-simplex is a segment, a 2-simplex is a (filled) triangle, a 3-simplex is a (filled) tetrahedron, etc.:

*y*(1) *y*(2) *y*(3) *y*(4)

Notice that the *n*-simplex has *n* faces which are (*n −* 1)-dimensional simplices, and these are given by the image under *y*(*s*n), with *i ∈* [*n*], of the unique element of *y*(*n*+1)(*n*+1): the *i*-th face of an *n*-simplex is the (*n−*1)-simplex obtained by remov- ing the *i*-th vertex from the simplex. More generally, a

i

presheaf *P ∈ L*ˆfin (a finite presimplicial set) is a finite colimit of rep-

+

resentables: every such presheaf can be pictured as a gluing of sim- plices. For instance, the half-filled square on the right corresponds to the presimplicial set *P* with *P* (1) = *{a, b, c, d}*, *P* (2) = *{ƒ, g, h, i, j}*,

*P* (3) = *{α}* with faces *P* (*s*1)(*ƒ* )= *a*, *P* (*s*1)(*ƒ* )= *b*, etc.

*b g c*

*ƒ jα h*

*a i d*

1 0

Similarly, in the labeled case, a labeled presheaf (*P, l*) *∈ L/*! can be pictured as a

presimplicial set whose vertices (0-simplices) are labeled by elements of *L*. The word labeling of higher-dimensional simplices can then be

*ab*

*a*

abc *ac*

*bc c*

deduced by concatenating the labels of the vertices it has as iterated

faces. For instance, an edge (a 1-simplex) whose source is labeled by *a b*

and target is labeled by *b* is necessarily labeled by the word *ab*, etc.

More generally, presheaves in *L*+ can be pictured as *augmented presimplicial sets* and presheaves in *L* as *augmented simplicial sets*, a description of those can for instance be found in Hatcher’s book *Algebraic Topology*.

# Proofs of classical propositions

In this section, we briefly recall proofs of well-known propositions as our proofs rely on a fine understanding of those. We refer the reader to [[7](#_bookmark32)] for further details.

**Proposition** [**4.1**](#_bookmark9)*Given a functor F* : *C → D, with D cocomplete, the associated nerve N*F : *D → C*ˆ *admits a left adjoint R*F : *C*ˆ *→ D called the* realization *along F. This functor is deﬁned on objects P ∈ C*ˆ *by*

*R*F (*P* ) = colim(El(*P* ) *−→*π *C −→*F *D*)

**Proof** In order to show the adjunction, we have to construct a natural family of isormorphisms *D*(*R*F (*P* )*, D*) *∼*= *C*ˆ(*P, N*F *D*) indexed by a presheaf *P ∈ C*ˆ and an object *D ∈ D*. A natural transformation *θ ∈ C*ˆ(*P, N*F *D*) is a family of functions (*θ*C : *PC → N*F *DC*)C*∈C* such that for every morphism *ƒ* : *C∗ → C* in *C* the diagram

*P* (*C*) θ*C*  *D¸*(*F C, D*)

P (f) J

*P* (*C∗*)

θ*C′*

J*D*(Ff,D)

*D¸*(*FC∗, D*)

commutes. It can also be seen as a family (*θ*C(*p*): *FC → D*)(C,p)*∈*El(P ) of morphisms in *D* such that the diagram

z*'*

*F C*,*,*θ*C* (p)

*D*

Ff *,*

or equivalently

*F π*P (,*C,,p*)

F π*P* f

θ*C′* (p*′*)

θ*C* (p)

v*˛*

*D*

*FC∗*θ*C′* (P (f)(p))

*F π*P (*C∗, p∗*)

commutes for every morphism *ƒ* : *C∗ → C* in *C*. This thus defines a cocone from *F π*P : El(*P* ) *→ D* to *D*, and those cocones are in bijection with morphisms *R*F (*P* ) *→ D* by definition of *R*F (*P* ) as a colimit: we have shown *D*(*R*F (*P* )*, D*) *∼*= *C*ˆ(*P, N*F (*D*)), from which we conclude. *2*

The equivalence between the two conditions of Definition [4.2](#_bookmark10) can be shown as follows.

**Proposition B.1** *Given a functor F* : *C → D, the two following conditions are equivalent:*

* 1. *the associated nerve functor N*F : *D → C*ˆ *is full and faithful,*
  2. *every object of D is canonically a colimit of objects in C: for every D ∈ D,*

*D ∼*= colim(El(*N*F *D*) *−→*π *C −→*F *D*)

**Proof** In the case where *D* is cocomplete, the nerve functor *N*F : *D →*

*C*ˆ ad-

mits *R*F :

*C*ˆ *→ D* as right adjoint, and the equivalence amounts to showing

that the right adjoint is full and faithful if and only if the counit is an isomor- phism, which is a classical theorem [[6](#_bookmark30), Theorem IV.3.1]. The construction can be adapted to the general case where *D* is not necessarily cocomplete by considering

colim(El(*−*) *−→*π

*C −→*F

*D*) : *C*ˆ *→ D* as a partially defined left adjoint (see [[8](#_bookmark34)]) and

generalizing the theorem. *2*

# Proofs of the construction of the finite cocompletion

**Lemma** [**4.4**](#_bookmark15)*A presheaf P ∈ L*ˆ *preserves ﬁnite limits, if and only if it sends the colimits of diagrams of the form*

El(*G*) π*G* I

*−−→G −→L*

*to limits in* **Set***, where G ∈ G*ˆ *is a ﬁnite graph such that the above diagram admits a colimit. Such a diagram in L is said to be* generated by the graph *G.*

**Proof** In order to check that a presheaf *P ∈ L*ˆ preserves finite limits, we have to check that it sends colimits of finite diagrams in *L which admit a colimit* to limits in **Set**, and therefore we have to characterize diagrams which admit colimits in *L*. The number of diagrams to check can be reduced by using the facts that limits commute with limits [[6](#_bookmark30)]. For instance, the inclusion functor *I* : *G → L* is dense, which implies that every object *n ∈L* is canonically a colimit of the objects 1 and 2

by the formula *n ∼*= colim(El(*N*I *n*) *−→*π

*G −→*I

*L*), see Definition [4.2](#_bookmark10). Thus, given a

finite diagram *K* : *J → L*, we can replace any object *n* different from 1 and 2

occurring in the diagram by the corresponding diagram El(*N*I *n*) *−→*π

*G −→*I

*L*, thus

obtaining a new diagram *K∗* : *J → L* which admits the same colimit as *K*. This shows that *P* will preserve finite limits if and only if it preserves limits of finite diagrams in *L* in which the only occurring objects are 1 and 2. Since the only

non-trivial arrows in *L* between the objects 1 and 2 are *s*1*, s*1 :1 *→* 2, and removing

0 1

an identity arrow in a diagram does not change its colimit, the diagram *K* can thus

be assimilated to a bipartite graph with vertices labeled by 1 or 2 and edges labeled by *s*1 or *s*1, all edges going from vertices 1 to vertices 2.

0 1

We can also reduce the number diagrams to check by remarking that some pairs

of diagrams are “equivalent” in the sense that their image under *P* have the same limit, independently of *P* . For instance, consider a diagram in which an object 2 is the target of two arrows labeled by *s*1 (on the left). The diagram obtained by identifying the two arrows along with the objects 1 in their source (on the right) can easily be checked to be equivalent by constructing a bijection between cocones of the first and cocones of the second.

0

2,,,..*,*. *¸* *¸*2,,*,¸*

1

1

% 2 *¸*,¸,*%,¸*¸

2,,..*,*. %*¸* 2,*%,,*

2,,,..*,*. *¸* *¸*2,,*,*%*¸* 2,,,*%,,¸* *¸*2,,..*,*. %*¸* 2,*%,,*

s

1

1 ... 1

1 0 0 1

1 ... 1

1 ... 1

0

1 1 ... 1

s

s

More precisely, if we write *K* : *J ∗ → L* and *K* : *J → L* for the two diagrams and *J* : *J ∗ →J* for the obvious functor, the canonical arrow colim(*K◦J* ) *→* colim(*K*) is an isomorphism, i.e. the functor *J* is final. The same reasoning of course also holds with *s*1 instead of *s*1. We can therefore restrict ourselves to considering diagrams

1 0

in which 2 is the target of at most one arrow *s*1, and of at most one arrow *s*1.

0 1

Conversely, if an object 2 is the target of no arrow *s*1 (on the left), then we can add

0

a new object 1 and a new arrow from this object to the object 2 (on the right) and

obtain an equivalent diagram:

2 ,,*¸¸*2,,..*,*. *¸* 2,*,,*

2,,,*,¸¸*2,,..*,*. *¸* 2,*,,*

0

s1

1 ... 1 1 1 ... 1

The same reasoning holds with *s*1 instead of *s*1 and we can therefore restrict our-

1 0

selves to diagrams in which every object 2 is the target of exactly one arrow *s*1 and

0

one arrow *s*1.

1

Any such diagram *K* is obtained by gluing a finite number of diagrams of the form

1 2 ,s*¸*1

s

0

1

1 1

along objects 1, and is therefore of the form El(*G*) *−→*π

*G −→*I

*L* for some finite graph

*G ∈ G*ˆ: the objects of *G* are the objects 1 in *K*, the edges of *G* are the objects 2 in

*K* and the source and target of an edge 2 are respectively given by the sources of the corresponding arrows *s*1 and *s*1 admitting it as target. For instance, the diagram

1 0

on the left

s1 2 s1 s1 2 s1 s1 2 s1

1 ,0*¸* 1 ,0*¸* 1 ,0*¸*

1 1 1 1

1 z2s1

s

0

1

0  1 *¸* 2 *¸* 3 *¸*

is generated by the graph on the right. *2*

**Lemma** [**4.5**](#_bookmark16)*Given a graph G ∈ G*ˆ*, the associated diagram* ([8](#_bookmark14)) *admits a colimit in L if and only if there exists n ∈L and a morphism ƒ* : *G → N*I *n in L*ˆ *such that every morphism g* : *G → N*I *m in L*ˆ*, with m ∈ L, factorizes uniquely through N*I *n:*

*G*  f *N¸n*  ˛*N¸$m*

I I

g

**Proof** We have seen in proof of Proposition [4.1](#_bookmark9) that morphisms in *L*ˆ(*G, N*I *n*) are

in bijection with cocones in *L* from El(*G*) π*G* I to *n*, and moreover given

*−−→ G −→ L*

a morphism *h* : *n → m* in *G* the morphism

*L*ˆ(*G, N*I *n*) *→*

*L*ˆ(*G, N*I *m*) induced

by post-composition with *N*I *h* is easily checked to correspond to the usual notion of morphism between *n*-cocones and *m*-cocones induced by *N*I *h* (every morphism *N*I *n → N*I *m* is of this form since *N*I is full and faithful). We can finally conclude using the universal property defining colimiting cocones. *2*

# Proofs for deletions of lines

In this section, we detail proofs of properties mentioned in Section [6](#_bookmark23).

* 1. *Sets and partial functions*

Before considering the conservative finite cocompletion of the category *L*, as intro- duced in Definition [2.1](#_bookmark1), it is enlightening to study the category **PSet** of sets and partial functions. A partial function *ƒ* : *A → B* can always be seen

* + 1. as a total function *ƒ* : *A → B {⊥*A*}* where *⊥*A is a fresh element wrt *A*, where given *a ∈ A*, *ƒ* (*a*)= *⊥*A means that the partial function is undefined on *A*,
    2. alternatively, as a total function *ƒ* : *A {⊥*A*}→ B {⊥*B*}* such that *ƒ* (*⊥*A)= *⊥*B. This thus suggests to consider the following category:

**Definition D.1** The category **pSet** of *pointed sets* has pairs (*A, a*) where *A* is a set and *a ∈ A* as objects, and morphisms *ƒ* : (*A, a*) *→* (*B, b*) are (total) functions *ƒ* : *A → B* such that *ƒ* (*a*)= *b*.

Point (ii) of the preceding discussion can be summarized by saying that a partial function can be seen as a pointed function and conversely:

**Proposition D.2** *The category* **PSet** *of sets and partial functions is equivalent to the category* **pSet** *of pointed sets.*

It is easily shown that the forgetful functor *U* : **pSet** *→* **Set**, sending a pointed set (*A, a*) to the underlying set *A*, admits a left adjoint *F* : **Set** *→* **pSet**, defined on objects by *FA* = (*A {⊥*A*}, ⊥*A). This adjunction induces a monad *T* = *UF* on **Set**, from which point (i) can be formalized:

**Proposition D.3** *The category* **PSet** *is equivalent to the Kleisli category* **Set**T

*associated to the monad T* : **Set** *→* **Set***.*

Finally, it turns out that the category **pSet** of pointed sets might have been discovered from **PSet** using “presheaf thinking” as follows. We write *G* for the full subcategory of **PSet** containing two objects: the empty set 0 = *∅* and a set 1 = *{∗}* with only one element, and two non-trivial arrows: 0 *→* 1 and *⊥* : 1 *→* 0 (the undefined function) such that *⊥◦* = id0. We write *I* : *G →* **PSet** for the inclusion

functor. Consider the associated nerve functor *N*I : **PSet** *→ G*ˆ. Given a set *A* the

presheaf *N*I *A ∈ G*ˆ is such that:

* *N*I *A*0= **PSet**(*I*0*, A*) *∼*= *{ }*: the only morphism 0 *→ A* in **PSet** is noted,
* *N*I *A*1 = **PSet**(*I*1*, A*) *∼*= *A {⊥*A*}*: a morphism 1 *→ A* is characterized by the image of *∗∈ A* which is either an element of *A* or undefined,
* *N*I *A* : *N*I *A*1 *→ N*I *A*0 is the constant function whose image is,
* *N*I *A⊥* : *N*I *A*0 *→ N*I *A*1 is the function such that the image ofis *⊥*A.

Moreover, given *A, B ∈* **PSet** a natural transformation from *N*I *A* to *N*I *B* is a pair of functions *ƒ* : *A {⊥*A*}→ B {⊥*B*}* and *g* : *{ }→ { }* such that the diagrams

*A {⊥*A*}* f *B¸ {⊥*B*} A {⊥*A*}* f *{¸⊥*B*}*

*I*

*I*

N*I* A٨ J

JN*I* B٨

and

N A*⊥* ,*,*

,N *,*B*⊥*

*{ }* g *{ ¸} { }* g *{ ¸}*

commutes. Since *{ }* is the terminal set, such a natural transformation is character- ized by a function *ƒ* : *A {⊥*A*} → B {⊥*B*}* such that *ƒ* (*⊥*A)= *⊥*B. The functor *N*I : **PSet** *→ G*ˆ is thus dense and its image is equivalent to **pSet**.

* 1. *A cocompletion of L*

The situation with regards to the category *L* is very similar. We follow the plan of Section [4](#_bookmark8) and first investigate the unlabeled case: *L* is the category with integers as objects and partial injective increasing functions *ƒ* : [*m*] *→* [*n*] as morphisms *ƒ* : *m → n*.

We write *G* for the full subcategory of *L* whose objects are 0, 1 and 2. This is the free category on the graph

1

s

s0 0 *¸*

0 *~~¸~~*s1 *¸*

0 ¸*,* 1

d

0

1

0

¸d*,*1 2

¸*,*0

d

1

1

subject to the relations

*s*1*s*0 = *s*1*s*0 *d*0*s*0 = id1 *d*1*s*1 = id2 *d*1*s*1 = id2 *d*0*d*1 = *d*0*d*1

(D.1)

0 0 1 0 0 0

0 0 1 1

0 0 0 1

(see Proposition [2.4](#_bookmark3)). We write *I* : *G → L* for the embedding and consider the associated nerve functor *N*I : *L→ G*ˆ. Suppose given an object *n ∈ L*, the associated presheaf *N*I *n* can be described as follows. Its sets are

* *N*I *n*0= *L*(*I*0*, n*) *∼*= *{ }*,
* *N*I *n*1= *L*(*I*1*, n*) *∼*= [*n*] *{⊥}*,
* *N*I *n*2= *L*(*I*2*, n*) *∼*=

*{*(*i, j*) *∈* [*n*] *×* [*n*] *| i < j} {*(*⊥, i*) *| i ∈* [*n*]*} {*(*i, ⊥*) *| i ∈* [*n*]*} {*(*⊥, ⊥*)*}*: a partial function *ƒ* : 2 *→ n* is characterized by the pair of images (*ƒ* (0)*,ƒ* (1)) of 0*,* 1 *∈* [*n*], where *⊥* means undefined.

and morphisms are

* *N*I *ns*0 : *N*I *n*1 *→ N*I *n*0 is the constant function whose image is,

0

* *N*I *nd*0 : *N*I *n*0 *→ N*I *n*1 is the function whose image is *⊥*,

0

* *N*I *ns*1 : *N*I *n*2 *→ N*I *n*1 is the second projection,

0

* *N*I *ns*1 : *N*I *n*2 *→ N*I *n*1 is the first projection,

1

* *N*I *nd*1 : *N*I *n*1 *→ N*I *n*2 sends *i ∈* [*n*] *{⊥}* to (*⊥, i*)

0

* *N*I *nd*1 : *N*I *n*1 *→ N*I *n*2 sends *i ∈* [*n*] *{⊥}* to (*i, ⊥*)

1

Such a presheaf can be pictured as a graph with *N*I *n*1 as set of vertices, *N*I *n*2 as set of edges, source and target being respectively given by the functions *N*I *ns*1 and

1

*N*I *ns*1:

0

*⊥*

*...*

0 1 2 *n −* 1

Its vertices are elements of [*n*] *{⊥}* and edges are of the form

* + *i → j* with *i, j ∈* [*n*] such that *i < j*,
  + *i →⊥* for *i ∈* [*n*]
  + *⊥→ i* for *i ∈* [*n*]
  + *⊥→⊥*

Morphisms are usual graphs morphisms which preserve the vertex *⊥*. We are thus naturally lead to define the following categories of pointed graphs and graphs with partial functions. We recall that a graph *G* = (*V, s, t, E*) consists of a set *V* of vertices, a set *E* of edges and two functions *s, t* : *E → V* associating to each edge its source and its target respectively.

**Definition D.4** We define the category **pGraph** of *pointed graphs* as the category whose objects are pairs (*G, x*) with *G* = (*V, E*) and *x ∈ V* such that for every vertex there is exactly one edge from and to the distinguished vertex *x*, and mor- phisms *ƒ* : *G → G∗* are usual graph morphisms consisting of a pair (*ƒ*V *, ƒ*E) of functions *ƒ*V : *V*G *→ V*G*′* and *ƒ*E : *E*G *→ E*G*′* such that for every edge *e ∈ E*G, *ƒ*V (*s*(*e*)) = *s*(*ƒ*E(*e*)) and *ƒ*V (*t*(*e*)) = *t*(*ƒ*E(*e*)), which are such that the distinguished vertex is preserved by *ƒ*V .

**Definition D.5** We define the category **PGraph** of *graphs and partial morphisms* as the category whose objects are graphs and morphisms *ƒ* : *G → G∗* are pairs (*ƒ*V *, ƒ*E) of partial functions *ƒ*V : *V*G *→ V*G*′* and *ƒ*E : *E*G *→ E*G*′* such that

* + for every edge *e ∈ E*G such that *ƒ*E(*e*) is defined, *ƒ*V (*s*(*e*)) and *ƒ*V (*t*(*e*)) are both defined and satisfy *ƒ*V (*s*(*e*)) = *s*(*ƒ*E(*e*)) and *ƒ*V (*t*(*e*)) = *t*(*ƒ*E(*e*)),
  + for every edge *e ∈ E*G such that *ƒ*V (*s*(*e*)) and *ƒ*V (*t*(*e*)) are both defined, *ƒ*E(*e*) is also defined.

More briefly: a morphism is defined on an edge if and only it is defined on its source and on its target.

Similarly to previous section, a partial morphism of graph can be seen as a pointed morphism of graph and conversely:

**Proposition D.6** *The categories* **pGraph** *and* **PGraph** *are equivalent.*

Now, notice that the category *L* is isomorphic to the full subcategory of **PGraph**

whose objects are the graphs whose set of objects is [*n*] for some *n ∈* N, and

such that there is an edge *i → j* precisely when *i < j*. Also notice that the full subcategory of **pGraph** whose objects are the graphs *N*I *n* (with *⊥* as distinguished vertex) with *n ∈* N is isomorphic to the full subcategory of *G*ˆ whose objects are the *N*I *n* with *n ∈* N. And finally, the two categories are equivalent via the isomorphism of Proposition [D.6](#_bookmark38). From this, we immediately deduce that the functor *N*I : *L→ G*ˆ is full and faithful, i.e.

**Proposition D.7** *The functor I* : *G → L is dense.*

We can now follow Section [4](#_bookmark8) step by step, adapting each proposition as necessary. The conditions satisfied by presheaves in *P* introduced in Proposition [4.3](#_bookmark11) are still valid in our new case:

**Proposition D.8** *Given a presheaf P ∈ L*ˆ *which is an object of P,*

* + 1. *the underlying graph of P is ﬁnite,*
    2. *for each non-empty path x* → *y there exists exactly one edge x → y,*
    3. *P* (*n* + 1) *is the set of paths of length n in the underlying graph of P, and P* (0)

*is reduced to one element.*

**Proof** The diagrams of the form ([6](#_bookmark12)) and ([7](#_bookmark13)) used in proof of Proposition [4.3](#_bookmark11) still admit the same colimit *n* + 1 with the new definition of *L* and 0 is still initial. It can be checked that the limit of the image under a presheaf *P ∈ L*ˆ of a diagram ([6](#_bookmark12)) is still the set of paths of length *n* in the underlying graph of *P* . *2*

Lemma [4.4](#_bookmark15) is also still valid:

**Lemma D.9** *A presheaf P ∈ L*ˆ *preserves ﬁnite limits, if and only if it sends the colimits of diagrams of the form*

π*G* I

El(*G*) *−−→G −→L*

*to limits in* **Set***, where G ∈ G*ˆ *is a ﬁnite pointed graph such that the above diagram admits a colimit. Such a diagram in L is said to be* generated by the pointed graph *G.*

**Proof** The proof of Lemma [4.4](#_bookmark15) was done “by hand”, but we mentioned a more abstract alternative proof. In the present case, a similar proof can be done but would be really tedious, so we provide the abstract one. In order to illustrate why we have to do so, one can consider the category of elements associated to the presheaves representable by 0 and 1, which are clearly much bigger than in the case of Section [4](#_bookmark8):

1

s

s0 0 *¸*

El(*N* 0) *∼*= ¸*,* 0 *⊥~~¸~~* s1 (*⊥¸, ⊥*)

1

I 0 ¸*,* d1

0 ¸*,* 0

d

d

1

1

and

1

s

0

1

s

1

1

d01

d1

%*%*

(*⊥, ⊥*)

%*%*

*~ ⊥* s1 (*⊥¸,* 1)

El(*N*I1) =

%*%*

*,* 1

0 1

s

s

0 0

d0 1

0 d1 s0 z

,*c*  *¸* 0

s1 *¸*

0 1 ¸*,* 1

s

d

0

1

1

(1*, ⊥*)

subject to relations which follow directly from ([D.1](#_bookmark37)).

Before going on with the proof, we need to introduce a few notions. A functor *F* : *C → D* is called *ﬁnal* if for every category *E* and diagram *G* : *D → E* the canonical morphism colim(*G ◦ F* ) *→* colim(*G*) is an isomorphism [[6](#_bookmark30)]: restricting a diagram along *F* does not change its colimit. Alternatively, these functors can be characterized as functors such that for every object *D ∈ D* the category is non- empty and connected (there is a zig-zag of morphisms between any two objects). A functor *F* : *C → D* is called a *discrete ﬁbration* if for any object *C ∈ C* and morphism *g* : *D → FC* in *D* there exists a unique morphism *ƒ* : *C∗ → C* in *C* such that *Fƒ* = *g* called the lifting of *g*. To any such discrete fibration one can associate

a presheaf *P ∈ D*ˆ defined on any *D ∈ D* by *PD* = *F−*1(*D*)= *{C ∈ C | FC* = *D}*

and on morphisms *g* : *D∗ → D* as the function *Pg* which to *C ∈ PD* associates the source of the lifting of *g* with codomain *C*. Conversely, any presheaf *P ∈ D*ˆ induces

a discrete fibration El(*P* ) *−→*π *D*, and these two operations induce an equivalence of

categories between the category *D*ˆ and the category of discrete fibrations over *D*. It

was shown by Par´e, Street and Walters [[8](#_bookmark34),[11](#_bookmark36)] that any functor *F* : *C → D* factorizes as final functor *J* : *C → E* followed by a discrete fibration *K* : *E → D*, and this factorization is essentially unique: this is called the *comprehensive factorization* of a functor. More explicitly, the functor *K* can be defined as follows. The inclusion functor **Set** *→* **Cat** which send a set to the corresponding discrete category admits a left adjoint Π0 : **Cat** *→* **Set**, sending a category to its connected components (its set of objects quotiented by the relation identifying two objects linked by a zig-zag

of morphisms). The discrete fibration part *K* above can be defined as El(*P* ) *−→*π *D*

where *P ∈ D*ˆ is the presheaf defined by *P* = Π0(*−/F* ). In this precise sense, every

diagram *F* in *D* is “equivalent” to one which is “generated” by a presheaf *P* on *D* (we adopted this informal terminology in the article in order to avoid having to introduce too many categorical notions).

In our case, we can thus restrict to diagrams in *L* generated by presheaves on

*L*. Finally, since *I* : *G → L* is dense, we can further restrict to diagrams generated by presheaves on *G* by interchange of colimits. *2*

Lemma [4.6](#_bookmark17) applies almost as in Section [4](#_bookmark8): since the morphism *ƒ* : *G → N*I *n* (seen as a partial functions between graphs) has to satisfy the universal property of Lemma [4.5](#_bookmark16), by choosing for every vertex *x* of *G* a partial function *g*x : *G → N*I *m* which is defined on *x* (such a partial function always exists), it can be shown that the function *ƒ* has to be total. The rest of the proof can be kept unchanged. Similarly,

Proposition [4.7](#_bookmark18) applies with proof unchanged.

Finally, we have that

**Theorem D.10** *The free conservative ﬁnite cocompletion P of L is equivalent to the full subcategory of L*ˆ *whose objects are presheaves P satisfying the conditions of Proposition* [*D.8*](#_bookmark39)*. Since its objects P satisfy I∗I∗*(*P* ) *∼*= *P, it can equivalently be characterized as the full subcategory of G*ˆ *whose objects P are*

1. finite*,*
2. transitive*: for each non-empty path x* → *y there exists exactly one edge x → y,*
3. pointed*: P* (0) *is reduced to one element.*

From this characterization (which can easily be extended to the labeled case), along with the correspondence between pointed graphs and graphs with partial functions (Proposition [D.6](#_bookmark38)), the category is shown to be equivalent to the category described in Theorem [6.1](#_bookmark24): the relation is defined on vertices *x, y* of a graph *G* by *x < y* whenever there exists a path *x* → *y*.

As in case of previous section, the forgetful functor **pGraph** *→* **Graph** admits a left adjoint, thus inducing a monad on **Graph**. The category **pGraph** is equivalent to the Kleisli category associated to this monad, which is closely related to the *exception monad* as discussed in [[3](#_bookmark27)].

# E Modeling repositories

We briefly detail here the modeling of repositories evoked in Section [7](#_bookmark25). As explained in the introduction, repositories can be modeled as partially ordered sets of patches,

i.e. morphisms in *L*. Since some of them can be incompatible, it is natural to model them as particular labeled event structures.

**Definition E.1** An *event structure* (*E, ≤,* #) consists of a set *E* of *events*,a partial order relation *≤* on *E* and *incompatibility* relation on events. We require that

1. for any event *e*, the downward closure of *{e}* is finite and
2. given *e*1, *e∗*

1

and *e*2 such that *e*1 *≤ e∗*

and *e*1#*e*2, we have *e∗* #*e*2.

Two events *e*1 and *e*2 are *compatible* when they are not incompatible, and *inde- pendent* when they are compatible and neither *e*1 *≤ e*2 nor *e*2 *≤ e*1. A *conﬁguration x* is a finite downward-closed set of compatible events. An event *e*2 is a *successor* of an event *e*1 when *e*1 *≤ e*2 and there is no event in between. Given an event *e* we write *↓e* for the configuration, called the *cause* of *e*, obtained as the down- ward closure of *{e}* from which *e* was removed. A *morphism* of event structures *ƒ* : (*E, ≤,* #) *→* (*E∗, ≤∗,* #*∗*) is an injective function *ƒ* : *E → E∗* such that the image of a configuration is a configuration. We write **ES** for the category of event structures.

1

1

To every event structure *E*, we can associate a *trace graph T* (*E*) whose vertices

are configurations and edges are of the form *x −→*e *x {e}* where *x* is a configuration

such that *e /∈ x* and *x {e}* is a configuration. A *trace* is a path *x* → *y* in this

graph. Notice that two paths *x* → *y* are of the same length. Moreover, given two configurations *x* and *y* such that *x ⊆ y*, there exists necessarily a path *x* → *y*. It can be shown that this operation provides a faithful embedding *T* : **ES** *→* **Graph** from the category of event structures to the category of graphs, which admits a right adjoint.

**Example E.2** An event structure with five events is pictured on the left (arrows represent causal dependencies and *∼* incompatibilities). The associated trace graph is pictured on the right.

*{a, b, c, d}*

,*,*

*b*

*c∗*

*d* ,*,¸*

,*¸* *c ,*

*{a, b, c∗}* ¸c*′ ,{a, b}*

d

*{a, b, c}*

c

¸b*¸*

*{a, c}*

c

*a* b *{a, c∗}* ¸*, {a}*

¸*¸*

¸*¸*b

c a ,*,*

*′*

*∅*

**Definition E.3** A *categorical event structure* (*E, λ*) in a category *C* with an initial object consists of an event structure equipped with a *labeling* functor *λ* : (*TE*)*∗ → C*, where (*TE*)*∗* is the free category generated by the graph *TE*, such that *λ∅* is the initial object of *C* and the image under *λ* of every square

e2 "*z* ,e*¸*1

*y*1 ,*¸* "*y*2*,*

e1 *x* e2

in *TE* is a pushout in *C*.

The following proposition shows that a categorical event structure is character- ized by a suitable labeling of events of *E* by morphisms of *C*.

**Proposition E.4** *The functor λ is characterized, up to isomorphism, by the image*

*of the transitions ↓e −→*e *↓e {e}.*

We can now define a *repository* to be simply a finite categorical event structure (*E, ≤,* #*,λ* : *T* (*E*) *→ L*). Such a repository extends to a categorical event structure (*E, ≤,* #0*,I ◦ λ* : *T* (*E*) *→ P*), where #0 is the empty conflict relation. The *state S* of such an event structure is the file obtained as the image *S* = *I ◦ λ*(*E*) of the maximal configuration: this is the file that the users is currently editing given his repository. Usual operations on repositories can be modeled in this context, for instance importing the patches of another repository is obtained by a pushout construction (the category of repositories is finitely cocomplete).