Geometric Models of Concurrent Computations

Samuel Mimram

▶ **To cite this version:**

Samuel Mimram. Geometric Models of Concurrent Computations. Logic in Computer Science [cs.LO]. Université Paris 7, 2016. tel-01783442

HAL Id: tel-01783442
https://inria.hal.science/tel-01783442

Submitted on 2 May 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

École Polytechnique

Geometric Models of Concurrent Computations

Mémoire d'habilitation à diriger des recherches
présenté et soutenu publiquement par

Samuel Mimram

le 16 septembre 2016
devant le jury composé de

M.	Pierre-Louis	Curien	rapporteur
M.	Patrick	Dehornoy	
M.	Éric	Goubault	
M.	Jean	Goubault-Larrecq	rapporteur
Mme	Kathryn	Hess Bellwald	
M.	Timothy	Porter	
M.	Glynn	Winskel	rapporteur

Abstract

Since the 90s, geometric models have been introduced for concurrent programs. In those, a point corresponds to a state, a path to an execution and a deformation of a path to an equivalence between executions. They are useful to analyze programs because they provide a convenient representation of their state space, on which one can use some of the well-developed tools and invariants from geometry (curvature, homology, etc.). Conversely, the study of the spaces arising as models brings new problems of purely geometric nature: most importantly, they are naturally equipped with a direction (of time), which requires adapting most usual notions. In this habilitation thesis, we present such models that we have developed and studied, as well as general techniques to do so and the results they have allowed us to obtain. Those have been applied to various notion of "concurrent programs": programs in an imperative language extended with a parallel construction and resources, but also distributed protocols, version control systems, or rewriting systems. The "geometric models" we have studied for those are also of various nature: precubical sets, directed topological spaces, generalized metric spaces, or polygraphs.

Contents

Gately begins to consider this hopefully nonrecurring dream even more unpleasant than the tiny-pocked-Oriental-woman dream, overall. Other terms and words Gately knows he doesn't know from a divot in the sod now come crashing through his head with the same ghastly intrusive force, e.g. ACCIACCATURA and ALEMBIC, LATRODECTUS MACTANS and NEUTRAL DENSITY POINT, CHIAROSCURO and PROPRIOCEPTION and TESTUDO and ANNULATE and BRICOLAGE and CATALEPT and GERRYMANDER and SCOPOPHILIA and LAERTES – and all of a sudden it occurs to Gately the aforethought EXTRUDING, STRIGIL and LEXICAL themselves – and LORDOSIS and IMPOST and SINISTRAL and MENISCUS and CHRONAXY and POOR YORICK and LUCULUS and CERISE MONTCLAIR and then DE SICA NEOREAL CRANE DOLLY and CIRCUMAMBIENTFOUNDDRAMALEVIRATEMARRIAGE and then more lexical terms and words speeding up to chipmunkish and then HELIATED and then all the way up to a sound like a mosquito on speed, and Gately tries to clutch both his temples with one hand and scream, but nothing comes out.

David Foster Wallace, *Infinite Jest*

À Papy.

Chapter 0

Introduction

This memoir presents some of the works that I have done, often in collaboration with other people, in the last few years. I provide here a synthetic, yet detailed, exposition of those, which were selected and organized in order to achieve a coherent presentation of the subjects I have studied, the tools I have used, and the results I have obtained. We begin by briefly describing what we mean here by geometric models and concurrent computations, as well as recalling some historical background on the subjects.

0.1 Geometric models for concurrent computations

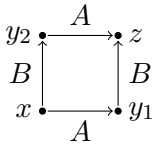
Concurrent programs consist of multiple processes running in parallel. Their use has become more and more widespread in order to efficiently exploit recent architectures (processors with many cores, clouds, etc.), but they are notoriously difficult to design and to reason about: one has to ensure that the program will not go wrong, regardless of the way the different processes composing the program are scheduled. In principle, in order to achieve this task with the help of a computer, we could apply traditional verification techniques for sequential programs on each of the possible executions of the program. But this is not feasible in practice because the number of those executions, or *schedulings*, may grow exponentially with the size of the program. Fortunately, it can be observed that many of the schedulings are equivalent in the sense that one can be obtained from the other by permuting independent instructions: such equivalent executions will always lead to the same result. Hence, if one of those executions can be shown not to lead to an error, neither will any other execution which is equivalent to it.

This suggests that a model for concurrent programs should incorporate not only the possible executions of the program (as in traditional interleaving semantics), but also the commutations between instructions, following the principle of what is now called *true concurrency*. Interestingly, the resulting models are algebraic structures which can be interpreted *geometrically*: roughly as topological spaces in which paths correspond to executions and two executions are equivalent when the corresponding paths are homotopic, i.e. connected by a continuous deformation from one to the other. In order to make this connection precise, it turns out that topological spaces are not exactly the right notion for our purposes. One needs to use a *directed* variant, i.e. to incorporate a notion of irreversible time.

Starting from very practical motivations (the verification of concurrent programs), questions of a more theoretical nature arise. What is the geometry of concurrent programs? What is a good notion of a directed space, and how do classical techniques from (algebraic) topology

apply to this setting? How can a geometrically refined understanding of concurrency be used in order to design new and more efficient algorithms for studying concurrent programs? The goal of this memoir is to give a general overview of our current understanding, regarding these questions, starting from the point of view that the code of a concurrent program is only a convenient notation for a geometric object which we will describe and study.

Interleaving models for concurrency. Historically, the first models for concurrent programs were the so-called *interleaving* models, which essentially consist of all sequences of actions that could occur in the execution of a program. In particular, the sequences of actions performed by two processes in parallel are those performed by each of the processes, scheduled in an arbitrary way. For instance, consider a program of the form $A \parallel B$, made of two instructions A and B executed in parallel. Its semantics would be the following graph with four vertices and four edges:



(1)

Notice that the two maximal paths are labeled by $A \,.\, B$ and $B \,.\, A$, i.e. the two interleavings of A and B. These models are however somehow limited. Firstly, they assume that the execution of program is *sequentially consistent* [Lam79], i.e. can be reduced to interleavings of actions, whereas in practice processes executed in parallel can exhibit more behaviors than their interleavings. Secondly, as explained below, they forget about some important relationships between traces, because of which these models lack desirable properties, such as being stable under refinement of actions (replacing an action by a sequence of actions for instance), and makes the models very large and thus difficult to exploit in practice for verification purposes (this is sometimes called the *state space explosion problem*).

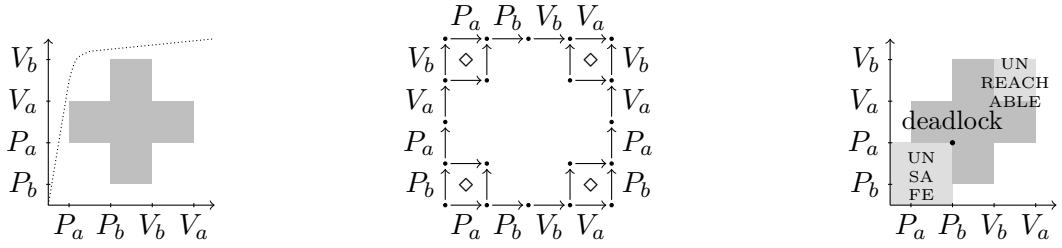
Asynchronous models for concurrency. The previous semantics does not take into account when two sequences of instructions are equivalent, i.e. whether the two actions A and B are independent or not in previous example. For instance, with A and B being respectively `x:=1` and `y:=2`, the two actions are independent because any execution of the program will lead to a state where the variables `x` and `y` respectively contain 1 and 2. However, this is not the case when A and B are, respectively, `x:=2` and `x:=2*x`. Starting from a state where `x` contains 0, the execution $A \,.\, B$ will end in a state where `x` contains 4, while execution $B \,.\, A$ will end in a state where `x` contains 2. Even worse, the simultaneous execution of A and B can even end in other states, as sometimes happens in practice. In this case, the order in which the actions are scheduled matters. In order to distinguish between the two types of situation, we will equip our graph with a relation $\diamond$ on paths, indicating when they are equivalent in this sense, in order to obtain what is called an *asynchronous graph*. This idea of taking into account commutation of actions in concurrent systems dates back to Mazurkiewicz [Maz88] and has led to development of the theory of *trace monoids* [DR95, DM97] of which asynchronous graphs are a "typed" variant, as well as more generally all *truly concurrent* models [WN95] (see also Section 1.4).

In order to avoid two incompatible instructions being executed at the same time, most operating systems provide *mutexes*, as introduced by Dijkstra [Dij68a]. Those are particular kinds of resources which can be held by at most one process at a time: given a mutex a, a

process can either lock or release the resource by respectively performing the instructions P_a or V_a, and if a process tries to lock a mutex which was already taken, it is then frozen until the mutex is released. From the point of view of the semantics, the usage of those instructions has two effects: firstly, it forbids some states (those in which more than one process would have locked the mutex), and secondly it explicitly states that some schedulings are not equivalent. For instance, the semantics of $(A_1 ; A_2 ; A_3) \parallel (B_1 ; B_2 ; B_3)$ and $(P_a ; A ; V_a) \parallel (P_a ; B ; V_a)$ are, respectively,

Any two maximal paths in the first graph are equivalent, while the two maximal paths in the second graph are not equivalent. Moreover, the second graph can be obtained from the first one by removing vertices in the middle and adjacent edges (those vertices would correspond to positions where the mutex a is locked twice): we explain in Chapter 1 that a semantics of programs with mutexes can be generally obtained in this way, by associating an asynchronous graph with a program, and then removing forbidden vertices.

Topological models for concurrency. In the asynchronous graph semantics presented above, the executions of the program correspond to paths in the graph. Moreover, the squares where the paths in the boundary are equivalent (i.e. those squares marked with "$\diamond$") can be regarded as "filled squares" and the other ones as "empty squares": intuitively, when a square of the form (1) is filled, there is enough room to allow for a deformation, or *homotopy*, to exist between paths $A \,.\, B$ and $B \,.\, A$. In order to make this intuition more formal, it is tempting to investigate another type of geometric model for programs, based on topological spaces instead of graphs, in which an execution corresponds to a path and an equivalence corresponds to a homotopy between paths. For instance, with the program $(P_a ; P_b ; V_b ; V_a) \parallel (P_b ; P_a ; V_a ; V_b)$, is associated the topological space on the left below, obtained from $[0,1] \times [0,1]$ by removing the darkened region (the points in this region would correspond to the states where either a or b has been locked twice, which is forbidden):



(2)

In this space, the paths starting from the beginning position (the lower left corner) and which are "increasing" (i.e. going right and up) correspond to executions. For instance, the dotted path corresponds to the second process executing $P_b \,.\, P_a \,.\, V_a$; then the first process executing P_a; then the second process executing V_b; and finally the first process executing $P_b \,.\, V_b \,.\, V_a$. Paths which are not increasing make no sense from a computational point of view: they correspond to executions which go backwards in time at some point. We thus have

to consider a variant of the notion of topological space which is *directed* in the sense that the space comes equipped with a time direction, i.e. extra structure specifying which paths can be considered as "increasing" or "directed". One can then study the geometry of these spaces, and in particular the structure of directed paths up to a suitable notion of homotopy, which corresponds to equivalence classes of executions, up to commutation of independent actions, by adapting (when possible) classical constructions from algebraic topology. The topological semantics can also be precisely related to the asynchronous semantics via the process of geometric realization: notice the similarity of the topological space on the left with the asynchronous graph in the middle! If the situation seems to be quite simple and clear in the above examples, many subtleties occur when more than two processes are involved, i.e. when considering spaces of dimension greater than 2.

The use of geometry to model concurrent processes was first advocated as a fundamental tool to solve the state space explosion problem by Pratt [Pra91]. It has led to many developments: a state of the art at the beginning of the century can be found in [Gou95] and a more up-to-date and recent one in [FGH$^+$16]. Recent developments include applications to deadlock detection [FGR98], distributed databases [YPK79, Pap83, LP81, Gun94, FRG06], state space analysis [GR02, FGH$^+$12], and more generally static analysis [GH05, BCC$^+$11, H$^+$16].

Some of the (directed) topological models for programs are detailed in Chapter 3, where we also introduce and study a more "quantitative" variant, using a generalized notion of metric space, where the distance between two points measures the time that an execution would take to go from the first to the second state [MG16]. The resulting model, in the case of concurrent programs using mutexes only, allows us to exhibit quite interesting invariants in Chapter 4: they are *non-positively curved*, in the general sense introduced by Gromov for metric spaces [Gro87], from which it follows that they enjoy many useful properties.

Cubical models for concurrency. In the same way that algebraic topology usually starts from simplicial sets rather than directly topological spaces, many properties of programs are more easily studied not directly on topological models, but on more algebraic ones. The most widespread and used such model is the one of *precubical sets*, also called *Higher-Dimensional Automata* (HDA) in this context. These correspond to spaces formally obtained by gluing cubes of any dimension along their faces (just as simplicial sets are made of simplices). They can also be seen as a refinement of the model of asynchronous graphs described earlier, which not only takes into account whether two actions commute or not, but also whether k actions commute, for every $k \in \mathbb{N}$: from a geometric standpoint these not only encode information about the fundamental group (or category), but also higher ones. Of course, they can be precisely related to topological models by the means of geometric realization. Their use to model concurrency dates back to the beginning of the subject [Pra91, Gou96a, Gou01] and coincide with the introduction of bisimulation semantics for those by van Glabbeek [vG91]. The need for a proper adaptation of classical geometric invariants such as homotopy and homology in this context was first advocated in [GJ92] and has been developed considerably since then [Gou95, Gau03, GG03, Fah04, Gra04, Gau05, Gra05, BW06, Gau06, Gau08, Gra09, Gau10, Gau11, Kah13, Gau14a, DGG15], see also Section 3.1. In Chapter 2, we recall the formal definition of precubical models for concurrency and construct adjunctions with more classical models for concurrency (asynchronous transition systems, event structures, Petri nets), generalizing those investigated by Winskel et al. [WN95], in order to make explicit the way in which they are related to those.

Verification of concurrent programs. One of the main interests in the connection between semantics of concurrent programs and geometry is that it provides us with a new point of view on programs, thus allowing for the formulation of new algorithms for program verification. For instance, consider the rightmost state space in (2). Illustrated is a *deadlock* point. Starting from this point, there exists no non-constant increasing path; in other words, the point corresponds to a state of the program in which no instruction can be executed. This kind of undesirable behavior is specific to concurrent programs, and typically occurs when processes are waiting for each other (e.g. to free a resource or to produce data). The points in the lower left square are called *unsafe*: they correspond to states of the program from which an execution might lead to a deadlock. The points in the upper right square are called *unreachable*: no directed path from the beginning position ends in that square, which indicates the existence of states which can never occur during an execution. While this is not an error per se, their presence is often the sign of a poor design in the program (or worse). Based on the geometric characterization of such states (and others of similar interest), people were able to formulate algorithms to compute them, thus providing guarantees about the safety of programs [FGR98, FGH$^+$16].

Another fundamental application of the geometric techniques is the reduction of the number of paths or states to explore, based on the idea that the evaluation of two homotopic paths always lead to the same result. A first construction is provided by the *category of components* [FGHR04, Hau05b, Hau05a, GH05, Hau06, GH07, GHK10], which identifies portions of programs in which "nothing interesting happens" from the concurrency point of view, thus providing us with a compact description of the geometry of the program. We show in Chapter 2 (Section 2.5, see also [GHM13]) that this construction, arising from theoretical considerations, is closely related to the technique of partial order reduction [God96], introduced in order to improve verification of concurrent systems. A second construction is the computation of the *path space* (the space of directed paths up to homotopy): once this space is computed, it suffices to apply traditional (sequential) verification techniques on each representative of each homotopy class of paths, in order to cover all possible schedulings of the program. The implementation of an efficient algorithm to compute it is presented in Chapter 3 (Section 3.3, see also [FGH$^+$12]).

Computability with fault-tolerant protocols. Another area where geometric models and tools have proven very successful is to determine whether a given task can be implemented by concurrent programs, also called *protocols* in this context, in the presence of failures. Typically, the *consensus* task consists in making all the process agree on a value, which is chosen among those they initially have in their memory. In the case of two processes, this was shown to be impossible by Fischer, Lynch and Paterson [FLP85] using a "connectedness argument" on a certain graph, already quite geometric in nature, and was then generalized to any number of processes in the presence of a single failure [BMZ88]. However, to show results in the presence of more failures, people agreed that more powerful techniques would be needed. A few years later, the conjecture that the *k-set agreement* task (a relaxed form of consensus) cannot be solved in the general case, formulated in [Cha90], was shown [BG93, SZ93, HS93]. Basically, the idea of Herlihy and Shavit was to generalize the previous argument using, instead of a graph, a simplicial complex called the *protocol complex* which encodes all the possible knowledge of the processes and their coherence. This approach turned out to be quite fruitful and generalizes to many other tasks, see [HKR14] for a recent panorama of the subject. The links between these geometric constructions and geometric models for

concurrency described earlier were first hinted at, in simple cases, by Goubault [Gou96b, Gou96c, Gou97] and fully formalized only recently in some of our work [GMT15]; this is detailed in Chapter 5.

Geometric invariants of rewriting systems. In this memoir, concurrent systems should be understood in a broad sense: we are more interested in understanding how general and powerful geometric methods are than studying a particular computation model. In this perspective, a string rewriting system can also be thought of as a concurrent program: here, the state is constituted of a single string, and the various "processes" concurrently modify some portion of it according to the rules, some actions being incompatible (the *critical pairs*). The study of properties of monoids (or groups) through the homology of spaces (or chain complexes obtained from resolutions) associated with those dates back to the beginnings of algebraic topology, in the first half of last century. Those invariants were first linked with properties of rewriting systems presenting monoids by Squier [Squ87]. He showed that a monoid with a finite convergent presentation necessarily had finite homology groups in low dimensions, and constructed a particular monoid for which it was not the case, thus showing that rewriting is not universal to decide the word problem: not every finitely presented decidable monoid admits a finite convergent presentation. The homological property was then refined into a homotopical finiteness property [SOK94], and can be generalized in any dimension, giving rise to a fine hierarchy of monoids [Bro87, Coh92]. We refer the reader to Chapter 7 for a more detailed presentation of these aspects of rewriting.

Higher-dimensional rewriting systems. The starting point of algebraic topology is to consider paths, paths between paths (called homotopies), homotopies between homotopies, and so on. In the same vein, it is natural to extend rewriting in higher-dimensions and consider rewriting paths between rewriting paths, rewriting paths between rewriting paths between rewriting paths, and so on. A generalization of rewriting systems in order to achieve this was proposed by Burroni [Bur93] under the name of *polygraphs*. Depending on the point of view, they can be seen

— as higher-dimensional generalizations of rewriting systems,
— as higher-dimensional generalizations of presentations (from monoids to n-categories),
— as generalizations of presheaf categories such as simplicial and cubical sets (in the sense that they are directed, and they contain cells of any "shape"),
— as non-abelian variants of resolutions (as initiated by Métayer [Mét03, LM09, LMW10]).

This last point hints at why they can also be themselves seen as a geometric model. Many ongoing work study various aspects of these. Firstly, generalizations of traditional rewriting techniques (for showing confluence and termination) have been proposed [Laf03, Gui04, Gui06, Mim08, Mim10a, Mim14]. They are not straightforward: for instance, contrarily to string or term rewriting systems, a finite rewriting system can give rise to an infinite number of critical pairs [Laf03]. Secondly, polygraphs can also be used to keep track of higher-dimensional coherence cells in presentations as developed by Guiraud and Malbos [GM09, GM12b, GM13, GM12c, GMM$^+$13, GM14, GGM15], which is closely related to the notion of polygraphic resolution mentioned above, see Chapter 7 and 8 for more details. Thirdly, there is still some debate about what one should expect from the notion of higher-dimensional rewriting. In particular, as for now, in an n-dimensional rewriting system, the k-dimensional generators for $k \leqslant n$ are considered as building blocks for the structure (they freely generate an n-category) and $(n+1)$-dimensional generators as oriented relations between n-cells. However, it is

sometimes desirable to allow rewriting at lower levels too. A first contribution toward the definition of a generalized notion allowing this is presented in Chapter 8, see also [CM15]. Fourthly, the structure of higher-dimensional rewriting system is very nice from a theoretical point of view, but, apart from restrictions in low dimensions, there is currently no practical implementation of them. Some progress in this direction is reported in Chapter 10.

New points of view and research topics. We believe that the investigation of the connections between concurrent computations and geometric models has three noticeable byproducts: firstly, it helps to abstractly visualize the state space of programs and thus to come up with new algorithms by better understanding their structure (for computing deadlocks, trace spaces, etc.), secondly one can hope to meaningfully use the well-developed tools and invariants from geometry (curvature, homology, etc.), thirdly the study of spaces arising from programs brings new problems of purely geometric nature (properly defining invariants for directed spaces for instance). This memoir aims at studying and developing some of the aspects of this correspondence, through the definition and the study of particular models for concurrent programs.

0.2 About this manuscript

This *Habilitation à Diriger des Recherches* summarizes some of my recent work. I have tried to attribute clearly all the mentioned results, whether they are mine or not. The only chapters where I do not claim any originality, except perhaps in the way of presenting things, are Chapters 1 and 7, which are introductory. Except for particularly important or enlightening ones, most proofs have been removed and can be found in the references to detailed articles. We also have mentioned future work, because we think that those research tracks are interesting and some other people might enjoy following them too; it also shows that our results are often a step which is part of a larger programme. Many of the sections can be read independently, even though they are in fact quite related.

Plan. In Chapter 0, as the reader has probably noticed, we provide a general introduction to the present manuscript.

In Chapter 1, we begin by introducing a simple concurrent programming language, which we use throughout the memoir to provide examples: we follow the presentation of [FGH+16] and illustrate our techniques on a realistic language with manipulations of values, control flow operations, concurrency and resources. Starting from the control flow graph of a program, we introduce its interleaving semantics, which is then modified in order to obtain a truly concurrent semantics, and present the most important properties a program should satisfy with respect to this semantics.

In Chapter 2, we then explain how it can be generalized into a precubical semantics, which encodes all the k-ary commutations of actions. We first recall general theorems about presheaf categories, then introduce more specifically the category of precubical sets and use it to provide a semantics for our programming language. In order to relate the resulting model to more classical ones for concurrency (asynchronous transition systems, event structures and Petri nets), we extend the adjunctions of Winskel et al. [WN95] to precubical sets. In order to also relate it to existing techniques on a more practical side, we show that the abstract

construction of the category of future components of Haucourt et al. [FGHR04] can be seen as some form of generalized partial order reduction, as introduced by Godefroid [God96].

In Chapter 3, we first recall topological models for concurrency: usual topological spaces have to be equipped with a notion of "time direction" and most constructions have to be adapted accordingly. In order to provide these models with quantitative information (the duration of an execution), we introduce a variant using metric spaces and relate it to topological models: the category of metric spaces lacking good properties, one has to work with the relaxed variant of metric spaces introduced by Lawvere [Law73].

In Chapter 4, we study the particular case of programs using only mutexes, the most widely used synchronization primitive in practice, exhibiting quite interesting properties. Firstly, we show that the associated metric model has non-positive curvature, in the sense of Gromov for metric spaces [Gro87], which has interesting consequences, and leads us to formulate a reasonable definition of a non-positively curved precubical set, admitting such spaces as metric realization. Secondly, we show that in this case homotopy and dihomotopy (the directed variant of homotopy) coincide, by using higher-dimensional rewriting techniques.

In Chapter 5, we study the links between geometric models and topological tools used in asynchronous computability theory. In particular, we explain how the protocol complex of Herlihy et al. [HS99] can be reconstructed directly from the geometric semantics of the protocols. The protocols complex can be obtained as some form of iterated subdivision of a complex corresponding to the possible inputs; we also study directly this simplicial complex and show that in the case where we start from the standard simplicial complex, the iterated subdivision is collapsible: this provides a new constructive and purely geometric proof of this fact, on which most impossibility results are based.

In Chapter 6, we describe a simple categorical model for another kind of practical concurrent system: distributed version control systems. Here, the operation of merging patches is naturally modeled using pushouts and we observe that some patches are conflicting, in the sense that they do not admit a pushout. In order to find a proper representation for files with conflicts, we investigate a free finite cocompletion of the model and provide a concrete description of it. This approach should help in solving many of the problems present in the current implementations of version control systems.

In Chapter 7, we give a general introduction to rewriting systems and the geometric techniques associated with them. Squier's theorems [Squ87, SOK94] are notably presented: one of them states that a monoid which admits a convergent rewriting system admits a finite homotopy basis, i.e. a finite set of generators for the equality between relations, for every presentation. We also recall the definition of polygraphs, as a higher-dimensional generalization of rewriting systems.

In Chapter 8, we extend the Knuth-Bendix completion algorithm in order to build presentations of monoids which are coherent, i.e. equipped with a homotopy basis. We also explain that adding superfluous generators can make the algorithm terminate in certain cases, and provide a method to reduce the size of those presentations. Our results are illustrated by experiments obtained from the implementation of a prototypical tool.

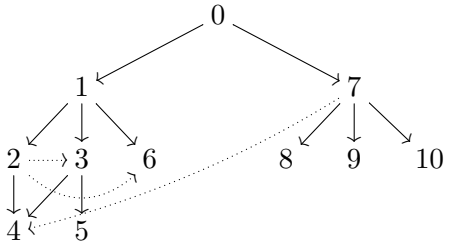
In Chapter 9, we introduce a notion of presentation modulo of a category. Traditionally, a category is presented as a free category on a graph quotiented by a relation on morphisms; here, we also allow a quotient on objects. We provide coherence conditions, so that the three natural ways of identifying objects (quotient, localization, and full subcategory on normal forms) coincide. We also hint at generalizations to presentations of 2-categories, which were the original motivation for this work.

In Chapter 10, we report on preliminary works toward the implementation of polygraphs, by finding a suitable data-structure to represent the morphisms they generate. We begin by illustrating our ideas on two simpler cases: the free category generated by a graph, and the free n-category on an n-globular set.

Topics. Each of those chapters is concerned with a specific *concurrent computing model*: our concurrent imperative programming language (Chapters 1 to 4), asynchronous protocols (which can be seen as a particular case of the former, extended with failures, Chapter 5), distributed version control systems (Chapter 6), or rewriting systems (Part II, Chapters 7 to 10). Each of them is also concerned with a *geometric model*: graphs and asynchronous graphs (Chapters 1 and 6), precubical sets (Chapters 2 and 4), topological models (Chapters 3 and 5), metric models (Chapters 3 and 4), simplicial models (Chapters 5 and 6), polygraphs (Part II, Chapters 7 to 10).

Even though the topics we address are apparently of quite diverse nature, especially those in the distinct parts of the manuscript, the tools used to work on them are often common: the use of category theory is pervasive, especially presheaf categories (Chapters 2 to 6 and 10), the cube property and residuation (Chapters 2, 4 and 9), the slice constructions for labeling (Chapters 2, 5, 6 and 10), the collapse sequences (Chapter 5) and the Tietze equivalences (Chapters 7 to 9) are essentially the same concept applied to simplicial complexes or presentations, the equivalence between quotient and localization in presentations modulo (Chapter 9) had already been investigated in order to define categories of components (Chapter 2), higher-dimensional rewriting systems (Chapter 7 and following) are the tool which allowed us to show the equivalence between homotopy and dihomotopy for non-positively curved precubical sets (Chapter 4), etc.

Reader's guide. The manuscript is divided in two parts, each of which contains an introductory chapter, and subsequent chapters which are more or less independent, so that the whole manuscript does not have to be read linearly. The partial order in which the chapters should be read is



(dotted lines indicate recommended, but not required, dependencies). Part I is mostly concerned with models for concurrency: Chapter 1 which presents the general ideas, Section 2.1 which recalls general properties of presheaf categories, and Section 2.2 which presents the precubical semantics have to be read before the remainder. Part II is concerned with extensions of rewriting theory, of which the necessary knowledge is recalled in Chapter 7.

Notations. Generally, notations are introduced at the point where they are first used. We only recall here the most important ones. We write $[n]$ for the ordinal set with n elements $[n] = \{0, 1, \ldots, n-1\}$, $\#X$ for the cardinal of a set X, $f : A \to B$ for a morphism from A to B, $s : x \twoheadrightarrow y$ for a path from a point x to a point y, and $\sim$ denotes the dihomotopy relation between paths.

0.3 Some personal remarks

A brief summary of my recent cursus. From 2005 to 2009, I have worked on my PhD thesis [Mim08] under the supervision of Paul-André Melliès in the PPS team of Paris 7. In one way or another, many of the topics I developed subsequently, some being presented here, were already in germ: the cubical models were called *asynchronous games* at that time, and were used to study linear logic instead of concurrent programs, but the spirit was the same. It was during this period that I learned about the cube property and its consequences on residuation, which turned out to be such a useful tool to tackle various problems I was faced with; I also discovered and began to work on higher-dimensional rewriting, which is a fascinating subject on which so much work remains to be done.

Starting from 2009 I began to work with Éric Goubault and Emmanuel Haucourt, as a post-doc and then as a permanent researcher, at the Commissariat à l'Énergie Atomique and at the École Polytechnique, where I gradually understood the interest of considering geometric models to study concurrent programs, including its practical aspects by contributing to the tool ALCOOL [H$^+$16]. In particular, the interest of resorting to topological models to study programs, which are discrete by nature, was quite unexpected for me, but turned out to be a remarkable source of new ideas and points of view.

As explained earlier, this manuscript does not mention some other work that I have done since my PhD thesis, notably on methods for verifying hybrid systems using abstract interpretation based on the domain of affine forms [BCM12, BCM14], models of hybrid systems using non-standard analysis [BM11], particular presentations of monoidal categories [Mim11, Mim15], and generators of musical streams for webradios [BM08, BBM11].

Acknowledgments. Thanks everyone! (this section will be filled later on...)

Part I

Geometric Models for Concurrency

Chapter 1

Modeling a concurrent language

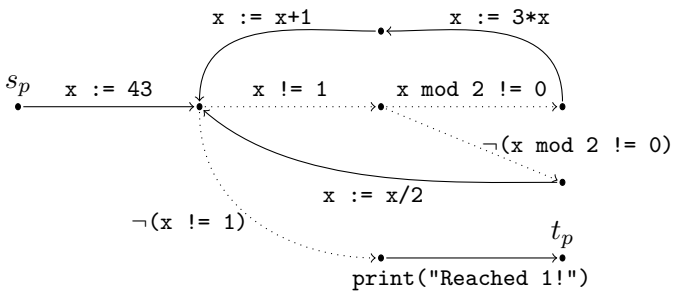
This chapter is mostly based on [FGH+16] (chapters 1 and 2).

After introducing the programming language we will use as an illustration in this manuscript, and recalling the notion of associated control flow graph (Section 1.1), we detail its concurrency primitives (Sections 1.2 and 1.3). We then introduce the asynchronous semantics of programs (Section 1.4) and hint at higher-dimensional generalizations (Section 1.5).

1.1 The control flow graph of a program

In this memoir, we will be interested in models of concurrent programs. In order to illustrate our discussion, we consider an imperative programming language such as IMP [Win93], constituted of arithmetic expressions manipulating integers, boolean expressions, *actions* of the form `x:=`e assigning to a variable `x` the evaluation of an arithmetic expression e, and control-flow constructions such as conditional branching and loops. For instance, the program on the left, computing the Syracuse sequence starting from 43. When analyzing such a program, in order to perform optimizations or study its possible executions, it is often natural and useful to consider the associated *control flow graph* (or CFG) as a representation. This is depicted on the right for our example:

```
x := 43;
while x != 1 do (
    if x mod 2 != 0 then
        (x := 3*x; x := x+1)
    else
        x := x/2
);
print("Reached 1!")
```



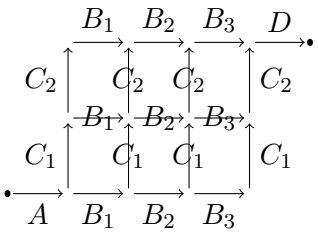
We write G_p for the CFG of a program p, V_p for its set of vertices (or *positions*), E_p for its set of edges (or *transitions*) and s_p (resp. t_p) for its beginning (resp. end) vertex. Its edges are labeled either by actions of the program or by boolean conditions, in which case they are drawn dotted. A most important feature of this graph is that the executions of the program correspond to some of the paths in this graph, starting from the initial vertex, such paths being called *feasible*. In order to verify that the program will not go wrong, one has to ensure

that each of these paths does not lead to an error (such as performing a division by zero), and various techniques have been developed in order to perform this, ranging from abstract interpretation tools [Min11] to proof-assistants such as COQ [BBC+97]. We do not detail these general techniques, because they are well-known and we are here mostly interested in the problems that are specific to concurrent languages

1.2 Concurrent programs

Since we are interested in concurrency aspects, we suppose that our programming language moreover contains a construction $p_1 \parallel p_2$ which means "execute the program p_1 in parallel with the program p_2", corresponding to parallel composition in theoretical languages (CCS, π-calculus [Mil89]) or to spawning threads in concrete ones. A program which does not use this construction is called *sequential*. For now, we assume that our language is *sequentially consistent*, meaning that "the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program", as phrased by Lamport [Lam79]. It is therefore natural to define the CFG of $p_1 \parallel p_2$ as the tensor product (or asynchronous product) $G_{p_1} \otimes G_{p_2}$ of the CFG of p_1 and p_2, whose paths will correspond to interleavings of paths in p_1 and p_2: its vertices are pairs (x_1, x_2) of vertices in G_{p_1} and G_{p_2}, and an edge $A : (x_1, x_2) \to (y_1, y_2)$ is either an edge $A : x_1 \to y_1$ in G_{p_1}, in which case $x_2 = y_2$, or an edge $A : x_2 \to y_2$, in which case $x_1 = y_1$. For instance, the program on the left has the graph on the right as CFG:

$$A\,;((B_1\,;B_2\,;B_3) \parallel (C_1\,;C_2))\,;D \qquad (1.1)$$



Here, we can already observe a first problem purely related to the presence of concurrency: the verification of concurrent programs is inherently combinatorially difficult.

Concurrency problem 1. The number of maximal execution traces can be exponential in the size of the program, even without loops. For instance, one easily observes that a program of the form $A \parallel A \parallel \ldots \parallel A$, with n copies of some action A in parallel, generates $n!$ maximal execution traces. This is called the *state-space explosion problem* [Cla09].

1.3 Resources

In practice, when two threads try to access a shared resource, such as memory, the result is often unspecified. For instance, consider the program

```
x:=0; (x:=x+1 || x:=x+1)
```
(1.2)

in which two threads concurrently increment a variable `x` whose value is initially 0. The first intuition is that after the execution of the program, the variable `x` should contain the value 2.

However, because of the way threads' accesses to memory are implemented in practice, it happens that the variable might also contain 1, or even a completely unrelated value (sequential consistency is not a valid assumption in practice!): in most programming languages concurrent access to shared memory is unspecified. In order to achieve reasonably predictable behavior when using shared memory, most operating systems provide a construction, called a *mutex* (short for *mut*ual *ex*clusion), which is a resource that can be held by at most one thread. Given such a mutex a, a thread can perform two operations on it [Dij68b]:

— *lock* the resource, which is modeled by the instruction P_a,
— *release* the resource, which is modeled by the instruction V_a.

The system guarantees that a mutex can be locked at most once simultaneously: if a thread tries to lock a resource that has previously been locked by another thread, it remains frozen until the mutex is released (if multiple processes are frozen, only one of them is awaken when the mutex is released). In order to guarantee predictable behavior, the program (1.2) should thus be rewritten as

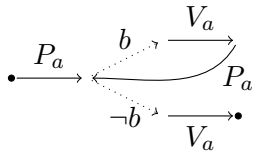
$$\texttt{x:=0; (}P_a\texttt{;x:=x+1;}V_a \;\|\; P_a\texttt{;x:=x+1;}V_a\texttt{)} \tag{1.3}$$

Another useful feature of mutexes is that they can be used to ensure that a sequence p of instructions is *atomic*, i.e. will never be interrupted to execute another instruction performed in parallel: in a subprogram of the form $P_a\texttt{;}p\texttt{;}V_a$, we know that the instructions in p will not be interleaved with instructions from other subprograms running in parallel which are also using the locking and unlocking the mutex a in the same way. The portion of code between P_a and V_a is thus called a *blocking section*. The operations P and V are often referred to as *synchronization primitives* because they help the programmer to regulate how threads will execute wrt each other, and make it easier to reason about concurrent programs.

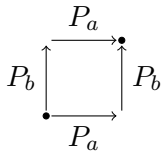
We thus extend once again our programming language to incorporate those constructions. We suppose fixed a set $\mathcal{R}$ of *resources* together with a function $\kappa : \mathcal{R} \to \mathbb{N}$ which, with every resource, associates its *capacity*, i.e. the number of threads that can lock it at once: a mutex is the particular case of a resource of capacity one, but other capacities are also useful. We also suppose that our language contains, for every resource a, two actions P_a and V_a whose intended effect was described above, which allows the implementation of most classical concurrent data-structures [Dow09]. In order to be able to efficiently study the resulting programs, we will need to make two assumptions on how those are used: conservativity and coherence (see below and Section 1.4). These are very often satisfied in practice and make the programs much more manageable theoretically.

Definition 1. A program is *conservative* when the number of times a resource has been released minus the number of times it has been locked depends only on the position in the CFG (and not on the execution path which has led to it).

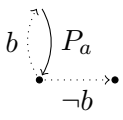
Example 2. The program $P_a\texttt{;(while }b\texttt{ do (}V_a\texttt{;}P_a\texttt{));}V_a$ is conservative, as well as $P_a \| P_b$,



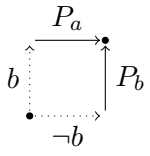
$P_a\texttt{;(while }b\texttt{ do (}V_a\texttt{;}P_a\texttt{));}V_a$



$P_a \| P_b$



$\texttt{while }b\texttt{ do }P_a$



$\texttt{if }b\texttt{ then }P_a\texttt{ else }P_b$

but not the program `while` b `do` P_a (the number of times a is taken in the end position depends on the number of loops) nor the program `if` b `then` P_a `else` P_b (which mutex is taken in the end position depends on the chosen branch).

The naming comes from an analogy with physics, where a force is said to be conservative when its action does not depend on the chosen path, but only on its endpoints. In this case (and we will see that this also applies to ours), it derives from a potential on simply connected parts of the space. For programs, this property can be shown to be equivalent to the fact that the resource consumption is well-defined:

Definition 3. Given a program p, its *resource consumption* $\Delta(p) : \mathcal{R} \to \mathbb{Z}$ gives, for each resource a, the number $\Delta(p)(a)$ of resources a it has taken or released (depending on whether this number is negative or positive), i.e. the difference between the number of V_a instructions and the number of P_a instructions encountered in an execution of p. It is defined by induction on p by

$$\begin{array}{rclcrcl}
\Delta(A) & = & 0 & & \Delta(\texttt{skip}) & = & 0 \\
\Delta(P_a) & = & -\delta_a & & \Delta(V_a) & = & \delta_a \\
\Delta(p;q) & = & \Delta(p) + \Delta(q) & & \Delta(p \parallel q) & = & \Delta(p) + \Delta(q) \\
\Delta(\texttt{if } b \texttt{ then } p \texttt{ else } q) & = & \Delta(p) & & \multicolumn{3}{l}{\text{whenever } \Delta(p) = \Delta(q)} \\
\Delta(\texttt{while } b \texttt{ do } p) & = & 0 & & \multicolumn{3}{l}{\text{whenever } \Delta(p) = 0}
\end{array}$$

where A is an arbitrary action. Above, 0 denotes the constant function whose image is 0, the addition of two functions is the pointwise addition and δ_a denotes the function such that $\delta_a(a) = 1$ and $\delta_a(b) = 0$ for any $b \neq a$. Notice that the function is only partially defined because of the side conditions in the cases of branching and loop.

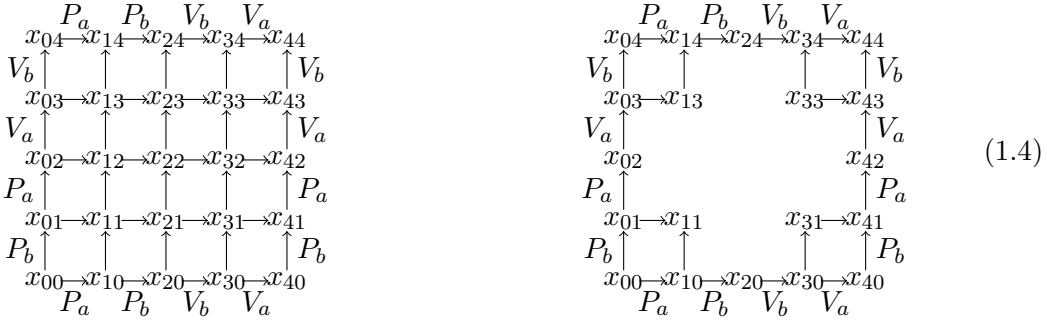
The above definition should make it clear that whether a program is conservative can be checked in linear time.

For a conservative program, the *resource potential* $r_p : V_p \to \mathcal{R} \to \mathbb{Z}$ is defined as the function which with every vertex x, and every resource a, associates $r_p(x)(a) = \kappa(a) + \Delta(u)(a)$, the *residual capacity* of a at position x, where u is any path from the initial vertex to x. The operational semantics of the programming language is so that a path ending in a position x such that $r_p(x)(a) < 0$ for some resource a is not feasible, because it would correspond to a situation where the resource has been locked more than what is authorized. Such a position is thus *forbidden* and it is thus natural to remove those positions from the CFG of the program.

Definition 4. The *pruned* CFG $\check{G}_p$ of a program p is its CFG restricted to vertices which are not forbidden.

Example 5 (Swiss flag). Consider the following program p: $P_a ; P_b ; V_b ; V_a \parallel P_b ; P_a ; V_a ; V_b$ with $a, b \in \mathcal{R}$ mutexes ($\kappa a = \kappa b = 1$), which is conservative and whose CFG G_p is drawn on the left. The vertex x_{12} is forbidden because we have $r_p(x_{12})(a) = -1$: in this position, the mutex a would have been locked twice. By removing this position, as well as other forbidden ones, we

obtain the pruned CFG $\check{G}_p$ depicted on the right:



(1.4)

This example is often called the *Swiss flag* because of the shape of the pruned CFG.

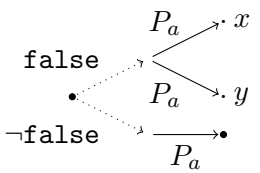
Two interesting observations can be made on the above example, in order to illustrate problems which can be detected on the CFG and are purely related to the presence of concurrency and resources. Notice that there is no edge starting from the vertex x_{11}. Such a vertex is called a *deadlock*: an execution reaching this position will be stalled forever, and will never be able to reach the terminal position x_{44}, essentially because the two threads are mutually waiting for the other to release a resource. This is generally the sign of a major conception problem in the program. Dually, the vertex x_{33} is not reachable from the initial position x_{00}: such a position is witnessing the presence of *dead code*. In a critical system, every single piece of code is usually written for some purpose, and the fact that some part of the code is formally useless is generally a good indicator of some misconception on the part of the programmer regarding the possible executions of the program.

Concurrency problem 2. A sensible program should have no deadlock (and, to a lesser extent, no dead code).

The detection of such deadlocks depends only on the resource-related actions performed by the program, which is why many articles focus on simplified languages containing only P and V as actions, without memory manipulations or computations on values [FGR98, GR02, Hau05b, GH05, FRG06, BH10, FGH+12].

Remark 6. The CFG actually allows one to detect *potential* deadlocks, i.e. positions which are deadlocks if they are reachable. In order to ensure that such a position is actually a deadlock, one should moreover ensure that it is reachable by a feasible path, i.e. one in which the conditions are satisfied, which can be done using standard analysis techniques for programs such as abstract interpretation. For instance, consider the program on the left, whose pruned CFG is shown on the right:

`if false then ((`P_a `||` P_a`);`V_a`) else` P_a



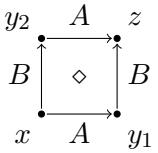
On the CFG, one can detect two potential deadlock points, x and y, however the program never actually deadlocks because no execution can reach those points: the condition `false` is never satisfied.

1.4 Asynchronous semantics

In order to address the state-space explosion problem, one should observe that some sequence of actions are observationally equivalent, meaning that they lead to the same state when they are executed in the same state. More formally, writing $\mathcal{V}$ for the set of variables, a *state* is an element of the set $\Sigma = \mathbb{Z}^{\mathcal{V}} \times \mathbb{Z}^{\mathcal{R}}$, consisting of a pair of functions associating an integer value with each variable, and its residual capacity with each resource. By induction, it is not difficult to assign, to each sequential program p, a function $[\![p]\!] : \Sigma \to \Sigma$, called its *denotational semantics*, which computes the state resulting from the execution of the program in a given initial state [Win93, FGH+16]. To be more precise, this function is only partially defined, since for instance $[\![$`y:=x/0`$]\!]$ is not defined, nor is $[\![P_a]\!]$ on states where the residual capacity of a is 0, and the definitions can actually be modified in order to distinguish between the two cases, the first one corresponding to an error and the other one to a feature of the language. Two programs p and p' are *observationally equivalent* when $[\![p]\!] = [\![p']\!]$: this means that they will evaluate in the same way, in whichever state. Noticing that a path in the CFG of a program can be seen as a program constituted of a sequence of actions, the observational equivalence relation can be extended to paths, and is very interesting because it allows one to reduces the number of paths to explore in order to verify a program: since two equivalent execution paths exhibit similar behaviors, it is enough to check one path in each equivalence class.

Observational equivalence on paths is undecidable however, which is why we are interested in equivalence subrelations, which are "as big as possible", and generated from "local" principles. One way to achieve this is to consider commutations relations:

Definition 7. Two actions A and B *commute* when $[\![B]\!] \circ [\![A]\!] = [\![A]\!] \circ [\![B]\!]$, or equivalently when for every CFG containing a subgraph of the form



(1.5)

the semantics of the two paths from x to z are the same.

Example 8. The actions `x:=2*x` and `x:=x+1` do not commute, but the actions `x:=2*x` and `y:=y+1` do. More generally, two actions using distinct variables do commute (the only conflicts in our language occur when a process reads on a variable while another one writes to it, or two processes write on the same variable).

This suggests that the information of which actions are commuting should be incorporated in the control flow graph: one should associate, with a program, not only a graph, but an *asynchronous graph*, i.e. a graph equipped with a set $\diamond$ of pairs of paths labeled by $A \,.\, B$ and $B \,.\, A$ and with the same source and the same target, as in (1.5), which are called *independence tiles*. The equivalence relation $\sim$ on paths, defined as the smallest congruence containing $\diamond$, is called *dihomotopy*, and is contained in observational equivalence. The equivalence class of a path is often called a *trace*.

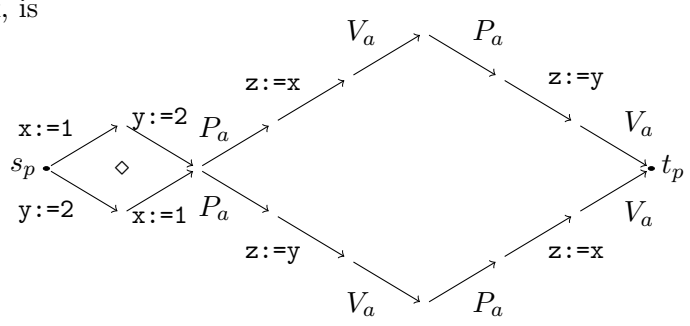
The construction for the CFG, sketched in previous sections, can be generalized in order to associate an asynchronous graph instead of a graph as follows: G_p is defined as before except that the tensor product $G_1 \otimes G_2$ of asynchronous graphs now adds an independence relation

for every paths of the form (1.5) whenever A is an edge of G_1 and B is an edge of G_2, or conversely. For instance, in (1.4), every square belongs to independence. The pruned CFG $\check{G}_p$ is obtained as before, by restricting (vertices, edges and independence) to vertices which are not forbidden, and is called its *asynchronous semantics*.

Example 9. The asynchronous semantics of the program

$$p \quad = \quad \texttt{(x:=1 } \parallel \texttt{ y:=2); (}P_a\texttt{;z:=x;}V_a \parallel P_a\texttt{;z:=y;}V_a\texttt{)}$$

where a is a mutex, is

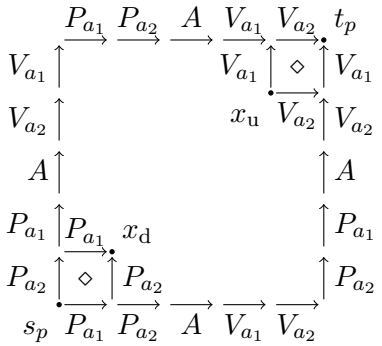


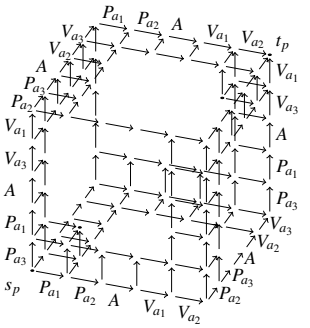
which, as expected, encodes the fact that the actions `x:=1` and `y:=2` commute, but the actions `z:=x` and `z:=y` do not.

Example 10 (Dining philosophers). Dijkstra's well-known dining philosophers example can be programmed as follows in our language [Hoa78]. We suppose given n mutexes a_i, and n processes p_i of the following form, executed in parallel:

$$p_i \quad = \quad P_{a_i}\,;P_{a_{i+1}}\,;A;V_{a_i}\,;V_{a_{i+1}}$$

where the indices i, above, are to be considered modulo n. For two and three processes, the pruned CFG are





with all the squares in the independence relation in the asynchronous graph on the right. For n philosophers, there are more than 2^{2n} states, and more than $2^{(n-1)^2}$ maximal traces, hence the state space and the path space are growing exponentially in the number of philosophers (another illustration of the state-space explosion problem). In comparison, there are only $2^n - 1$ classes of maximal traces up to dihomotopy, which is much less than the number of traces without the quotient, see [FGH+16].

We see that the dihomotopy relation can be quite useful in practice, by considering traces instead of execution paths. However, there is nothing that guarantees for now that the dihomotopy relation is included in observational equivalence. For instance, with $A = \texttt{x:=0}$ and $B = \texttt{x:=1}$, the asynchronous semantics of the program $A \parallel B$ is of the form (1.5), with the two paths being in the independence relation, but the actions A and B obviously do not commute. A program is coherent when this is the case:

Definition 11. A program is *coherent* when for every two dihomotopic paths $t, u : x \twoheadrightarrow y$ we have $[\![t]\!] = [\![u]\!]$.

There are various ways of ensuring that a program is coherent [FGH+16]: either by adding blocking sections to it, so that two actions accessing the same memory locations are mutually exclusive, or by checking that it is the case using traditional verification techniques.

Remark 12. This formalization presented here follows the general POSIX philosophy [Gro13], used in most languages, where the programmer has to ensure by himself, using synchronization primitives, that concurrent accesses to shared resources are impossible. Another sound approach would be to remove from the semantics all the independence tiles which are not semantically valid. For instance, there would be one path up to dihomotopy in $\texttt{x:=1} \parallel \texttt{y:=2}$, but two in $\texttt{x:=1} \parallel \texttt{x:=2}$.

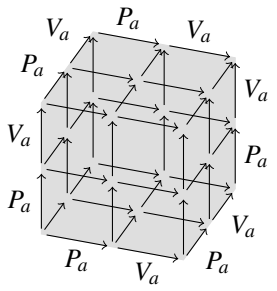
In the following, we will implicitly consider only program which satisfy both the conservativity and coherence condition.

Bibliographical references. The idea of taking in account commutation of actions in concurrent systems dates back to Mazurkiewicz [Maz88], which has led to the development of the theory of *trace monoids* [DR95, DM97], and the introduction of *asynchronous automata* by Zielonka as recognizing corresponding trace languages [Zie87]. Many variants of structures adapted to the study concurrent systems, with a state and possibly independent events, where then introduced and studied: *asynchronous transition systems* [Shi85, Bed88], *concurrent transition systems* [Sta89], *trace automata* [Sta90], *automata with concurrency relations* [Dro90], *transition systems with independence* [WN95]. A comparison between some of those models can be found in [WN95, SNW96, Sta90, HS96, Mor05]. The formulation given here is based on asynchronous graphs, and is close to labeled transition with independence; it was chosen because it generalizes well, as explained in next section. On the practical side, commutations between events have also been used as the starting point of partial order reduction techniques [God96], which are detailed in Section 2.5. Personally, I first encountered those models through Melliès' work on asynchronous game semantics [Mel04], which is based on asynchronous graphs and aims at studying the concurrency inherent to proofs in the framework of game semantics. I have further worked on this topic during my PhD thesis [Mim08], notably studying the links with event structures [MM07] and focusing [Mim10b].

1.5 Toward higher-dimensional models

The asynchronous semantics is more informative than the CFG because it takes in account whether two actions commute or not. However, there is no reason to stop at two actions, and one naturally more generally wants to know whether a given set of n actions commutes or not. For instance, in a program of the form $A \parallel B \parallel C$, it might happen that any pair of actions (A and B, A and C, B and C) commute but the three actions cannot be executed in

parallel, i.e. they do not "commute altogether". This is typically the case with the program $P_a;V_a \parallel P_a;V_a \parallel P_a;V_a$ where a is a resource such that $\kappa(a) = 2$: the associated pruned asynchronous transition graph is



(1.6)

where all the squares are filled, but the interior of the cube is empty. Notice that there is no vertex in the middle and the figure can thus be seen as a subdivided hollow cube. In order to formalize this, we will use a generalization of asynchronous graphs called precubical sets. An asynchronous graph consists of three kinds of objects: 0-dimensional ones (the vertices), 1-dimensional ones (the edges) and 2-dimensional ones (the independence tiles). A precubical set will consist of sets of n-dimensional cubes for each $n \in \mathbb{N}$, together with their faces. This will be detailed in next chapter. This generalization essentially amounts to having the possibility of distinguishing between a hollow n-cube from a filled one, generalizing the fact that the asynchronous graphs allow one to distinguish between a hollow (non-commuting) square from a filled one (commuting), thus exhibiting much richer and finer geometric invariants for programs.

Chapter 2

Cubical models for concurrency

The most convenient and widespread way to generalize the model of asynchronous graphs, in order to incorporate formal cubes of arbitrary dimension, is based on precubical sets. These are defined as the category of presheaves over a certain base category, and since we will consider other sorts of presheaves, we begin by a general introduction about presheaf categories and their properties (Section 2.1). We then introduce precubical sets and provide a semantics of concurrent programs using those (Section 2.2) and recall some useful classical theoretical constructions that can be performed on them (Section 2.3). Finally, we show that this model is related to, or generalizes previously known ones, by constructing adjunctions with most commonly used models for concurrency (asynchronous transition systems, event structures and Petri nets, see Section 2.4) and showing that the category of future components can be interpreted as a form of partial order reduction (Section 2.5).

2.1 Presheaf categories

We begin by introducing the general setting of presheaf categories. This summarizes all the basic properties we will need on those, which will be used in many places in this memoir.

Definition 13. Given a category $\mathcal{C}$, called a *base category*, we write $\hat{\mathcal{C}}$ for the category of *presheaves* over $\mathcal{C}$, i.e. functors $\mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$ and natural transformations between them.

Presheaf categories enjoy many interesting properties, we only briefly recall some of those that we will be using, and refer to standard textbooks for details [MM92]. We suppose fixed an arbitrary presheaf category $\hat{\mathcal{C}}$.

Limits and colimits. The category $\hat{\mathcal{C}}$ is always a topos: in particular, it has all small limits and colimits. Moreover, the limits (and similarly for colimits) can be computed pointwise: given a diagram $F : \mathcal{J} \to \hat{\mathcal{C}}$, we have for every object $A \in \mathcal{C}$ an induced diagram $F_A : \mathcal{J} \to \mathbf{Set}$, obtained by evaluating the presheaves in the image at A, and we have $(\lim F)(A) \cong \lim F_A$ (in fact, this property generalizes straightforwardly to any functor category $\mathcal{D}^{\mathcal{C}}$ with $\mathcal{D}$ (co)complete).

Representable presheaves. Any object $B \in \mathcal{C}$ induces a presheaf $\mathcal{C}(-, B) : \mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$, which associates the set $\mathcal{C}(A, B)$ with an object A, and is induced by pre-composition on morphisms. The *Yoneda functor* $Y : \mathcal{C} \to \hat{\mathcal{C}}$ associates the presheaf $\mathcal{C}(-, A)$ with an object A, and is defined on morphisms by post-composition. A presheaf of the form YA, for some object

$A \in \mathcal{C}$, is called *representable*. The celebrated *Yoneda lemma* [Yon54] asserts that the elements of a presheaf $P \in \hat{\mathcal{C}}$ are classified by the morphisms originating from the representables:

Lemma 14 (Yoneda)**.** *Given an object $A \in \mathcal{C}$, we have an isomorphism $P(A) \cong \hat{\mathcal{C}}(YA, P)$.*

In particular, given two objects $A, B \in \mathcal{C}$, we have $\mathcal{C}(A, B) = YBA \cong \hat{\mathcal{C}}(YA, YB)$, which shows that the Yoneda functor is always full and faithful.

Realizations. We will see below that every presheaf is canonically a colimit of representables. Because of this, in order to define a functor $\hat{\mathcal{C}} \to \mathcal{D}$ to a cocomplete category, it often suffices to define this functor on the representables and extend it to all presheaves by colimit. This methodology includes in particular, the definition of the geometric realization of cubical/simplicial/etc. sets. It can be formalized as follows.

Definition 15. Given a presheaf $P \in \hat{\mathcal{C}}$, its *category of elements* $\mathrm{El}(P)$ is the category whose objects are pairs (A, x) with $A \in \mathcal{C}$ and $x \in P(A)$ and morphisms $f : (A, x) \to (B, y)$ are morphisms $f : A \to B$ in $\mathcal{C}$ such that $P(f)(y) = x$.

The category $\mathrm{El}(P)$ can also be more abstractly defined as the comma category Y/P. Notice that there is a projection functor $\pi : \mathrm{El}(\mathcal{C}) \to \mathcal{C}$. Now, suppose given a functor $F : \mathcal{C} \to \mathcal{D}$ (which should be thought of as providing the realization FA of the representable presheaf YA, for any object $A \in \mathcal{C}$). This functor induces a *nerve functor* $N_F : \mathcal{D} \to \hat{\mathcal{C}}$ defined on an object $B \in \mathcal{D}$ by $N_F B = \mathcal{D}(F-, B)$ (this generalizes the above definition of the Yoneda functor, which is the particular case where $F = \mathrm{Id}_{\mathcal{C}}$).

Proposition 16. *Given a functor $F : \mathcal{C} \to \mathcal{D}$ from a small category $\mathcal{C}$ to a cocomplete category $\mathcal{D}$, the nerve functor $N_F : \mathcal{D} \to \hat{\mathcal{C}}$ admits a left adjoint $R_F : \hat{\mathcal{C}} \to \mathcal{D}$, called the* realization along F*, which is defined on objects $P \in \hat{\mathcal{C}}$ by*

$$R_F(P) \quad = \quad \mathrm{colim}\left(\mathrm{El}(P) \xrightarrow{\pi} \mathcal{C} \xrightarrow{F} \mathcal{D}\right)$$

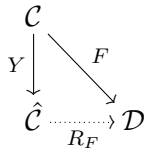
It can also be described as $\mathrm{Lan}_Y F$, the left Kan extension of F along Yoneda, see (2.1)*.*

In the particular case with $\mathcal{D} = \hat{\mathcal{C}}$ and $F : \mathcal{C} \to \hat{\mathcal{C}}$ the Yoneda functor, the associated nerve $N_Y : \hat{\mathcal{C}} \to \hat{\mathcal{C}}$ is defined, for $P \in \hat{\mathcal{C}}$ and $A \in \mathcal{C}$, by $N_Y PA = \hat{\mathcal{C}}(YA, P) \cong P(A)$ and thus N_Y is isomorphic to the functor $\mathrm{Id}_{\hat{\mathcal{C}}}$. Its left adjoint is thus necessarily isomorphic to the identity functor, and we deduce $P \cong \mathrm{colim}\left(\mathrm{El}(P) \xrightarrow{\pi} \mathcal{C} \xrightarrow{Y} \hat{\mathcal{C}}\right)$: this shows that every presheaf is canonically a colimit of representable functors, the diagram being given by the category of elements of the presheaf.

A free cocompletion. The category $\hat{\mathcal{C}}$ is the free cocompletion of the category $\mathcal{C}$. Namely, the situation can be thought of as follows: the forgetful functor from the category of cocomplete categories and cocontinuous functors to the category of categories has a left adjoint (sending $\mathcal{C}$ to $\hat{\mathcal{C}}$ and F to $R_{Y \circ F}$), and the unit of the adjunction is given by Yoneda functors. This explanation is however bound to remain informal, because of size issues (cocomplete categories are not generally small), and a proper formulation is the following.

Proposition 17. *Given a functor $F : \mathcal{C} \to \mathcal{D}$ from a small category $\mathcal{C}$ to a cocomplete category $\mathcal{D}$, the realization functor $R_F : \hat{\mathcal{C}} \to \mathcal{D}$ of Proposition 16 is, up to isomorphism,*

the only cocontinuous (i.e. colimit-preserving) functor which makes the following diagram commute:



(2.1)

Variants of the notion of cocompletion of a category are studied in details in Chapter 6.

Restrictions. Suppose given a functor $F : \mathcal{C} \to \mathcal{D}$, which can typically be thought of as exhibiting $\mathcal{C}$ as a full subcategory of $\mathcal{D}$. The "restriction along F" admits both a left and a right adjoint which can be described as follows. Any presheaf $P \in \hat{\mathcal{D}}$ induces a presheaf $P \circ F^{\mathrm{op}} \in \hat{\mathcal{C}}$, and this operation extends as a functor $F^* : \hat{\mathcal{D}} \to \hat{\mathcal{C}}$, defined on morphisms $\phi : P \to Q$ by $(F^*\phi)_A = \phi_{FA}$. Alternatively, this functor can be defined as $F^* = N_{Y \circ F}$. If we suppose that the category $\mathcal{C}$ is small, this functor admits a left adjoint $R_{Y \circ F}$, often noted $F_!$, which can also be described as the left Kan extension functor $\mathrm{Lan}_{F^{\mathrm{op}}} : \hat{\mathcal{C}} \to \hat{\mathcal{D}}$ along F^{op}. This last formulation makes it clear that, dually, it also admits a right adjoint $\mathrm{Ran}_{F^{\mathrm{op}}}$, often noted F_* (these Kan extensions exist because the category **Set** is both cocomplete and complete):

Proposition 18. *Given a functor $F : \mathcal{C} \to \mathcal{D}$ with $\mathcal{C}$ small, the restriction functor $F^* : \hat{\mathcal{D}} \to \hat{\mathcal{C}}$ admits both a left and a right adjoint.*

Tensor product. It happens quite often that the base category $\mathcal{C}$ on which we consider presheaves is monoidal, with tensor denoted $+$ and unit 0. In this case, one can canonically equip the category $\hat{\mathcal{C}}$ with a structure of (closed) monoidal category, making the Yoneda embedding $Y : \mathcal{C} \to \hat{\mathcal{C}}$ strong monoidal [Day70]. The *Day tensor product* $P \otimes Q$ of two presheaves $P, Q \in \hat{\mathcal{C}}$ is defined, for $A \in \mathcal{C}$, by

$$(P \otimes Q)(A) \quad = \quad \int^{B,C \in \mathcal{C}} \mathcal{C}(A, B + C) \times P(B) \times Q(C) \tag{2.2}$$

and admits $Y(0)$ as unit. This tensor product can be seen as a cocontinuous extension of the one on $\mathcal{C}$, in the sense of Proposition 17 (generalized to bifunctors).

Graphs. In order to illustrate some of the previous concepts, consider the category of *graphs*. This can be defined as the category $\hat{\mathcal{G}}$ of presheaves over the category

$$\mathcal{G} \quad = \quad 0 \underset{t}{\overset{s}{\rightrightarrows}} 1$$

with two objects and two non-trivial arrows. A graph $G \in \hat{\mathcal{G}}$ thus consists of two sets $G(0)$, its *vertices*, and $G(1)$, its *edges*, together with two functions $G(s), G(t) : G(1) \to G(0)$ associating to each edge its source and target respectively.

The representable graphs $Y0$ and $Y1$ are respectively $\cdot$ and $\cdot \to \cdot$, for which the Yoneda lemma can be easily verified directly. Consider the functor $F : \mathcal{G} \to \mathbf{Top}$ such that $F(0)$ is a point, $F(1)$ is the interval $I = [0, 1]$ equipped with the euclidean topology, and $F(s)$ (resp. $F(t)$) sends the point of $F(0)$ to the point 0 (resp. 1) of $F(1) = I$. The associated realization functor $R_F : \hat{\mathcal{G}} \to \mathbf{Top}$ is called the *geometric realization* functor and associates,

with each graph, the corresponding topological graph, i.e. the "same" graph, but "drawn" as a topological space. Conversely, the associated nerve $N_F : \mathbf{Top} \to \hat{\mathcal{G}}$ associates, with each topological space X, the graph whose points are points in X, and edges are paths in X (i.e. continuous functions $I \to X$), with the expected source and target. Now consider the terminal category $\mathcal{G}_0$ and denote by $F : \mathcal{G}_0 \to \mathcal{G}$ the functor sending the object of $\mathcal{G}_0$ to the object 0 of $\mathcal{G}$. The associated restriction functor $F^* : \hat{\mathcal{G}} \to \hat{\mathcal{G}}_0 \cong \mathbf{Set}$ sends a graph to its set of vertices. Its left (resp. right) adjoint $\mathbf{Set} \to \hat{\mathcal{G}}$ sends a set X to the discrete graph, with no edge, (resp. the complete graph) with X as vertices. There is no monoidal structure on $\mathcal{G}$ giving rise to a meaningful one on presheaves, and this concept will therefore be illustrated on the next example.

Presimplicial sets. The fundamental example of presheaf category is the category of simplicial sets, whose use is pervasive in algebraic topology [Hat02, GJ09]. We present here the simpler variant of presimplicial sets. The *presimplicial category* $\triangle$ has natural numbers as objects and morphisms $f : m \to n$ are strictly increasing functions $f : [m] \to [n]$ where, given a natural number n, we write $[n]$ for the ordinal set $\{0, \ldots, n-1\}$ with n elements. A presheaf S over this category is called an (*augmented*) *presimplicial set*. The elements of $S(n)$ are called *n-simplices* and the functions between those are generated by $\partial_i : S(n+1) \to S(n)$, for $i \in [n+1]$, associating to a simplex its i-th face. Given $n \in \triangle$, the representable presheaf Yn has, as k-simplices, the sets $\{j_0, \ldots, j_{k-1}\} \subseteq [n]$ such that $j_0 < j_1 < \ldots < j_{k-1}$ and the i-th face of such a simplex is given by removing the element j_i. If we exclude the object 0 from the category $\triangle$, the resulting presheaves are called *non-augmented presimplicial sets*. The traditional *geometric realization* of a non-augmented presimplicial set is its realization along the functor sending an object n to the standard n-simplex, which is the subspace of $\mathbb{R}^{n+1}$ consisting of points $\{(x_0, \ldots, x_n) \mid \sum_i x_i = 1\}$. The *tensor product* of two presimplicial sets S and T has n-simplices $(S \otimes T)(n) = \coprod_{j+k=n} S(j) \times T(k)$ and the i-th face of a simplex $(x, y) \in S(j) \times T(k)$ is $(\partial_i(x), y)$ if $i \in [j]$ and $(x, \partial_{i-j}(y))$ otherwise.

2.2 The precubical semantics

2.2.1 Precubical sets

The main presheaf category we will be considering is the following one. It can be seen as a variant of the presimplicial sets based on squares instead of triangles (and more generally cubes instead of simplices).

Definition 19. The *precubical category* $\square$ is the category whose objects are natural numbers. A morphism $f : m \to n$ is a word of length n over the alphabet $\{-, +, 0\}$ containing m occurrences of 0 (note that this implies $m \leqslant n$). The composite of two morphisms $f : m \to n$ and $g : n \to p$ is obtained by replacing in g the i-th occurrence of 0 by the i-th letter of f for every $i \in [n]$.

Lemma 20. *The precubical category is the free category with natural numbers as objects, and morphisms generated by $\varepsilon_i^\epsilon : n \to n+1$ with $\epsilon \in \{-, +\}$, with $n \in \mathbb{N}$ and $0 \leqslant i \leqslant n$, subject to the relations*

$$\varepsilon_i^{\epsilon'} \varepsilon_j^{\epsilon} \quad = \quad \varepsilon_{j-1}^{\epsilon} \varepsilon_i^{\epsilon'} \tag{2.3}$$

whenever $i < j$ and $\epsilon, \epsilon' \in \{-, +\}$.

Definition 21. The category of *precubical sets* is the category $\hat{\square}$ of presheaves over this category.

A precubical set $C \in \hat{\square}$ thus consists of a family of sets $(C(n))_{n\in\mathbb{N}}$ together with maps $\partial_i^\epsilon : C(n+1) \to C(n)$, with $0 \leqslant i \leqslant n$ and $\epsilon \in \{-,+\}$, where the map ∂_i^ϵ is a notation for $C(\varepsilon_i^\epsilon)$, satisfying relations which are dual to (2.3). An element $c \in C(n)$ is called an *n-cube of C*, $\partial_i^\epsilon(c)$ is called a *face* of c (a *lower*, resp. *upper* face, when $\epsilon = -$, resp. $\epsilon = +$), and given a morphism $\phi : n \to p$ in $\square$, $C(\phi)(c)$ is called an *iterated face* of c. The *dimension* of a precubical set C is the smallest natural number $d \in \mathbb{N} \sqcup \{\infty\}$ such that $C(n) = \emptyset$ for $n > d$. A precubical set C is *finite* when it is finite-dimensional and each $C(n)$ is finite. The representable precubical sets are studied in Section 4.1.1.

Example 22. We write $\vec{I}$ for the precubical set of dimension 1, called the *standard interval*: $x \xrightarrow{a} y$, with $\vec{I}(0) = \{x, y\}$, $\vec{I}(1) = \{a\}$, $\vec{I}(n) = \emptyset$ for $n \geqslant 2$, $\partial_0^-(a) = x$, $\partial_0^+(a) = y$.

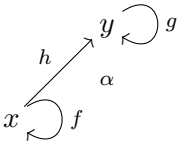
Example 23. The geometric intuition underlying precubical sets is the following one. An n-cell x of a cubical set should be seen as an n-dimensional cube, the $(n-1)$-dimensional cubes $\partial_i^-(x)$ and $\partial_i^+(x)$ being respectively the source and target in dimension i of x. So for example, a "cylinder" can be described as a precubical set C with

$$C(0) = \{x, y\} \qquad C(1) = \{f, g, h\} \qquad C(2) = \{\alpha\} \qquad C(n) = \emptyset \quad \text{for } n > 2$$

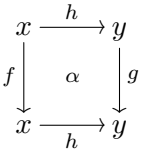
with the following sources and targets, given by

$$\begin{array}{lll} \partial_1^-(\alpha) = f & \partial_0^-(\alpha) = h & \partial_0^-(f) = \partial_0^+(f) = \partial_0^-(h) = x \\ \partial_1^+(\alpha) = g & \partial_0^+(\alpha) = h & \partial_0^-(g) = \partial_0^+(g) = \partial_0^+(h) = y \end{array}$$

This cylinder can be pictured graphically as



or



(in an unfolded representation)

Examples with higher-dimensional cubes can easily be given, but they are harder to draw...

The base category for graphs $\mathcal{G}$ can be recovered as the full subcategory of $\square$ on the objects 0 and 1. The inclusion functor $\mathcal{G} \to \square$ induces a restriction functor $\hat{\square} \to \hat{\mathcal{G}}$: every precubical set C has an underlying graph, and for this reason we sometimes call the elements of $C(0)$ (resp. $C(1)$), *vertices* (resp. *edges* or *transitions*). The elements of $C(2)$ are sometimes called *squares*.

One can notice that the category $\square$ is monoidal, with tensor product being given on objects by addition and "concatenation" on morphisms. With this monoidal structure in mind, the category $\square$ has the following third description.

Lemma 24. *The precubical category is the free monoidal category containing an object* 1 *and two morphisms $\varepsilon^-, \varepsilon^+ : 0 \to 1$, where* 0 *denotes the unit of the monoidal category.*

In addition of providing a concise description of $\square$, the above lemma has the following consequence. Given a monoidal category $\mathcal{C}$ with I as unit, a *co-precubical object* (C, e^-, e^+) consists of an object $C \in \mathcal{C}$ together with two morphisms $e^-, e^+ : I \to C$. The category of monoidal functors $\square \to \mathcal{C}$ and monoidal natural transformations is equivalent to the category of co-precubical objects and their morphisms. Moreover, this monoidal structure induces one on precubical sets by (2.2), which can be described explicitly as follows:

Definition 25. Given two precubical sets C and D, their *tensor product* $C \otimes D$ is the precubical set defined by

$$(C \otimes D)(n) \quad = \quad \coprod_{i+j=n} C(i) \times D(j)$$

for $n \in \mathbb{N}$, and

$$\partial^\epsilon_k(C \otimes D)(c,d) \quad = \quad \begin{cases} (\partial^\epsilon_k(c), d) & \text{if } 0 \leqslant k < i \\ (c, \partial^\epsilon_{k-i}(d)) & \text{if } i \leqslant k < n \end{cases}$$

for $n \in \mathbb{N}$, $0 \leqslant k < n$, $\epsilon \in \{0,1\}$ and $(c,d) \in C(i) \times D(j) \subseteq (C \otimes D)(n)$ with $i + j = n$. This tensor equips the category $\hat{\square}$ with a monoidal structure, whose unit is the precubical set C reduced to one 0-cube.

This tensor product generalizes in particular the one described on graphs in Section 1.1: the restriction functor $\hat{\square} \to \hat{\mathcal{G}}$ is monoidal.

2.2.2 The precubical semantics

In order to generalize the constructions of Chapter 1, we consider a variant of precubical sets whose edges are labeled.

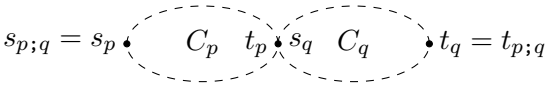
Definition 26. Given a set $\mathcal{L}$ of *labels*, a *labeled precubical set* consists of a precubical set $C \in \hat{\square}$ together with a function $\ell : C(1) \to \mathcal{L}$, associating a label to each edge, such that for every 2-cube $x \in C(2)$, any two parallel edges have the same label: $\ell \circ \partial^-_i(x) = \ell \circ \partial^+_i(x)$ for $i \in [2]$.

It can be checked that most constructions (colimits, tensor product) extend in the expected way to the labeled case.

In the same way as in Chapter 1, one can associate, with each program p, a precubical set C_p whose edges are labeled by actions or conditions:

— an action is still interpreted as the standard interval, labeled by the action,

— control-flow operations are interpreted using colimits, for instance $C_{p;q}$ is obtained from C_p and C_q by identifying the end vertex t_p of the former with the source vertex s_q of the latter:



— parallel composition is interpreted by the tensor product described above: we have $C_{p \parallel q} = C_p \otimes C_q$.

Definition 27. The *precubical semantics* $\check{C}_p$ of a program p is the labeled precubical set obtained as the sub-precubical set of C_p whose n-cubes are those which do not have a forbidden vertex as iterated face.

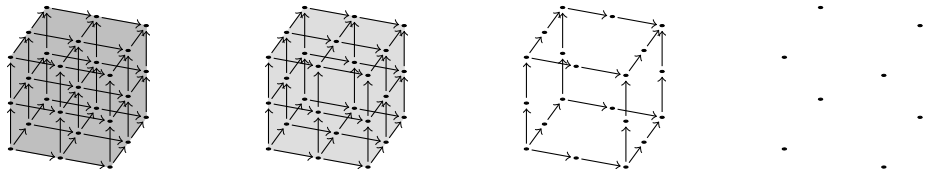
Example 28. Consider the program p which is $P_a ; V_a \parallel P_a ; V_a \parallel P_a ; V_a$. Depending on the capacity of the resource a, the cubical semantics $\check{C}_p$ of p is as follows:

— if $\kappa(a) = 3$ then $\check{C}_p$ is a subdivided filled cube,

— if $\kappa(a) = 2$ then $\check{C}_p$ is a subdivided hollow cube,

— if $\kappa(a) = 1$ then $\check{C}_p$ is a subdivided "skeletal" cube reduced to its edges,

— if $\kappa(a) = 0$ then $\check{C}_p$ only consists of 8 vertices.



$\kappa(a) = 3$ $\kappa(a) = 2$ $\kappa(a) = 1$ $\kappa(a) = 0$

The precubical semantics is a labeled precubical set equipped with a distinguished beginning vertex. Such a *pointed precubical set* is sometimes also called an *higher-dimensional automaton* (or HDA) and those have been the subject of many autonomous studies, notably studying various notion of bisimulation between them [Pra91, vG91, vG06, FL13, FL14].

Notice that the precubical semantics of a program is always finite and thus satisfies the following property, which will be of interest when studying the links with geometric realization in Section 3.1:

Definition 29. A precubical set is *locally finite* if for every vertex x, there are only finitely many cubes having x as iterated face.

2.2.3 Variants of precubical sets

The notion of precubical set admits many variants, as detailed in [GM03], for which the above cubical semantics can be adapted in order to capture in detail various aspects of programs.

Cubical sets. Cubical sets are akin to precubical sets, except that cubes can be degenerate: some of the edges can be identities, etc. The *cubical category* $\boxdot$ can be characterized as the free monoidal category on an object 1, with morphisms generated by $\varepsilon^-, \varepsilon^+ : 1 \to 0$ and $\eta : 0 \to 1$ subject to the relations $\varepsilon^- \circ \eta = \varepsilon^+ \circ \eta = \mathrm{id}_0$. Presheaves on this category are called *cubical sets*. Every cubical set $C \in \hat{\boxdot}$, is thus equipped with functions $\iota_i : C(n) \to C(n+1)$, for $i \in [n+1]$, sending an n-cube to the corresponding $(n+1)$-cube degenerated in dimension i.

The relation between precubical sets and cubical sets can be thought of as a generalization of the following situation on sets. We write $\mathbf{Set}'$ for the category of sets and partial functions and $\mathbf{Set}^*$ for the category of pointed sets. The forgetful functor $\mathbf{Set}^* \to \mathbf{Set}$ has a left adjoint and we write ? for the associated monad on $\mathbf{Set}$. It is well-known that the category $\mathbf{Set}'$ is isomorphic to the Kleisli category $\mathbf{Set}_?$ and equivalent to the category $\mathbf{Set}^*$: this roughly says that a partial function $f : A \to B$ can also be seen as a total function $f : A \to B \sqcup \{*\}$ where $f(a) = *$ means corresponds to f not being defined on $a \in A$. Similarly, we write $\mathbf{PCSet} = \hat{\square}$ for the usual category of precubical sets, $\mathbf{PCSet}'$ for the category of precubical sets and partial functions (satisfying suitable axioms), and $\mathbf{CSet} = \hat{\boxdot}$ for the category of cubical sets. The inclusion $\square \to \boxdot$ induces a restriction functor $\mathbf{CSet} \to \mathbf{PCSet}$, which admits a left adjoint, and we write ? for the corresponding monad on $\mathbf{PCSet}$.

Proposition 30 ([GM12a])**.** *The category* $\mathbf{PCSet}'$ *is isomorphic to the Kleisli category* $\mathbf{PCSet}_?$ *and equivalent to the category* $\mathbf{CSet}$*.*

This is useful in practice, since it can often be used to show that an adjunction between precubical sets and some other model extends to an adjunction between cubical sets and the other model with partial morphisms, by only showing that it is compatible with the corresponding comonad, based on general theorems for lifting adjunctions [Mul94]. This is explained in [GM12a] and will not be detailed here.

Symmetric precubical sets. In models of concurrent programs, n-cubes with $n \geqslant 2$, express the fact that n actions are independent. This independence relation is not fundamentally directed: by permuting the directions in which the actions are performed, the actions should remain independent. This means that whenever we have a cube, there is another cube with the same edges as faces but in other directions, and this for every permutation of the directions. In order for the models to retain good properties, one should keep track of these relationship between the cubes (otherwise, the models contain many artificial "holes" due to symmetries). This motivates the following variant of precubical sets. The *symmetric precubical category* $\square_S$ is the free symmetric monoidal category on two generators $\varepsilon^-, \varepsilon^+ : 1 \to 0$, and *symmetric precubical sets* are presheaves on this category. A symmetric precubical set thus consists of a precubical set C equipped with a suitable action on $C(n)$ of $\mathfrak{S}_n$, the symmetric group on n elements, for each $n \in \mathbb{N}$. A notion of *symmetric cubical set* can be defined similarly.

Labeled precubical sets. The category of labeled precubical sets, see Definition 26, is most elegantly defined using a slice category construction, which generalizes easily to other variants. The category **Set** is monoidal when equipped with the cartesian structure, and every set L is equipped with a structure of co-precubical object, with $\varepsilon^-, \varepsilon^+ : L \to 1$ both being the terminal arrow. By Lemma 24, this induces a precubical set $!\,L$ and this operation extends to a functor $! : \mathbf{Set} \to \hat{\square}$. More explicitly, the n-cubes of $!\,L$ are words over L of length n, and the i-th source and target of such a word are equal and obtained by removing the i-th letter of the word.

Definition 31 ([GM12a])**.** The category of *labeled precubical sets* is the comma category $\hat{\square}/\,!$.

This definition agrees with the one we have given earlier, and easily generalizes to the other variants. It is sometimes not desirable to have two parallel cubes with the same label: locally, a label should determine a cube. A labeled precubical set (C, ℓ) is *strongly labeled* when for every two n-cubes $x, y \in C(n)$ such that $\ell(x) = \ell(y)$ and $\partial_i^\epsilon(x) = \partial_i^\epsilon(y)$ for every $i \in [n]$ and $\epsilon \in \{-, +\}$, we have $x = y$.

Truncated precubical sets. For simplicity, or in order to compare to other low-dimensional models, one is sometimes not interested in k-cubes for k greater than some fixed n. We write $\square_n$ for the full subcategory of $\square$ whose objects are in $[n+1]$. The presheaves in $\hat{\square}_n$ are called *n-truncated precubical sets*. As explained before, the canonical inclusion $\square_n \to \square$ induces a *truncation functor* $\hat{\square} \to \hat{\square}_n$ which admits both adjoints: this means that any adjunction with $\hat{\square}_n$ extends to an adjunction with $\hat{\square}$. Other variants can be truncated in the same way.

2.3 Constructions on the precubical semantics

In this section, we recall some classical terminology as well as theoretical constructions on precubical sets which will be used later on.

2.3.1 The fundamental category and groupoid

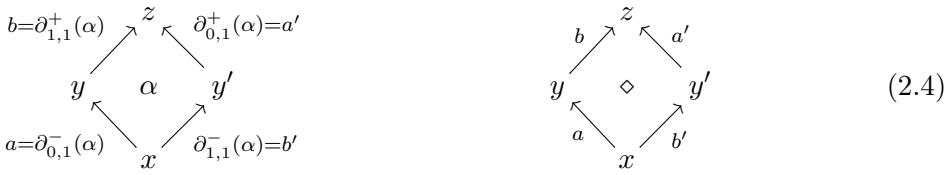
Suppose fixed a precubical set C. It has an underlying graph, and a path in this graph will be called a *dipath* (for *directed path*): it consists of a sequence $f_1, \ldots, f_k$ of edges such

that $\partial_0^+(f_i) = \partial_0^-(f_{i+1})$. From now on, we will always refer to those as "dipaths" and keep the denomination *path* for a path in the underlying non-directed graph: a path in C thus consists of a composable sequence of edges, which are possibly reversed (i.e. "taken backwards"). We write $s : x \twoheadrightarrow y$ to indicate that s is a path from a vertex x to a vertex y, and $\overline{s} : y \twoheadrightarrow x$ for the corresponding reversed path. An analogue of the independence relation in asynchronous graphs can be recovered as follows:

Definition 32. We define a symmetric relation $\diamond$ on dipaths of length 2 as the smallest symmetric relation such that, given two paths $a \,.\, b$ and $b' \,.\, a'$ of length 2 consisting of edges a, b, a' and b', we have $a \,.\, b \diamond b' \,.\, a'$ whenever there exist a square α such that

$$a = \partial_{0,1}^-(\alpha) \qquad b = \partial_{1,1}^+(\alpha) \qquad b' = \partial_{1,1}^-(\alpha) \qquad a' = \partial_{0,1}^+(\alpha)$$

or symmetrically. Graphically, we have a square α as on the left, what we often schematically picture as on the right



(2.4)

This relation is extended to (non-directed) paths of length 2 by imposing that, whenever there is a square as above, we have $a \,.\, b \diamond b' \,.\, a'$, $\overline{b} \,.\, \overline{a} \diamond \overline{a'} \,.\, \overline{b'}$, $\overline{a} \,.\, b' \diamond b \,.\, \overline{a'}$ and $\overline{b'} \,.\, a \diamond a' \,.\, \overline{b}$.

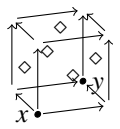
Definition 33. We write $\approx$ for the smallest equivalence relation on edges such that $a \approx a'$ whenever there exists edges b, b' for which $a \,.\, b \diamond b' \,.\, a'$. Its equivalence classes are called *events*.

Two edges which are in the same event are called *parallel*. Such edges can be thought of as different instances of a same event: for example, in the precubical semantics of programs, different schedulings of the same action are parallel.

Generalizing the definitions of Section 1.4, a notion of (di)homotopy can also be defined:

Definition 34. The *dihomotopy* relation $\overset{\diamond}{\sim}$ on paths is the smallest congruence wrt to concatenation which contains $\diamond$. The *homotopy* relation $\sim$ is the smallest congruence which contains $\diamond$ and such that $a \,.\, \overline{a} \sim \mathrm{id}_x$ and $\overline{a} \,.\, a \sim \mathrm{id}_y$ for every edge $a : x \to y$. The equivalence class of a path s is sometimes written $[s]$ and called a *trace*.

Example 35 (Cube without bottom). Consider the 2-dimensional precubical set below, where all the squares are filled, except the bottom one:



The two dipaths from x to y are homotopic but not dihomotopic. This is detailed in Example 117.

These notions should of course be thought of as the counterpart on precubical sets of the classical ones on (directed) topological spaces, see Section 3.1 for more details, and share many properties with those. In particular, the dihomotopy relation preserves the direction (and length) of paths: a path which is dihomotopic to a dipath is necessarily a dipath. Moreover two dihomotopic dipaths contain occurrences of the same events.

Definition 36. The *fundamental category* $\vec{\Pi}_1(C)$ is the category whose objects are the vertices of C and morphisms are dipaths up to dihomotopy. The *fundamental groupoid* $\Pi_1(C)$ is the category with same objects and paths up to homotopy as morphisms.

Higher-dimensional generalizations of these notions are introduced and studied in Section 4.3.1.

2.3.2 Unfolding

Loops are notoriously difficult to handle in geometric models; for instance, many constructions are only defined in the loop-free case (from practical ones such as verification techniques to theoretical ones such as the notion of category of components). Fortunately, one can always "unfold" loops in order to get rid of them as follows, at the cost of potentially producing an infinite model. It is still useful in practice in order to study programs up to a certain depth of loops. This unfolding process is also well-studied for many other models for concurrency such as transition systems [WN95] or Petri nets [NPW81, Eng91, MMS97, HW08]. A *pointed* precubical set (resp. asynchronous graph) consists of a precubical set (resp. asynchronous graph) together with a distinguished vertex: in precubical semantics of programs, this vertex typically corresponds to the beginning of the program. In the case of asynchronous graphs, the unfolding is easily defined:

Definition 37 ([WN95])**.** Given a pointed asynchronous graph (G, x_0), its *unfolding* is the asynchronous graph $\tilde{G}$ whose vertices are the dihomotopy classes of dipaths originating at x_0 and transitions and independence are defined as follows. For each transition $a : x \to y$, originating at vertex x which is reachable from x_0 by a dipath $s : x_0 \twoheadrightarrow x$, there is a transition $a : [s] \to [s \,.\, a]$ in $\tilde{G}$. For each pair of paths of the form $a \,.\, b : [s] \to [s \,.\, a \,.\, b]$ and $b' \,.\, a' : [s] \to [s \,.\, b' \,.\, a']$, with $[s \,.\, a \,.\, b] = [s \,.\, b' \,.\, a']$, we have $a \,.\, b \diamond b' \,.\, a'$.

This unfolding can be thought of as a directed variant of the classical notion of universal cover. There is an obvious "projection" morphism $\pi : \tilde{G} \to G$ sending a vertex $[s]$, with $s : x_0 \twoheadrightarrow x$, to x, and sending an edge a to itself. It can be shown that this morphism satisfies the dipath lifting property: given a path $s : x \twoheadrightarrow y$ in G and a vertex x' in $\tilde{G}$ such that $\pi(x') = x$, there is a unique dipath $s' : x' \to y'$ in $\tilde{G}$ such that $\pi(s') = s$, and this path is called the *lifting* of s' at x'. It also satisfies a form of dihomotopy lifting property: the liftings of two dihomotopic dipaths at the same vertex are dihomotopic.

In order to properly define the unfolding in the case of a pointed precubical set (C, x_0), a more general notion of "dipath" should be considered. A *cube path* in $(x_1, x_2, \ldots, x_n)$ is a sequence of cubes in C such that either x_i is an iterated lower face of x_{i+1}, or x_{i+1} is an iterated upper face of x_i for $i \in [n]$. For those, a notion of dihomotopy can be defined, generalizing the one on dipaths [Fah05].

Definition 38 ([vG91, Fah05])**.** Given a pointed precubical set (C, x_0), its unfolding $\tilde{C}$ is the cubical set whose n-cubes are homotopy classes $[x_1, \ldots, x_k]$ of cube paths $(x_1, \ldots, x_k)$ with $x_k \in C(n)$, and faces suitably defined.

A projection morphism $\pi : \tilde{C} \to C$ can be defined by sending an n-cube $[x_1, \ldots, x_k]$ to x_k, and it can be shown to enjoy expected lifting properties. It can also be shown that it corresponds, on the geometrical side, to the notion of universal dicoverings of d-spaces via geometric realization [Fah05], see Section 3.1.

2.3.3 Other constructions

Various other constructions can be defined on precubical sets, some of which will be introduced later on, when needed. For instance, a "compressed" representation of the state space of a precubical set can be obtained by computing the category of components of its fundamental category, this will be given in detail in Section 2.5.2. The geometric realization associates a topological (directed) space with a precubical set thus allowing one to compute classical invariants for those, such as homology groups; some of these are described in Chapter 3.

2.4 Comparing higher-dimensional models for concurrency

This section is mostly based on [GM12a].

As mentioned earlier, the choice of precubical sets comes quite naturally in order to model higher commutations of actions, and has the advantage of being a well-established description of geometric spaces. However, when introducing a new model for concurrency, one is bound to compare it to pre-existing ones, in order to ensure that it constitutes an interesting generalization and not a reinvention of the wheel. This is why we compare precubical sets to other classical models, by constructing adjunctions, which extend some of those constructed by Winskel et al. [WN95, NSW94] for 2-dimensional models, and refine the earlier work on the subject, initiated by Goubault [Gou01]. We begin by recasting some classical adjunctions in the setting of precubical sets. Most of the classical models do not encode higher-dimensional commutations of events, so that we can in fact restrict to 2-dimensional precubical sets (or even asynchronous graphs) as usually done, the only notable exception being the one of Petri nets. For reasons explained in Section 2.2.3, we will focus on the category $\hat{\square}_S/!$ of labeled symmetric precubical sets. The non-labeled case can be recovered using the following lemma, which states that a non-labeled symmetric precubical set can always canonically be seen as a labeled one, by labeling a transition by the corresponding event.

Lemma 39 ([GM12a])**.** *The functor $\hat{\square}_S \to$ **Set** sending a precubical set C to its set $C(1)/\approx$ of events admits $!$ as left adjoint. Moreover, there is an embedding $\hat{\square}_S \to \hat{\square}_S/!$ which labels a precubical set by its set of events.*

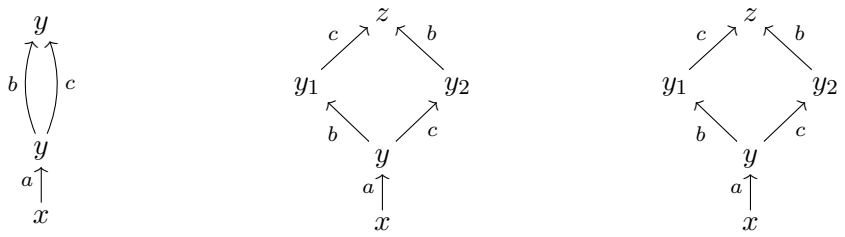
In the following, we write **HDA** for the category of HDA (i.e. pointed labeled symmetric precubical sets) and construct adjunctions between this category and classical models for concurrent systems. We also write $\mathbf{HDA}_n$ for the category of n-truncated HDA. By results of Section 2.2.3, an adjunction between a category $\mathcal{C}$ and $\mathbf{HDA}_n$ extends to an adjunction between $\mathcal{C}$ and **HDA**.

2.4.1 Asynchronous transition systems

Recall that a *transition system* (S, i, E, T), or TS for short, consists of a set S of *states*, an initial state $i \in S$, a set E of *events* and a set $T \subseteq S \times E \times S$ of *transitions*. In other

words, a TS is simply a graph labeled in events with no parallel edges with the same label. A TS is *deterministic* when any two transitions with the same source and label have the same target. We write **TS** for the category of transition systems and the expected notion of morphism. In order to take independence of actions in account, following the same pattern as in Section 1.4, this notion has been refined into the notion of *asynchronous transition system* (TS, I), or ATS, where TS is a deterministic transition system and I is a set of pairs of *independent* coinitial transitions such that whenever (x, a, y) and (x, b, y') are independent there exists a vertex z such that (y, b, z) and (y', a, z) are transitions (such a vertex z is necessarily unique by determinism). We write **ATS** for the corresponding category. Some variants of this notion were mentioned at the end of Section 1.4.

Example 40. The CCS processes $a\,.(b + c)$, $a\,.(b\,.\,c + c\,.\,b)$ and $a\,.(b|c)$ respectively induce the following asynchronous transition systems:



In the second example, no transitions are independent, whereas in the last one (y, b, y_1) and (y, c, y_2) are.

We define a pair of functors $F : \mathbf{HDA}_2 \to \mathbf{ATS}$ and $G : \mathbf{ATS} \to \mathbf{HDA}_2$. With any HDA C, one can associate a TS whose vertices are the elements of $C(0)$, initial vertex is the initial vertex of C, E is the set of labels of C, there is a transition (x, a, y) when there is an edge $x \to y$ in C labeled by a. This TS can be made deterministic by suitably quotienting vertices, and the ATS FC is defined as this TS such that for every square $\alpha \in C(2)$, the transitions corresponding to $\partial_0^-(\alpha)$ and $\partial_1^-(\alpha)$ are independent. Conversely, the functor G associates with an ATS A the HDA whose vertices are those of A, labels are the events of A, edges are the transitions of A labeled by their event, and there is a square for each independence relation. As expected,

Proposition 41 ([WN95, GM12a])**.** *We have an adjunction* $F : \mathbf{HDA}_2 \rightleftarrows \mathbf{ATS} : G$. *The induced comonad is the identity on* **ATS** *and the adjunction induces an equivalence if we restrict* $\mathbf{HDA}_2$ *to the full subcategory of deterministic, strongly labeled* HDA. *This adjunction lifts to the variants of categories with partial morphisms, with similar properties.*

2.4.2 Event structures

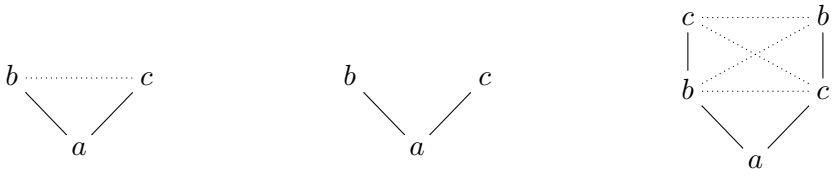
Event structures [NPW79, Win86] were introduced in order to abstract away from the precise places and times at which events occur in distributed systems. The idea is to focus on the notion of event and the causal ordering between them.

Definition 42. A *(prime) event structure* $(E, \leqslant, \#)$ consists of a poset $(E, \leqslant)$ of *events*, the partial order relation expressing *causal dependency*, together with a symmetric irreflexive relation $\#$ called *incompatibility* satisfying

— finite causes: for every event e, the set $\{e' \mid e' \leqslant e\}$ is finite,

— hereditary incompatibility: for events e, e' and e'', $e \# e'$ and $e' \leqslant e''$ implies $e \# e''$.
A *configuration* is a downward-closed set of compatible events. A morphism $f : E \to E'$ of event structures consists of a function between events which preserves configurations and is *locally injective*, i.e. injective when restricted to a configuration. A *labeled* variant is easily defined and we write **LES** for the corresponding category.

Example 43. The CCS processes $a.(b+c)$, $a.(b|c)$ and $a.(b.c+c.b)$ respectively induce the following labeled event structures (to be read from bottom up, the continuous lines representing the partial order and the dotted ones expressing incompatibilities):

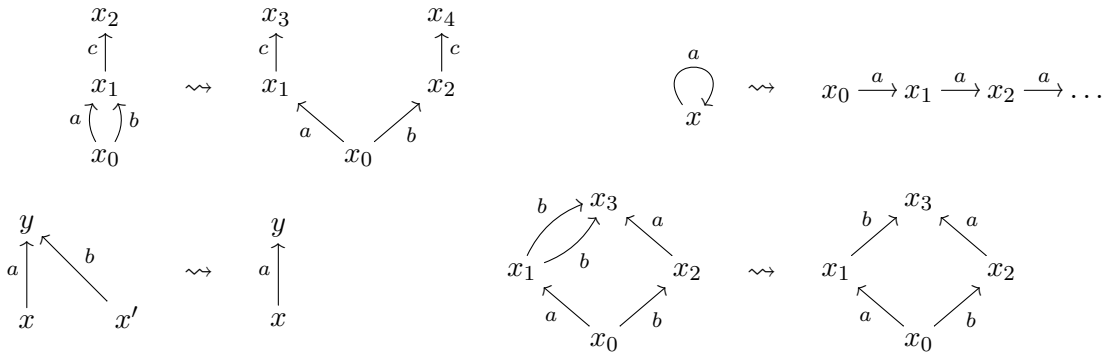


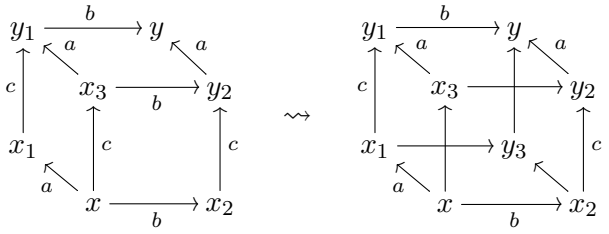
Notice that in the last one, b and c appear twice: this is because we have figured the labels and not the events (and two distinct events can of course have the same label).

We define a functor $F : \mathbf{LES} \to \mathbf{HDA}_2$ which, with an event structure, associates the HDA whose vertices are the configurations of the ES, with $\emptyset$ as distinguished vertex, edges are pairs $(x, e) : x \to x \cup \{e\}$, where x and $x \cup \{e\}$ are configurations and e an event not occurring in x, with x as source and $x \cup \{e\}$ as target and the label of e as label, and every square of edges is the border of a square. Conversely, we define a functor $G : \mathbf{HDA}_2 \to \mathbf{LES}$ as follows. Notice that there is intuitively no way of representing loops in event structures which explains why we have to unfold HDAs first. Given an HDA A, we begin by constructing its unfolding $\tilde{A}$ (see Definition 38) and associate with it the event structure whose events are the events of $\tilde{A}$ (in the sense of Definition 33), and causal dependency is such that $e \leqslant e'$ when for every dipath in $\tilde{A}$ with an instance of e' there is an instance of e occurring before, and $e \# e'$ when no dipath in $\tilde{A}$ contains both an instance of e and an instance of e'.

Proposition 44 ([SNW94, GM12a])**.** *The composite functor $G \circ F$ is isomorphic to the identity functor on* **LES***: the category* **LES** *embeds fully and faithfully into* $\mathbf{HDA}_2$*.*

Notice that we did not claim that F and G are part of an adjunction, because it is not the case. Namely, consider the effect of the endofunctor $T = F \circ G : \mathbf{HDA}_2 \to \mathbf{HDA}_2$: we have pictured some HDAs (on the left) together with their image under T (on the right):





In the middle left example, x is the initial position and in the bottom two examples all the squares for which it makes sense are filled with a 2-square. These examples are representative of various kinds of behaviors that can happen:

— the top two examples show that "shared transitions" are "unshared", and in particular loops are unrolled,
— the example on the middle left shows that the unreachable vertices of the HDA are removed,
— the middle right example shows that the HDA is made strongly labeled,
— the bottom example shows that if the HDA contains "half of a cube" then the other "half" of the cube is created.

If we write C for the HDA in the left of examples, in the first three examples there is an arrow $TC \to C$ (but not in the other direction), whereas in the last two examples there is a natural arrow $C \to TC$ (but not in the other direction). So, there is no hope that T would be either a monad or a comonad, and thus that F and G form an adjunction in either direction. It can however be shown [SNW94, GM12a] that G is right adjoint to F if we restrict $\mathbf{HDA}_2$ to the full subcategory whose objects are strongly labeled and satisfy the *cube property*, which states that if an asynchronous transition system contains half of a cube as in fourth example then it also contains the other half of the cube, as well as two other variants of this property, which are detailed in Chapter 4, see Definition 101. This adjunction can also be extended to an adjunction between general event structures, in which conflict is not necessarily a binary relation, and HDA.

2.4.3 Petri nets

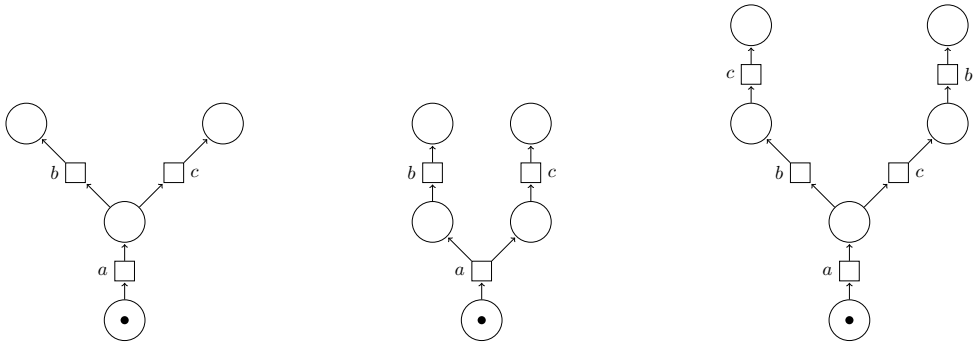
Petri nets are a well-known model for parallel computation, generalizing transition systems by using a built-in notion of resource. This allows us to derive a notion of independence of events, which is much more general than the independence relation of asynchronous transition systems. Numerous variants of Petri nets have been considered since they were introduced [Pet66], and we work with the definition used by Winskel and Nielsen in [WN95], since it is equipped with a notion of morphism which is well-suited for formal comparisons with other models for concurrency:

Definition 45. A *Petri net* N, or PN, is a tuple $(P, M_0, E, {}^\bullet{-}, {-}^\bullet)$ where

— P is a set of *places*,
— $M_0 \in \mathbb{N}^P$ is the *initial marking*,
— E is a set of *events* (or *transitions*),
— ${}^\bullet{-} : E \to \mathbb{N}^P$ and ${-}^\bullet : E \to \mathbb{N}^P$ are the *precondition* and *postcondition* functions.

A *morphism of Petri nets* $(\varphi, \psi) : N \to N'$ consists of a function $\varphi : P' \to P$ and a function $\psi : E \to E'$ such that for every event $e \in E$, $M'_0 = M_0 \circ \varphi$, ${}^\bullet\psi(e) = {}^\bullet e \circ \varphi$ and $\psi(e)^\bullet = e^\bullet \circ \varphi$. We write **PNet** for the category of Petri nets. A variant where events are labeled is defined easily and we write **LPNet** for the corresponding category.

Example 46. The CCS processes $a.(b+c)$, $a.(b|c)$ and $a.(b.c+c.b)$ respectively induce the following labeled Petri nets:



In the diagrams above, we have used the usual notation for Petri nets: square nodes represent transitions, circled ones represent places (with dots indicating tokens) and arrows represent pre- and postconditions.

Remark 47. Following [WN95], we have chosen to define morphisms in the opposite direction on places. With the adjunction with HDA defined below in mind, this can be explained as follows. Just as morphisms of HDAs, morphisms of PNs should preserve independence of events: if two events e and e' of a net N are independent and $(\varphi, \psi) : N \to N'$ is a morphism of nets, then their images $\psi(e)$ and $\psi(e')$ should also be independent. By contraposition, this means that if both events $\psi(e)$ and $\psi(e')$ depend on a common place p, then the events e and e' should depend on a corresponding common place $\varphi(p)$.

The rest of this section constitutes its most novel part. We extend the previously constructed adjunctions between 1-bounded Petri Nets and asynchronous transition systems [WN95, DS93, Muk92, vG99] to an adjunction between general Petri Nets and HDA. For similar reasons to previously, one needs to restrict to strongly labeled HDA in order to obtain a well-defined adjunction. We thus implicitly only consider strongly labeled HDAs in the following.

A *marking* M of a PN P is a function in $\mathbb{N}^P$, which associates, with every place, the number of resources (or tokens) that it contains. The sum $M_1 + M_2$ of two markings M_1 and M_2 is their pointwise sum. An event e induces a *transition* between two markings M_1 and M_2, that we write $M_1 \xrightarrow{e} M_2$, whenever there exists a marking M such that $M_1 = M + {}^\bullet e$ and $M_2 = M + e^\bullet$.

Cubical transition systems. We first introduce a general methodology for associating a symmetric precubical set to a model for concurrent processes, which we will use in order to associate an HDA with a PN. Since monoidal functors preserve the unit of monoidal categories, all precubical sets generated by co-precubical objects in **Set**, by the functor ! of Definition 31, contain only one vertex. Cubical sets with multiple vertices can be generated by *actions* of the labeling cubical set on the vertices, formalized as follows, in the same way that a transition system can be seen as an action of the free monoid on labels over the states. The resulting notion of *cubical transition system* (or CTS) generalizes to the setting of cubical sets the notion of *step transition system* [Muk92], which is a variant of transition systems in which multiple events can occur simultaneously.

Definition 48 ([GM12a])**.** A *cubical transition system* (S, i, E, t, ℓ, L) consists of
— a set S of *states*,

— a state $i \in S$ called the *initial state*,
— a set E of *events*,
— a *transition function* which is a partial function $t : S \times \coprod_{n\in\mathbb{N}} !\,E(n) \to S$,
— a set L of *labels*,
— a *labeling function* $\ell : E \to L$,
such that for every state x and every n-cell l of $!\,E$ for which $t(x, l)$ is defined,

1. if $l = l_1 \,.\, l_2$ for some cells l_1 and l_2 then $t(x, l_1)$ and $t(t(x, l_1), l_2)$ are both defined and we have $t(x, l) = t(t(x, l_1), l_2)$,
2. $t(x, ())$ is defined and equal to x (where $()$ denotes the 0-cell of $!\,E$),
3. for every symmetry $\sigma : n \to n$, $t(x, !\,E(\sigma)(l))$ is defined and equal to $t(x, l)$.

Cubical transition systems are thus generalized transition systems, which modify state upon incoming events. These differ from traditional transition systems in that they may accept a transition under n events $e_1, \dots, e_n$, specified by a transition under the word $e_1 \dots e_n$ of $!\,E$. With this understanding in mind, the axioms have simple interpretations: for example the first one states that the state reached under two simultaneous events e_1 and e_2 is the same as the state reached under e_1 followed by e_2.

An n-cell l of $!\,E$ is *enabled* at a position x if $t(x, l)$ is defined. Every such CTS defines an HDA C labeled by L whose n-cells are pairs (x, l) where x is a state and l is an n-cell of $!\,E$ which is enabled at x. Source and target functions are defined by $\partial_i^-(x, l) = (x, \partial_i^-(l))$ and $\partial_i^+(x, l) = t(t(x, e_i), \partial_i^+(l))$ where e_i is the i-th element of l and symmetries are defined by $\sigma(x, l) = (x, !\,E(\sigma)(l))$. The labeling function is given by $!\,\ell$ applied on the second component and the initial state is (i, ε). The notion of morphism of CTS can be defined as expected, thus inducing a category **CTS**, and the operation defined above extends to a functor **CTS** $\to$ **HDA** [GM12a].

From Petri nets to HDA. Suppose given a labeled Petri net $N = (P, M_0, E, \mathrm{pre}, \mathrm{post}, \ell, L)$. The pre and post operations can be extended to the cells of $!\,E$ by ${}^\bullet() = {}^\bullet(*) = 0$, ${}^\bullet(l_1 \,.\, l_2) = {}^\bullet l_1 + {}^\bullet l_2$, $()^\bullet = (*)^\bullet = 0$ and $(l_1 \,.\, l_2)^\bullet = l_1^\bullet + l_2^\bullet$, thus enabling us to consider the elements of $!\,E$ as generalized events. We also generalize the notion of transition and given two markings M_1 and M_2 and an event $l \in !\,E$, we say that there is a transition $M_1 \xrightarrow{l} M_2$ whenever there exists a marking M such that $M_1 = M + {}^\bullet l$ and $M_2 = M + l^\bullet$. In this case, the event l is said to be *enabled* at the marking M_1. The marking M_2 is sometimes denoted M_1/l. A marking M is *reachable* if there exists a transition l such that $M = M_0/l$ where M_0 is the initial marking of N.

Every labeled Petri net N induces a CTS (S, i, E, t, ℓ, L) whose states S are the reachable markings of the net, with the initial marking M_0 as initial state, events E are the events of the net, transition function $t(M, l)$ is defined if and only if l is enabled at M and in this case $t(M, l) = M/l$, with the set L as set of labels and $\ell : E \to L$ as labeling function. It is routine to verify that this actually defines a CTS and thus an HDA, denoted HDA(N). Its n-cubes consist of a marking M of the net and a list l of events which is enabled at M. Moreover, any morphism between labeled Petri nets induces a morphism between the corresponding CTS. We denote by HDA : **LPNet** $\to$ **HDA** the functor thus defined.

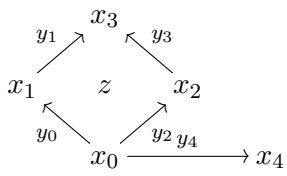
From HDA to Petri nets. We first introduce the notion of region of an HDA, which should be thought as a way of associating a number of tokens to each vertex of the HDA and a pre-

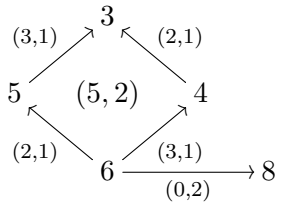
and postcondition to every transition of the HDA, in a coherent way. A *pre-region* R of a precubical set C is a sequence $(R_i)_{i\in\mathbb{N}}$ of functions $R_i : C(i) \to \mathbb{N}\times\mathbb{N}$ such that

1. for every $x \in C(0)$, $R_0(x) = (0,0)$
2. for every $x \in C(i+1)$ and $(\epsilon_k) \in \{-,+\}^k$, $R_{i+1}(x) = \sum_{k=0}^{i} R_1(\partial^{\epsilon_k}_{\neg k}(x))$ where the sum is computed coordinatewise on pairs of natural numbers and $\partial^{\epsilon}_{\neg k}$ is a notation for $\partial^{\epsilon}_0\partial^{\epsilon}_1\dots\partial^{\epsilon}_{k-1}\partial^{\epsilon}_{k+1}\dots\partial^{\epsilon}_i$, taking faces in every dimension except k.

Notice that, by the second property, a region is uniquely determined by the image of edges in $x \in C(1)$. We sometimes omit the index i since it is determined by the dimension of the cube in argument and respectively write $R'(x)$ and $R''(x)$ for the first and second components of $R(x)$, where x is a cube of C: these numbers intuitively specify a number of tokens that the cube x, considered as a generalized transition, consumes and produces respectively. It can be remarked that two edges which are part of the same event necessarily have the same image under a pre-region; a pre-region R thus induces a function from the events of C to $\mathbb{N}\times\mathbb{N}$, that we still write R. A *region* of a precubical set consists of a pre-region R together with a function $S : C(0) \to \mathbb{N}$ such that for every i-cube $y \in C(i)$ whose 0-source is x and 0-target is x', there exists an natural number n such that $(S(x), S(x')) = (n + R'(y), n + R''(y))$.

Example 49. Consider the precubical set pictured on the left:





A region (R, S) for this precubical set is for instance

$$R(y_0) = (2,1) \quad R(y_1) = (3,1) \quad R(y_2) = (3,1) \quad R(y_3) = (2,1) \quad R(y_4) = (0,2)$$

and

$$R(z) = (5,2) \quad S(x_0) = 6 \quad S(x_1) = 5 \quad S(x_2) = 4 \quad S(x_3) = 3 \quad S(x_4) = 8$$

as depicted on the right.

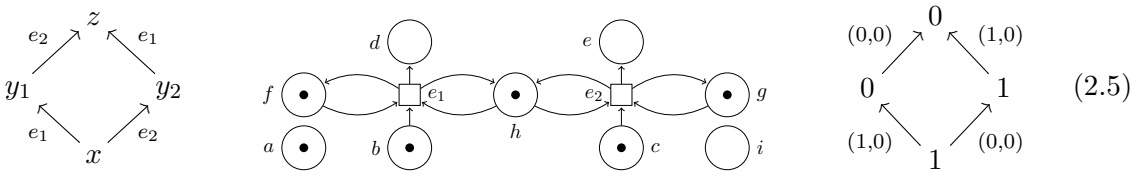
With every strict HDA C, we associate a labeled Petri net PN(C) whose
— places are the regions of C,
— events are the events of C, labeled as in C,
— pre and post functions are given on any event e and any place (R, S) by ${}^\bullet e(R, S) = R'(e)$ and $e^\bullet(R, S) = R''(e)$,
— initial marking M_0 is $M_0(R, S) = S(x_0)$, where x_0 is the initial state of C.

This operation extends to a functor PN : **HDA** $\to$ **LPNet**.

Theorem 50 ([GM12a])**.** *We have an adjunction* PN : **HDA** $\rightleftarrows$ **LPNet** : HDA.

Example 51. If we restrict to 1-bounded nets, which are nets in which a place can contain either 0 or 1 token, we recover the constructions of [WN95] for constructing an adjunction between asynchronous transition systems and nets. For instance, consider the asynchronous

transition system, depicted on the left of (2.5), with an empty independence relation.



(2.5)

The associated 1-bounded Petri net is shown in the middle. In this automaton the place d corresponds to the region (R, S) such that $R(e_1) = (1, 0)$, $R(e_2) = (0, 0)$, $S(x) = S(y_2) = 1$ and $S(y_1) = S(z) = 0$, pictured on the right. Now, if we consider the same automaton with e_1 and e_2 independent, we obtain the same Petri net with the place h removed. In general, the Petri net associated to an HDA is infinite (even for very simple examples) and thus difficult to describe, which is why we only have provided examples in the bounded case.

2.5 Partial order reduction and the category of components

This section is mostly based on [GHM13].

The work described in this section is motivated by the following, often asked, question: *how do geometric techniques compare to traditional verification techniques for concurrent programs?* In order to provide a first answer to this question, we compare two of the most important techniques in each field. On the verification side, *partial order reduction* aims at addressing the state space explosion problem by identifying transitions which can be performed as early as possible without restricting the possible behaviors of the program. On the other hand, the notion of *category of components* was introduced in order to obtain a small representation of the state space (i.e. the fundamental category of a model of the program) by forgetting about morphisms which are inessential, in the sense that they have no influence on the (future) behavior of the program. Although they originate from very distinct considerations, we show that, perhaps surprisingly, these two techniques essentially amount to the same thing for a large class of programs. This was first conjectured by Goubault and Raussen [GR02], although it was limited to observations in particular cases, and we handle the general case here.

We suppose fixed a *concurrent alphabet* (L, I), consisting of a set L of actions, and a symmetric irreflexive relation $I \subseteq L \times L$ indicating when two actions are *independent*. For simplicity, we suppose that our language does not contain resources, and two actions commute as specified by I in the asynchronous semantics, as in the variant explained in Remark 12. In practice, this means that we will consider here asynchronous graphs such that we have an independence tile $A \,.\, B \diamond B \,.\, A$ if and only if $A \; I \; B$, e.g. we have the following models:

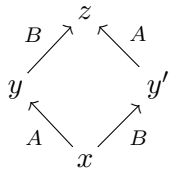
`x:=1 ‖ x:=2` `x:=1 ‖ y:=2`

where, obviously, `x:=1` is independent from `y:=2` but not from `x:=2`. It is enough to consider labeled asynchronous graphs and not the full precubical model since partial order reduction only depends on the 2-dimensional structure on the model.

2.5.1 Partial order reduction

On labeled transition systems. The partial order reduction techniques are generally formalized in the setting of *labeled transition systems*, or LTS, consisting of a graph labeled in L, whose vertices are often called *states* in this context. We begin by recalling the traditional definitions on those. We say that an action $A \in L$ is *enabled* in a vertex x when there exists a transition labeled by A originating in x. We shall only consider LTS, called LTS *with independence*, satisfying the two following conditions:

1. *determinism*: there are no distinct coinitial transitions with the same label,
2. *independence*: given actions $A, B \in L$ which are independent and a transition $x \xrightarrow{A} y$, B is enabled at x if and only if it is enabled at y. When this is the case, we moreover assume that we have a square



   i.e. the vertices reached from x by the paths $A \,.\, B$ and $B \,.\, A$ are the same (note that A is necessarily enabled at y' by symmetry of I).

Persistent sets are one of the most well-known techniques to reduce the number of executions to be explored in order to check that a concurrent program cannot deadlock. They are notably used in the SPIN model checker [Hol04]. For simplicity, we will only consider this technique here, but we believe that the methodology is general enough to adapt to other variants: stubborn sets [Val89], ample sets [Pel93], sleep sets [God96], etc.

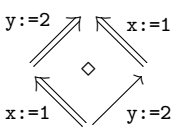
Definition 52 ([God96])**.** Given a vertex x of a fixed LTS, a set P of edges with x as source is *persistent* in x, if for every nonempty sequence of transitions

$$x = x_0 \xrightarrow{A_0} x_1 \xrightarrow{A_1} x_2 \ldots \xrightarrow{A_{n-1}} x_n$$

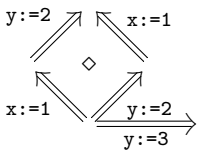
from x that satisfy $A_i \notin P$ for $i \in [n]$, we have $A_i \; I \; A$ holds for every transition $A \in P$ and $i \in [n]$. Since LTS are supposed to be deterministic, the set P can also be specified as a set of actions enabled at x.

A single transition can be considered persistent whenever it can always be "pulled back" to the state at which it was first enabled by permuting it with independent (not necessarily persistent) transitions. More generally, persistent sets typically comprise the actions of a conditional choice of some concurrent component, as illustrated in the following example.

Example 53. In the two LTS below, the set of transitions from any state that are drawn in double lines form a persistent set; we also have given programs whose semantics are the LTS.



`x:=1 || y:=2`



`if z<0 then (x:=1||y:=2) else y:=3`

In the LTS on the left, we have `x:=1` and `y:=2` running completely in parallel; thus, both {`x:=1`} and {`y:=2`} are persistent sets at the initial state. In contrast, in the system on the right, the transitions `x:=1` and `y:=2` are in conflict with `y:=3`; the only persistent set consists of a "monolithic" component with no threads running concurrently, i.e. {`x:=1`, `y:=2`, `y:=3`} is the only persistent set at the initial state.

Recall that a *deadlock* is a state in which no transition is enabled. The main interest of persistent sets is to narrow the search for deadlocks in a concurrent program: given a choice of persistent set for each state, a reachable deadlock is always reachable by a dipath containing only transitions in the chosen persistent sets; it is therefore sufficient to explore a system "along" persistent sets:

Proposition 54 ([God96])**.** *Given a choice of a non-empty persistent set P_x at each state x that is not a deadlock, for each dipath $x = x_0 \xrightarrow{A_0} x_1 \dots x_{n-1} \xrightarrow{A_{n-1}} y$ to a deadlock y, there exists a dipath $x = x'_0 \xrightarrow{A'_0} x'_1 \dots x'_{n-1} \xrightarrow{A'_{n-1}} y$ such that $A'_i \in P_{x'_i}$ for all $i \in [n]$.*

One can always trivially take, for every state x, P_x to be the set of all transitions enabled at x, but previous proposition is interesting when the chosen persistent sets are as small as possible.

On labeled asynchronous graphs. Any labeled transition system induces a labeled asynchronous graph, with the same underlying labeled graph, and we have $A \,.\, B \diamond B \,.\, A$ if and only if $A \; I \; B$. Reformulated in the setting of asynchronous graphs, the proof of the preceding Proposition 54 is based on the following observation: for each dipath $s : x \twoheadrightarrow y$ leading to a deadlock y and persistent set R at x, there exists a dipath $t \in [s]$, i.e. $s \sim t$, whose initial transition is in R. This motivates our generalization of the definition of persistent sets to asynchronous graphs.

We first recall some classical notions on asynchronous graphs [Mim08]. Given a dipath $s : x \twoheadrightarrow z$, a transition $a : x \to y$ is *initial modulo dihomotopy* in s if there exists a dipath $t : y \twoheadrightarrow z$ such that $s \sim a \,.\, t$. We write $\tilde{\imath}(s)$ for the set of such transitions. Two coinitial dipaths $s : x \twoheadrightarrow y$ and $t : x \twoheadrightarrow z$ are *compatible*, written $s \uparrow t$, if there exist cofinal dipaths $t' : y \twoheadrightarrow x'$ and $s' : z \twoheadrightarrow x'$ such that $s \,.\, t' \sim t \,.\, s'$. In particular, all transitions that are initial modulo dihomotopy in some dipath are pairwise compatible. We can now generalize persistence to asynchronous graphs as follows:

Definition 55 ([GHM13])**.** Let P be a set of transitions in an asynchronous graph sharing a state x as common source. The set P is *dihomotopy persistent*, if each dipath $s : x \twoheadrightarrow z$ is compatible with all transitions in P provided that no transition in P is among its dihomotopy initial ones, i.e. $P \cap \tilde{\imath}(s) = \emptyset$ implies $s \uparrow a$ for every $a \in P$.

Remark 56. A *persistent singleton* (a persistent set with a single element) corresponds to a transition that is compatible with all dipaths with the same source.

It remains to show that the notion of dihomotopy persistent set is in fact a conservative extension of the original one (Definition 52). The proof is based on the observation that in each AG induced by an LTS, a transition $a : x \to y$ has a unique residual dipath $s/a : y \twoheadrightarrow z'$ after a compatible dipath $s : x \twoheadrightarrow z$; we say this kind of AG *has compatible residuals*. The residual s/a of s after a has intuitively the "same effect" as s, once a has been performed. This notion is also presented and studied in more details, in a slightly different setting, in Chapter 9 (Section 9.2.1). We write ε_x for the empty path at x.

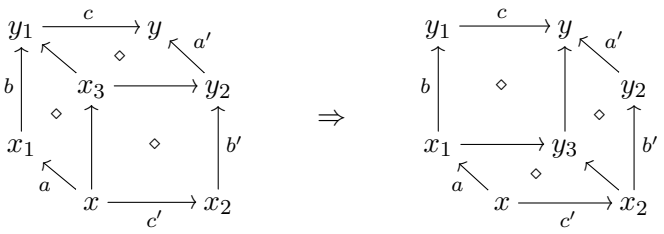
Definition 57. Let $a\colon x \to y$ be a transition, let $s : x \twoheadrightarrow z$ be a dipath. The *residual* of s after a, denoted by s/a, and the *residual* of a after s, denoted by a/s, are defined by induction on the length of s by

$$\varepsilon_x/a = \varepsilon_y \qquad a/\varepsilon_x = a \qquad (a \cdot s')/a = s' \qquad a/(a \cdot s') = \varepsilon \qquad (b \cdot s)/a = b' \cdot (s/a')$$

where, in the last case, we suppose that $a \neq b$ and a', b' are transitions such that $a \cdot b' \diamond b \cdot a'$. This definition can be extended in the expected way in order to define the residual t/s of a dipath t after a coinitial dipath s.

It can be shown that every AG which is induced by an LTS has compatible residuals: this can be deduced from the fact that they satisfy the forward cube property [Mor05], already encountered in Section 2.4.2, and detailed in Chapter 4, see Definition 101. In particular, we see there that it corresponds to the fact that independence I is a binary relation:

Definition 58. An AG satisfies the *forward cube property* (or FCP) if for every three tiles as shown on the left there exist three matching tiles as shown on the right:



Lemma 59. *The induced* AG *of an* LTS *satisfies the* FCP.

Our main interest in this property is the following consequence:

Proposition 60 ([Mim08])**.** *An* AG *with the* FCP *has compatible residuals.*

This proposition is the main tool to show that in fact our definition of persistent set is a conservative extension of the original one:

Proposition 61 ([GHM13])**.** *Given an* LTS *and a vertex x a set P of transitions is persistent at x if and only if P is dihomotopy persistent at x in the induced* AG.

With this result at hand, we refer to dihomotopy persistent sets in AG as persistent ones from now on. Moreover, we have a further successful "soundness check" for our generalization of persistent sets, namely, the fundamental fact about reachability of deadlocks "along" persistent sets, namely Proposition 54, lifts to any AG.

Proposition 62 ([GHM13])**.** *Given an* AG *and a choice of non-empty persistent set P_x for each non-deadlocking vertex x, a deadlock is reachable by a dipath if and only if it is reachable by a dihomotopic dipath containing only persistent transitions.*

2.5.2 The category of future components

The notion of category of components was introduced by Haucourt and others [Hau05b, FGHR04, Hau05a, Hau06, FGH$^+$16] as a way to provide a small representation of the state space. The idea is the following: given a category $\mathcal{C}$, which is typically the fundamental

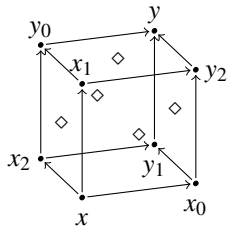
category of a geometric model of the program, see Definition 36, a smaller category can be obtained by quotienting the category under morphisms which are "inessential", in the sense that they correspond to execution traces that do not fundamentally change the states which can be attained in the past or the future. We reformulate here the construction of the category of future components in the setting of (fundamental categories of) asynchronous graphs, as well as related properties: we introduce the notion of future-reflecting trace, and the category of future components is then defined as the quotient of the category of traces by a consistent set of future-reflecting traces. After that, the crucial point is to make precise which transitions are considered as *inessential*. Intuitively, one might expect that all persistent transitions are inessential. However, the general definition of asynchronous graphs allows for peculiar situations which do not occur in those which are generated by usual programs (for instance persistent transitions might not be stable under residuation). Nevertheless, we will show, for suitably "well-behaved" AG, that inessential transitions are the same as persistent ones and that the category of future components does only contain states that do not enable persistent transitions.

The first requirement for *inessential* transitions is that they do not influence any choices that might lead to deadlocks. Intuitively, choices available in the future before and after performing an inessential transition should be the same: they should be future reflecting in the following sense.

Definition 63. A trace $[s] : x \twoheadrightarrow y$ is *future-reflecting* if for each position z reachable from y, precomposition with $[s]$ induces a bijection between traces from y to z and traces from x to z.

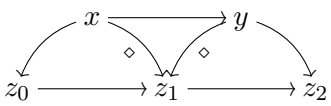
In other words, if a trace is future-reflecting, all choices in the future are already present at its source. The notion of future-reflecting *transition* is close to the notion of persistent transition; however, the two do not generally coincide as seen in the following examples.

Example 64. Consider the "cube without bottom" from Example 35:



where all the squares are commuting, except the one formed by $x \to x_0 \to y_1$ and $x \to x_2 \to y_1$. Now, both transitions $x \to x_0$ and $x \to x_2$ are persistent since they are compatible with all other transitions from x. However neither of them is a future-reflecting trace. To see that $x \to x_0$ is not a future-reflecting trace, consider the trace $[x_0 \to y_1]$. There is only one trace from x_0 to y_1 but two from x to y_1. The argument for $x \to x_2$ is symmetric. The important point to notice in this example is that the associated fundamental category (see Definition 36) is not a poset. In fact, it can be shown that when the trace category is a poset (and this is for instance the case for event structures), all persistent transitions are future-reflecting.

Example 65. Consider the asynchronous graph



with two independence tiles $x \to y \to z_{i+1} \diamond x \to z_i \to z_{i+1}$ for $i = 0$ and $i = 1$. It can be obtained from two squares

$$\begin{array}{ccc} x & \longrightarrow & y \\ \downarrow & \diamond & \downarrow \\ z_0 & \longrightarrow & z_1 \end{array} \qquad\qquad \begin{array}{ccc} x' & \longrightarrow & y' \\ \downarrow & \diamond & \downarrow \\ z'_1 & \longrightarrow & z_2 \end{array}$$

by identifying the edge $x \to y$ with $x' \to y'$ and the vertex z_1 with z'_1. The transition $x \to y$ is persistent. However, it is not a future-reflecting trace since the trace $[x \to z_1]$ does not factor through $[x \to y]$. Note that this AG is induced by an LTS, but the situation is still pathological in some sense because the path $z_0 \to z_1 \to z_2$ contains two transitions which are part of the same event. Such a situation will be called a *déjà vu* in the following, see Definition 71.

The basic idea of state space reduction used in the category of future components is that future-reflecting transitions are not informative from a concurrency point of view and thus need not be represented explicitly. So, starting from an asynchronous graph, one might be tempted to consider the associated fundamental category (Definition 36) and quotient it by all the future-reflecting traces, which amounts to formally turn them into identities. It turns out that this crushes too much information about traces, see [FGHR04], in particular there is no equivalence between the quotient and the category of fractions and no compositionality result via van Kampen theorems. A suitable solution is to quotient wrt a subset of all future-reflecting traces, namely those that are closed under composition and "residuals"; the formal details are as follows.

Definition 66 ([FGHR04])**.** Given an asynchronous graph, a set Σ of traces is a *system of inessentials*, or SOI, if

1. each element of Σ is a future-reflecting trace,
2. Σ contains all empty traces and is closed under composition,
3. Σ is stable under pushout, i.e. for every trace $[s] : x \twoheadrightarrow z \in \Sigma$ and for any trace $[t] : x \twoheadrightarrow y$, there exists a pushout $z \xrightarrow{[t']} x' \xleftarrow{[s']} y$ of $z \xleftarrow{[s]} x \xrightarrow{[t]} y$ and $[s'] \in \Sigma$.

In an AG satisfying the forward cube property, the pushout of an inessential trace along another one can be computed as its residual, see Chapter 9 (Proposition 241): in this case, condition 3. above amounts to suppose that Σ is closed under residuation.

Definition 67. A trace of an asynchronous graph is *inessential* if it belongs to the union of all its systems of inessentials (which is a non-empty SOI, maximal wrt inclusion).

Remark 68. The very similar notion of *equational morphism* will be introduced in Chapter 9, based on similar considerations transposed in the framework of presentations of categories.

Definition 69. The *category of future components* of an asynchronous graph is its trace category quotiented by inessential traces.

This construction amounts to forgetting inessential transitions by considering them as identities. Another point of view, formalized by the following proposition, is that this construction removes states which enable inessential transitions: informally, passing through them does not bring any new information about possible future traces (as no important choice can be made by the program or the scheduler).

Proposition 70 ([FGHR04])**.** *Suppose given an asynchronous graph G, whose maximal* SOI *Σ is finite. Then the category of future components is the full subcategory of the fundamental category $\vec{\Pi}_1(G)$ whose objects are all states that do not enable any transition in Σ.*

Finally, we introduce a last condition which will imply that the category of future components yields the expected state space. Its role is to prevent situations such as the one described in Example 65 from occurring:

Definition 71. A *déjà vu* is a transition $a : x \to y$ such that there exists a dipath $s : y' \twoheadrightarrow x$ and a transition $a' : x' \to y'$ which is in same event as a. An AG is *déjà vu free* if none of its transitions are déjà vus.

In other words, an AG is déjà vu free if none of its dipaths contains two transitions that are instances of the same event. Déjà vus are illustrated in Example 65; also, any AG with a non-trivial cycle has déjà vus. With this final proviso, we obtain a formal correspondence between inessential and persistent transitions, i.e. transitions $a : x \to y$ such that $\{a\}$ is a persistent set at x.

Theorem 72 ([GHM13])**.** *In an asynchronous graph which is déjà vu free and has the forward cube property, a transition is persistent iff it belongs to some* SOI *(and in particular to the maximal one, i.e. is inessential). If the fundamental category of the* AG *is finite, the category of future components is the full subcategory containing all objects that do not enable persistent transitions.*

Thus, in a large class of common systems, including those generated by event structures or Petri nets with acyclic causality, persistent singletons are in fact the same as inessential morphisms (the asynchronous graphs satisfying the cube property are further studied and detailed in Chapter 4). Despite the huge gap between the origins of the geometric approach and the POR technique, for those systems, we do not have only "obvious" similarities, but in fact, a formal argument that inessential transitions are exploited in the same way.

This illustrates the practical relevance of the theoretical construction of the category of future components and further results in [GH07], where state space reduction is performed for general directed topological spaces. Glossing over details, Theorem 72 says that inessential transitions are the same as persistent singletons. As a direct consequence, the construction of the category of future components roughly amounts to the application of the POR technique when we use only persistent *singletons*. Thus, we have found a common core of the geometric approach and the POR technique, while both approaches have additional heuristics and methods to improve performance. There are favorable examples for the geometric approach [FGH$^+$12], which motivates future research, and some practical experiments have already been performed inside the tool ALCOOL [GH05].

Chapter 3

Geometric models for concurrency

In this chapter, we relate precubical models to more traditional models in geometry. We first recall the topological models, as well as how they can be adapted in order to encompass a notion of time direction (Section 3.1). Then, we introduce metric models, which are more quantitative: in those, we can intuitively measure the time elapsed. For various reasons, the category of metric spaces does not have good properties and we have to consider the generalized notion of metric space introduced by Lawvere, and study some of its properties in detail (Section 3.2). Finally, we briefly report on an implementation of an algorithm proposed by Raussen in order to compute a small model of the trace space, i.e. the space of paths up to reparametrization, which corresponds, from a computer scientific point of view, to executions up to equivalence (Section 3.3). Intuitively, these techniques only apply to the control flow graphs ignoring the labels, i.e. the manipulations of values, they should thus be complemented by the previously described methods such as partial order reduction.

3.1 Directed topological spaces

Geometric realization. The precubical model presented in previous chapter can be seen as a geometric object through the process of geometric realization. Namely, one can define a functor $F : \square \to \mathbf{Top}$ which, with every object n, associates the standard topological n-cube, i.e. the set of points of $\mathbb{R}^n$ whose coordinates x_i satisfy $0 \leqslant x_i \leqslant 1$. Since the category **Top** is cocomplete, it extends as a cocontinuous functor $R_F : \hat{\square} \to \mathbf{Top}$, often written $|-|$, associating to each precubical set C a topological space obtained by gluing topological cubes (one for each n-cube of C) according to faces of C. More explicitly, we have

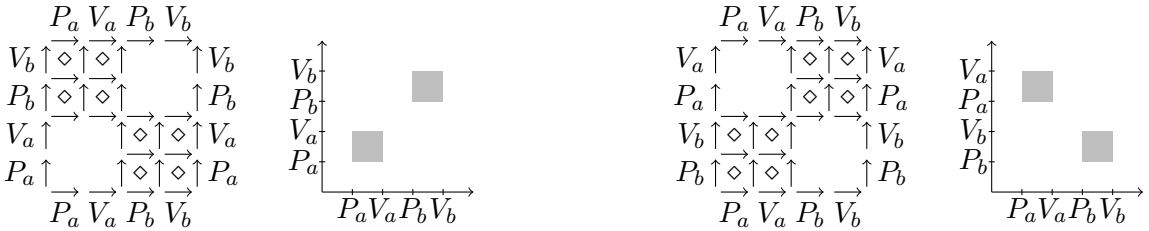
$$|C| \quad = \quad \coprod_{n \in \mathbb{N}} (C_n \times I^n) / \approx$$

where C_n is equipped with the discrete topology, and $\approx$ is the equivalence relation generated by relations $(\partial_i^\alpha(x), p) \approx (x, i_i^\alpha(p))$ for $x \in C_n$ and $p \in I^{n-1}$.

Example 73. Consider the programs

$$P_a ; V_a ; P_b ; V_b \parallel P_a ; V_a ; P_b ; V_b \qquad \text{and} \qquad P_a ; V_a ; P_b ; V_b \parallel P_b ; V_b ; P_a ; V_a$$

their precubical semantics and corresponding geometric realizations are respectively



where the grayed squares denote holes.

Towards directed topological models. In the example above, the two topological spaces are homeomorphic, whereas the two programs are quite different. For instance, there are four execution traces up to equivalence in the first program, whereas there are only three in the second one. This means that the topological realization forgets some important information: in this case, this is of course the notion of time direction. This can also be illustrated in the following way. We have seen in Section 1.4 that, from a semantic point of view, dihomotopy is the right notion of equivalence between dipaths in a precubical set C, corresponding to executions of programs. Now, a path in C induces a path in its geometric realization $|C|$, and it can be shown that two paths in C are homotopic if and only if the corresponding paths in $|C|$ are homotopic, see below. However there is no way of recovering the dihomotopy relation in the topological setting, which is the meaningful one.

Directed topological spaces. This suggests that the notion of topological space should be adapted in order to incorporate a notion of time direction. The most widely accepted notion up to now is Grandis' definition of d-spaces:

Definition 74 ([Gra03])**.** A *d-space* (X, dX) consists of a topological space X equipped with a set dX of paths in X, called *dipaths*, which contains constants paths and is closed under concatenation and partial reparametrizations. A morphism $f : (X, dX) \to (Y, dY)$ is a continuous map $f : X \to Y$ which preserves dipaths. We write **dTop** for the resulting category.

For instance, the d-space $\vec{I}$ consists of the unit interval equipped with weakly increasing paths as dipaths. This d-space represents dipaths, in the sense that for a d-space X we have $dX \cong \mathbf{dTop}(\vec{I}, X)$, and this is actually an isomorphism of topological spaces if we equip dX with the compact-open topology. More generally, the object $\vec{I}$ is exponentiable, meaning that we have an isomorphism $\mathbf{dTop}(X \times \vec{I}, Y) \cong \mathbf{dTop}(dX, Y^{\vec{I}})$ for arbitrary d-spaces X and Y. We write I for the unit interval with only constant paths as dipaths. Two paths $f, g : \vec{I} \to X$ are *dihomotopic* when there is a homotopy $H : I \to X^{\vec{I}}$ from f to g, i.e. $H(0) = f$ and $H(1) = g$, such that the intermediate paths $H(t)$ are dipaths for every $t \in I$. As in the case of precubical sets (see Definition 36), one can define the *fundamental category* $\vec{\Pi}_1(X)$ of a d-space X as the category of points and dipaths up to dihomotopy [Gra03], and the fundamental groupoid can also be defined similarly.

Directed geometric realization. The category of d-spaces is complete and cocomplete, and we write $\uparrow C \uparrow$ for the *directed geometric realization* of the precubical set C, sending the

standard n-cube (as a precubical set) to $\vec{I}^n$. In particular, given a program p, the space $\uparrow\check{C}_p\uparrow$ is called its *geometric semantics* and can be thought of as a continuous analogues of the precubical semantics. Notice that in most papers, this geometric semantics is defined directly as a colimit of d-spaces, e.g. [FGR98, FGH$^+$12], but its definition essentially coincides with the one given here, up to minor details [FGH$^+$16]. Given a dipath f in C, we write $\uparrow f\uparrow$ for the induced dipath in $\uparrow C\uparrow$. Given two dihomotopic dipaths f, g in C, their realizations $\uparrow f\uparrow$ and $\uparrow g\uparrow$ are dihomotopic. Fajstrup has shown that for reasonable precubical sets (called geometric, see Definition 77) the converse is true:

Theorem 75 ([Faj05])**.** *Given a geometric locally finite precubical set, the induced functor* $\uparrow -\uparrow : \vec{\Pi}_1(C) \to \vec{\Pi}_1(\uparrow C\uparrow)$ *is full and faithful.*

The proof goes by suitably subdividing the d-spaces in "squares", and is a particular case of a general cubical approximation theorem [Kri13].

Many classical constructions and properties on topological spaces extend to the directed setting. For instance, Fajstrup has proposed a generalization of the notion of covering space in the directed setting [Faj03, FR08, Faj11, FGH$^+$16] (but her approach is not the only possible one [GHK09]), and the geometric realization of the unfolding of a geometric precubical set, see Section 2.3.2, is precisely its universal dicover [Fah05]. Also, analogue of the Seifert-van Kampen theorem were shown by Goubault and Grandis [Gou03, Gra03].

Other models. The notion of d-space we use here is far from being the only setting introduced in order to be able to consider directed paths.

Historically, the notion of *partially ordered space* (or *pospace*) came first [Eil41, Nac65], originating in the study of positive cones of topological vector spaces occurring in functional analysis: it consists in equipping a topological space with a partial order (whose graph is closed in the product topology). Those spaces are convenient to work with, but their main drawback is that they cannot represent spaces containing nontrivial directed loops, and thus cannot model programs with `while` loops.

In order to overcome this limitation, the notion of *local pospace* was introduced by Fajstrup, Raussen and Goubault [FRG06]. A local pospace is a suitable sheaf of pospaces, i.e. a space such that every point admits a neighborhood with a pospace structure in a coherent way. Unfortunately, the resulting category is rather ill-behaved: it is finitely complete but lacks infinite products, the natural embedding of the category of pospaces into the category of local pospaces is not full, some colimits of local pospaces do not preserve the topology, etc. Nevertheless most finite precubical sets can be realized in the category of local pospaces [FRG06]: this category is nice enough if we restrict to computer-scientific applications.

The notion of *stream*, introduced by Krishnan [Kri09], bears nicer categorical properties. A stream is a cosheaf of locally preordered spaces: it consists of a topological space such that each open subset U is equipped with a preorder in such a way that for all open coverings $(U_i)_{i\in I}$ of U, the preorder on U is the least preorder containing all the preorders on the U_i. This model turns out to be very close to d-spaces: there is an adjunction between the categories of streams and d-spaces, which induces an isomorphism between subcategories that can explicitly be characterized [Hau12].

The preceding list is not exhaustive: approaches based on model categories [Gau03, Gau08, Gau14b, BW06, Wor10], and locally presentable categories [FR08] have also been investigated in this context.

Geometric precubical sets. A reasonably simple and general form of topological space obtained by "gluing cubes" is the following one. Many geometric properties of these objects, which are part of the family of *polyhedral complexes*, are known [BH09], some of which will be used in Chapter 4.

Definition 76. A *cubical complex* K is a topological space of the form $K = \left(\bigsqcup_{\lambda\in\Lambda} I^{n_\lambda}\right)/\!\approx$ where Λ is a set, $(n_\lambda)_{\lambda\in\Lambda}$ is a family of natural numbers, and $\approx$ is an equivalence relation, such that, writing $p_\lambda : I^{n_\lambda} \to K$ for the restriction of the quotient map $\bigsqcup_{\lambda\in\Lambda} I^{n_\lambda} \to K$, we have

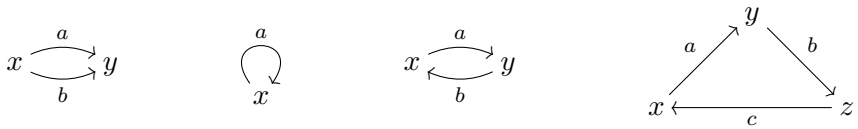
1. for every $\lambda \in \Lambda$, the map p_λ is injective,
2. given $\lambda, \mu \in \Lambda$, if $p_\lambda(I^{n_\lambda}) \cap p_\mu(I^{n_\mu}) \neq \emptyset$ then there is an isometry $h_{\lambda,\mu} : J_\lambda \to J_\mu$ from a face J_λ of I^{n_λ} to a face J_μ of I^{n_μ} such that $p_\lambda(x) = p_\mu(y)$ if and only if $y = h_{\lambda,\mu}(x)$.

A *directed cubical complex* is defined similarly, as d-space of the from $K = \left(\bigsqcup_{\lambda\in\Lambda} \vec{I}^{n_\lambda}\right)/\!\approx$.

Not every realization of a precubical set is a cubical complex, such as the graph $\cdot \rightrightarrows \cdot$ with two vertices and two edges. It turns out that a good characterization of precubical sets giving rise to cubical complexes is the following one:

Definition 77 ([Faj05])**.** A precubical set K is *geometric* when it has no self-intersection (two iterated faces in distinct directions of a cube in K are distinct) and common faces of largest dimension (two cubes admitting a common face admit a common face of largest dimension).

Example 78. Consider the following precubical sets of dimension 1



(1) (2) (3) (4)

(1) and (3) are not self-intersecting but fail to have largest common faces, (2) is self-intersecting and (4) is geometric.

We investigated those in [MG16]. The geometric precubical sets C can also be more abstractly characterized as those such that for every cube $x \in C$, the slice category $x/\operatorname{El}(C)$ is a meet semilattice, and moreover it can be shown that:

Proposition 79 ([MG16])**.** *The geometric realization of a geometric precubical set is a cubical complex, and the directed geometric realizations of geometric precubical sets are precisely the directed cubical complexes.*

Except for degenerated cases, the precubical semantics of programs satisfy this condition.

Invariants of directed topological spaces. As illustrated in Example 73, the classical invariants of topology (homotopy type, homotopy and homology groups, etc.) are rather weak since they do not distinguish between models which arise as geometric semantics of very different programs, and many attempts in order to find better notions in the directed case have been made. Directed notions of homotopy equivalence have been proposed [GG03, Gra05, Gau06, Gra09, Gau11, Gau14a], there are some approaches based on model categories [Gau03, BW06], directed variants of homology [GJ92, Gou95, Fah04, Gra04, Gau05, Kah13, DGG15].

3.2 Metric models

This section is mostly based on [MG16].

In order to gain more quantitative information about processes, it would be desirable to be able to use metric spaces instead of topological spaces: the idea is that the length of a path should correspond to the duration of its execution. We will see in Section 4.2 that this also provides us with new tools and notions, such as the curvature of the space. The basic idea here is to do "as usual": define a functor from the category $\square$ to metric spaces, which associates I^n with n (the product of n copies of the standard interval, seen as a metric space with the euclidean topology) and extend it cocontinuously in order to realize every precubical set. However, the category of metric spaces is unfortunately *not* cocomplete, and the previous methodology will not work on the nose. For instance, in the coproduct of two metric spaces, two points coming from the two distinct spaces should intuitively be at an infinite distance, and this is not allowed by the definition of a metric. This suggests that we should generalize the definition of metric spaces in order to obtain a category with better properties, and it turns out that the notion of metric space proposed by Lawvere [Law73] provides us with an interesting such category. The idea of using those as a directed model was also proposed by Grandis [GM03, Gra09]. Here, we provide a detailed study of this category and its relationships with other models. It will notably be used in Chapter 4 in order to link geometric properties of some programs with the curvature of corresponding metric models.

3.2.1 Generalized metric spaces

Recall that a *metric space* (X, d) consists of a set X equipped with a metric $d : X \times X \to [0, \infty]$, i.e. a function such that, given $x, y, z \in X$, we have

(1)	point equality:	$d(x, x) = 0$
(2)	triangle inequality:	$d(x, z) \leqslant d(x, y) + d(y, z)$
(3)	finite distances:	$d(x, y) < \infty$
(4)	separation:	$d(x, y) = d(y, x) = 0$ implies $x = y$
(5)	symmetry:	$d(x, y) = d(y, x)$

Definition 80. A *generalized metric space* is defined as above, but keeping only the first two axioms. We write **GMet** for the category of those spaces, with non-expansive functions as morphisms $f : X \to Y$, i.e. $d_Y(f(x), f(y)) \leqslant d_X(x, y)$ for every points $x, y \in X$.

This notion, also sometimes called hemi-metric space, was proposed by Lawvere based on the observation that it corresponds to a category enriched in $\mathcal{V} = [0, \infty]$ equipped with a suitable monoidal category structure [Law73]. Moreover, it encompasses most generalizations of metric spaces commonly found in literature: extended metrics (satisfying all axioms except (3)), pseudometrics (except (1) and (4)), quasimetrics (except (5)), hemimetrics (except (4) and (5)), etc. In the following, we will always refer to this notion when considering "metric spaces".

Example 81. Given $a, b \in \mathbb{R}$, we write $[a, b]$ for the interval equipped with the usual metric given by $d(x, y) = |y - x|$. We write $\overrightarrow{[a, b]}$ for the same interval metrized by $d(x, y) = y - x$ when $x \leqslant y$ and $d(x, y) = \infty$ when $x > y$. In particular, we often write I (resp. $\vec{I}$) instead of

$[0,1]$ (resp. $\overrightarrow{[0,1]}$) for the (*directed*) *standard interval*. Similarly, we write $\mathbb{R}$ (resp. $\vec{\mathbb{R}}$) for the (*directed*) *real line*. This space is sometimes called the *Sorgenfrey line* [Sor47], see also [GL13].

Example 82. The *directed unit circle* $\vec{\mathbb{S}}^1$ is the set of complex points of the form $\mathrm{e}^{\mathrm{i}2\pi\theta}$, with $\theta \in \mathbb{R}$, equipped with the distance $d(x,y) = \bigwedge \left\{ \rho - \theta \;\middle|\; x = \mathrm{e}^{\mathrm{i}2\pi\theta}, y = \mathrm{e}^{\mathrm{i}2\pi\rho}, \rho \geqslant \theta \right\}$.

3.2.2 Limits and colimits

Since the category $\mathcal{V}$ is complete and cocomplete, by general results about enriched categories [BCSW83], the category **GMet** has small limits and colimits. This will in particular allow us to properly define the metric realization of a precubical set in Section 3.2.6. More explicitly, those can be computed as follows:

Proposition 83 ([MG16])**.** *The forgetful functor* $\mathbf{GMet} \to \mathbf{Set}$ *admits both a left and a right adjoint, thus limits and colimits can be computed in* **Set** *and equipped with suitable distances:*

— *Given a family* $(X_i, d_i)_{i\in I}$ *of metric spaces, its product (resp. coproduct) is the set* $\prod_{i\in I} X_i$ *(resp.* $\coprod_{i\in I} X_i$*), respectively equipped with the distances*

$$d_{\prod X_i}((x_i),(y_i)) = \bigvee_{i\in I} d_i(x_i,y_i) \qquad\qquad d_{\coprod X_i}(x,y) = \begin{cases} d_i(x,y) & \text{if } x,y \in X_i \\ \infty & \text{otherwise} \end{cases}$$

— *Given a pair of morphisms* $f, g : (X, d_X) \to (Y, d_Y)$*, their equalizer is the equalizer set* $Z = \{x \in X \mid f(x) = g(x)\}$ *equipped with the restriction of* d_X *as distance.*

— *Given a pair of morphisms* $f, g : (X, d_X) \to (Y, d_Y)$*, their coequalizer is the set* $Y/\!\approx$*, where* $\approx$ *is the smallest equivalence relation such that* $y \approx y'$ *whenever there exists* $x \in X$ *with* $y = f(x)$ *and* $y' = g(x)$*, equipped with the following distance. Given two points* $x, y \in Y$*, a "chain" from* x *to* y *is a sequence of points* $x_1, y_1, x_2, y_2, \ldots, x_n, y_n \in Y$ *such that* $x = x_1$*,* $y_i \approx x_{i+1}$*, and* $y_n = y$*. The length of such a chain* u *is defined to be* $l(u) = \sum_{i=1}^{n} d(x_i, y_i)$*. The distance between two points* $x, y \in Y/\!\approx$ *is the infimum of the lengths of chains from a representative of* x *to a representative of* y*.*

Example 84. Consider the spaces (I_n, d_n) indexed by $n \in \mathbb{N}$ with $I_n = [0,1]$ and distance $d_n(x,y) = |y - x|/n$. The colimit $I_\omega = \coprod_{n\in\mathbb{N}} I_n/\!\approx$ where $\approx$ identifies points 0, resp. 1, in various I_n is the colimit of the sets equipped with the distance d_ω such that given $x \in I_n$ and $y \in I_m$,

$$d_\omega(x,y) \quad = \quad \begin{cases} 0 & \text{if } x,y \in \{0,1\} \\ |y-x|/n & \text{if } n = m \\ (x/n + y/m) \vee ((1-x)/n + (1-y)/m) & \text{otherwise} \end{cases}$$

In particular, we have $d_\omega(0,1) = 0$, which shows that coequalizers of separated spaces are not necessarily separated. Notice that, in the space I_ω, if we "cut in the middle" all the intervals we obtain two "star-shaped" spaces which are both separated. The space I_ω can then be obtained as a pushout of the two spaces (over the discrete space with $\mathbb{N}$ as points), showing that separated spaces are also not closed under pushouts (in **GMet**, however the category of quasi-metric spaces is complete and cocomplete for instance).

3.2.3 Symmetric spaces

The full subcategory **SGMet** of *symmetric* metric spaces is interesting. In particular, there is an obvious forgetful functor **SGMet** $\to$ **Top** sending a metric space (X, d) to the set X equipped with the topology generated by open balls, of the form $B^\varepsilon(x) = \{y \in X \mid d(x, y) < \varepsilon\}$ for some $x \in X$ and $\varepsilon > 0$: since the maps in **SGMet** are non-expansive, they are continuous wrt the induced topology. We will see that the situation is much less obvious in the case of (non-symmetric) metric spaces.

This category is closed under small limits and colimits and moreover,

Proposition 85 ([MG16])**.** *The forgetful functor* **SGMet** $\hookrightarrow$ **GMet** *admits a left adjoint sending a space* (X, d) *to the symmetric space* $(X, \overline{d})$ *where the metric* $\overline{d}$ *is called the* symmetric metric *generated by* d *and is defined by*

$$\overline{d}(x, y) \quad = \quad \bigwedge_{x=x_0,x_1,\ldots,x_{2n},x_{2n+1}=y} \sum_{i=0}^{n} d(x_{i+1}, x_i) + d(x_{i+1}, x_{i+2})$$

It also admits a right adjoint sending (X, d) *to the space* $(X, d^\vee)$ *where* $d^\vee(x, y) = d(x, y) \vee d(y, x)$*. The forgetful functor thus preserves limits and colimits.*

Example 86. For instance, consider the "directed plane" $\vec{\mathbb{R}}^2$, whose distance is given by $d((x_1, x_2), (y_1, y_2)) = d_{\vec{\mathbb{R}}}(x_1, y_1) \vee d_{\vec{\mathbb{R}}}(x_2, y_2)$, and $d_{\vec{\mathbb{R}}}$ is similar to the one on $\vec{I}$, see Example 81. The left and right adjoint of the proposition respectively equip $\mathbb{R}^2$ with the distances $\overline{d}$ and $d^\vee$ such that

$$\overline{d}((x_1, x_2), (y_1, y_2)) \quad = \quad \begin{cases} |y_1 - x_1| \vee |y_2 - x_2| & \text{if } (y_1 - x_1)(y_2 - x_2) \geqslant 0 \\ |y_1 - x_1| + |y_2 - x_2| & \text{if } (y_1 - x_1)(y_2 - x_2) \leqslant 0 \end{cases}$$

and

$$d^\vee((x_1, x_2), (y_1, y_2)) \quad = \quad \begin{cases} |y_1 - x_1| \vee |y_2 - x_2| & \text{if } (y_1 - x_1)(y_2 - x_2) \geqslant 0 \\ \infty & \text{if } (y_1 - x_1)(y_2 - x_2) \leqslant 0 \end{cases}$$

3.2.4 Underlying topological space

As mentioned above, for metric spaces it is not clear how one should define a forgetful functor **GMet** $\to$ **Top**. Before presenting our answer, we would first like to explain why other "intuitive" options are not satisfactory. Given a space (X, d) and $x \in X$, one can construct *past* and *future open balls* of radius $\varepsilon > 0$, defined by

$$B^\varepsilon_-(x) = \{y \in X \mid d(y, x) < \varepsilon\} \qquad\qquad B^\varepsilon_+(x) = \{y \in X \mid d(x, y) < \varepsilon\}$$

Considering the topology generated by future open balls is intuitive, but leads to a topology which is too fine: for instance, the map $f : \vec{I} \to \vec{I}$ such that $f(t) = 0$ if $t < 0.5$ and $f(t) = 1$ otherwise would be continuous wrt this topology. Another option could also consider the topology generated by sets of the form $B^\varepsilon_-(x) \cup B^\varepsilon_+(x)$. However, an easy computation shows that in $\vec{\mathbb{R}}^2$, the resulting topology is the discrete one. Namely, a generating open set is drawn on the left below (continuous lines mean that the border is included and dotted ones that the border is excluded). In the middle right, an open set is shown (it is obtained by intersecting

two generating open sets as drawn on the middle left). Finally, by intersecting two such open sets, we can obtain an open set reduced to a point, as shown on the right.

We hope to have convinced the reader that the forgetful functor **GMet** $\to$ **Top** should be defined as the composite **GMet** $\to$ **SGMet** $\to$ **Top** where the first functor **GMet** $\to$ **SGMet** is the left adjoint to the forgetful functor constructed in Proposition 85 and the second functor **SGMet** $\to$ **Top** is the forgetful functor described above. Notice that Example 86 illustrates why the left adjoint (and not the right one) is suitable here.

Remark 87. Given a symmetric space (X, d) and $x, x', y, y' \in X$, we have

$$|d(x', y') - d(x, y)| \quad \leqslant \quad |d(x', y') - d(x, y')| + |d(x, y') - d(x, y)| \quad \leqslant \quad d(x, x') \vee d(y, y')$$

when the distances are finite, the last step using the well-known "reverse triangle inequality". The distance d can thus be seen as a morphism $d : X \times X \to [0, \infty]$ in **SGMet** and is continuous wrt the induced topology. For non-symmetric metric spaces the distance is not necessarily continuous wrt to the induced topology. For instance, in $\vec{\mathbb{R}}^2$ (see the figure on the left),

when y "moves" vertically to y', the value of $d(x, y)$ suddenly "jumps" from a finite value to ∞. Similarly, in $\vec{\mathbb{S}}^1$ (figure on the right), when y moves to y', the value of $d(x, y)$ jumps from 0 to 2π.

Proposition 88 ([MG16])**.** *The forgetful functor* **GMet** $\to$ **Top** *preserves finite limits and small coproducts.*

Remark 89. The functor does not preserve coequalizers. Namely, if we consider the colimit of Example 84, the colimit as topological spaces has points 0 and 1 separated in I_ω, whereas $d_\omega(0, 1) = 0$ and thus the points are not separated in the topological space associated with the colimit of generalized metric spaces.

3.2.5 Directed paths

A *path* in a metric space (X, d) is a continuous function $\gamma : I \to X$ (which is not required to be non-expansive). Its length $\|\gamma\| \in [0, \infty]$ is defined by

$$\|\gamma\| \quad = \quad \bigvee_{n \in \mathbb{N}} \bigvee_{0 = t_0 < t_1 < \ldots < t_n = 1} \sum_i d(\gamma(t_i), \gamma(t_{i+1}))$$

When the length is finite the path is called a *dipath* (or a rectifiable path). One can also define a *dihomotopy* between dipaths, as a homotopy whose intermediate paths are directed. The terminology comes from the fact that, in the case of generalized metric spaces, the metric encodes a notion of direction as illustrated in following example.

Example 90. In the directed circle $\vec{\mathbb{S}}^1$ (see Example 82), directed paths are those which are "turning counter-clockwise", i.e. maps of the form $t \mapsto \mathrm{e}^{\mathrm{i}\theta(t)}$ where $\theta : I \to [0, 2\pi]$ is increasing modulo 2π.

Since maps are required to be non-expansive, we cannot expect every dipath to be represented by a morphism $I \to X$, but the following can be shown:

Lemma 91. *Given a separated space X, rectifiable traces are in bijection with morphisms $\overrightarrow{[0, a]} \to X$ sending distance to length, for some $a \geqslant 0$.*

Proposition 92 ([Gra03, MG16])**.** *The operation which, with a metric space, associates its underlying topological space together with the set of directed paths defines a functor* $\mathbf{GMet} \to \mathbf{dTop}$.

3.2.6 Geometric realization of precubical sets

Since the category **GMet** is cocomplete, we can define the metric realization of a precubical set as follows.

Definition 93 ([MG16])**.** The *metric realization* (resp. *directed metric realization*) of a precubical set C is its image $|C|$ (resp. $\uparrow C \uparrow$) under the cocontinuous extension of the functor $\square \to \mathbf{GMet}$ sending an object n to I^n (resp. $\vec{I}^n$). The (*directed*) *metric semantics* of a program p is $\left|\check{C}_p\right|$ (resp. $\uparrow \check{C}_p \uparrow$).

We sometimes write $|C|_{\mathbf{Top}}$ and $|C|_{\mathbf{GMet}}$ to distinguish between the geometric realization in topological spaces and metric spaces, etc.

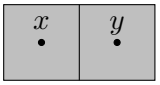
Example 94. For instance the metric semantics of the program $(A \parallel B)\,;C$ is shown on the left below:

Notice that the length of a path corresponding to an execution is precisely the duration of the corresponding execution if we assume that every action takes a unit of time to be performed. In particular, the semantics of two actions in parallel, such as `A` and `B` above, is the cartesian product of their semantics, where the distance is the maximum of the componentwise distances (see Proposition 83): the time taken to run two processes in parallel is the maximum of the time taken to run each process.

In order to show that these notions are reasonable and conservatively extend previously established constructions, we have shown two important properties that they satisfy. Firstly, the realization commutes with the forgetful functor to topological spaces. Secondly, the dipaths in the directed metric semantics coincide, up to dihomotopy, with execution paths of the program.

Topology of the metric realization. One of the main technical tools in order to show the first property, is the notion of *escape distance*, which is inspired by [BH09]. Given a point $x \in |C|$, this distance $\varepsilon(x)$ is defined as the infimum over cubes c containing x in their realization of the distance from x to a face of c not containing x. Its main interest is the fact that, given a point y contained in a cube which does not also contain x, a path from x to y

will have length at least $\varepsilon(x)$. For instance, consider the following realization, consisting of two (filled) unit squares:



We have $\varepsilon(x) = 0.5$ and clearly no path from x to y can be smaller than this, because it has to go through the segment face in the middle. In particular, given points $x, y \in |C|$ belonging to the realization of a common cube c, and such that $d_{|C|}(x, y) < \varepsilon(x)$, the distance between x and y in $|C|$ coincides with the distance in the realization of c. In the case of geometric precubical sets, we have $\varepsilon(x) > 0$ for every point $x \in |C|$, see [BH09, Section 7.33], and therefore the distance in the realization locally coincides with the distance in I^n.

Theorem 95 ([MG16])**.** *The geometric realization of a locally finite geometric precubical set C commutes with the forgetful functor $U : \mathbf{GMet} \to \mathbf{Top}$, i.e. $U\left(|C|_{\mathbf{GMet}}\right) \cong |C|_{\mathbf{Top}}$.*

Proof. The proof, which is technical, is summarized here. First, the property can be shown in the case where C is finite as follows. As a colimit, the metric realization of C is the quotient of the space $X = \coprod_{n\in\mathbb{N}} \coprod_{x\in C(n)} I(n)$ by a relation R. The space UX is compact, as a finite coproduct of compact spaces, and thus $(UX)/R$ too. Moreover, since the spaces I^n are separated, and C is without self-intersections (because it is geometric), the space $U(X/R)$ is also separated, because the distance locally coincides with the one in I^n as explained above. Finally, the identity function $(UX)/R \to U(X/R)$ can be shown to be continuous. It is therefore a homeomorphism, because any continuous function $f : X \to Y$ with X compact and Y separated is so, by a well-known general theorem.

This extends to the case where C is only supposed to be locally finite as follows. Given a point $x \in |C|$, its *carrier* c is the smallest cube of C which contains x in its realization, and we write C_c for the *link* of c, i.e. the smallest precubical subset of C containing the cubes of C having c as iterated face. Because C is locally finite, C_c is finite and we can apply the above reasoning. Since the distance is locally determined by the one in I^n, every point x in $|C|$ has a neighborhood which is isometric to a neighborhood of x in $|C_c|$, and we conclude. □

Because of Proposition 85, and the definition of the forgetful functor $\mathbf{GMet} \to \mathbf{Top}$, the above theorem can also easily be applied to the directed geometric realization.

Dihomotopy classes of dipaths. Given a precubical set C, we have seen in Theorem 75 that the dipaths up to dihomotopy in C coincide with dipaths up to dihomotopy in $\uparrow C \uparrow_{\mathbf{dTop}}$. The proof of [Faj05] adapts to the metric setting:

Theorem 96 ([MG16])**.** *Given a geometric locally finite precubical set, the induced functor $\uparrow - \uparrow_{\mathbf{GMet}} : \vec{\Pi}_1(C) \to \vec{\Pi}_1(\uparrow C \uparrow_{\mathbf{GMet}})$ is full and faithful.*

Other properties of the realization. Finally, we briefly list some other interesting properties of the metric realization of a precubical set C which are shown in [MG16] and apply to metric semantics of programs:

— it is separated when C is locally finite (see the proof of Theorem 95),
— it is a *length space*: the distance between two points x and y is the infimum of lengths of paths from x to y,
— it is complete when C is finite-dimensional,

— it is geodesic when C is finite-dimensional.

Moreover, we conjecture that the canonical embedding of an n-cube into the realization of a finite-dimensional geometric precubical set is an isometry.

3.3 Computing the trace space

This section is mostly based on [FGH+12].

We would finally like to briefly report on the work we did in order to improve and implement an algorithm originally proposed by Raussen [Rau10], and refer to the article [FGH+12] for details: properly presenting the algorithm would require introducing too much material for this memoir, and moreover most of the theory was developed by Raussen, not us. This section does not use the metric notions introduced in previous section. We have handled the case of *simple programs*, which are of the form $p = p_0 \,\|\, p_1 \,\|\, \ldots \,\|\, p_{n-1}$ where the processes p_i are sequences of actions (they may contain manipulations of resources of arbitrary capacity, but no control-flow construction such as branching and loops). It has since then been extended to more general programs [Rau12a, Rau12b, Faj14, FGH+16], but this restriction makes the semantics particularly simple to handle while capturing an important class of programs: in this case, the geometric semantics G_p consists of an n-cube minus l removed k-cubes, corresponding to forbidden regions and called *holes*, with $0 \leqslant k \leqslant n$,

The *trace space* of the geometric semantics is the topological space of dipaths from the beginning to the end point, up to reparametrization. Our goal is to compute its path-connected components, i.e. the set of dipaths up to dihomotopy, corresponding from an operational point of view to the behaviors the programs can exhibit. Using a boolean matrix M of size $l \times n$ (the number l of holes times the number n of processes), one can encode a subspace G_M of G_p obtained by infinitely extending downwards each hole in a number of directions (we extend the i-th hole in the direction j whenever $M_{i,j} = 1$). When each hole is extended in at least one direction (the matrix has non-null rows) the corresponding trace space can be shown to be either empty or contractible, i.e. it contains at most one dihomotopy class of traces. Moreover, whether the trace space of G_M is empty or not can be efficiently tested using the characterization of deadlocks in geometric semantics of simple programs [FGR98, FGH+16]; a matrix for which the trace space is non-empty is called *alive*. Finally, one can determine whether two matrices correspond to a same dihomotopy class of traces directly by computing the intersection of matrices (a simple operation that we do not detail here); we say that two matrices are *connected* when this is the case. Therefore one can compute connected components of the trace space by computing connected components of alive matrices. The resulting algorithm has been implemented as a prototype and was shown to be efficient on simple cases, for instance execution times on the example of n dining philosophers (Example 10) are comparable to those of the model-checker SPIN [Hol04], even though this latest program is much optimized whereas our implementation could be largely improved, but consumes much less memory (for 13 philosophers, we use around 60 MB whereas SPIN is using swap and does not terminate in a reasonable time).

In fact, not only the connected components of the trace space can be recovered in this way, but also the higher-dimensional geometric structure: the alive matrices can be organized into a prod-simplicial set (a presheaf whose representables can be thought of as products of simplices, see [Koz08]), whose geometric realization can be shown to be homotopy equivalent to the trace space [Rau10]. Using this fact it can be shown that very general spaces can be

obtained as trace spaces of simple programs; for instance Ziemiański has shown that for any finite dimensional simplicial complex S on n vertices, there exists a simple program p with n threads whose trace space is homotopy equivalent to the disjoint union of the complex S and an $(n-2)$-dimensional sphere [Zie15].

Chapter 4

Programs with mutexes only

Among the low-level synchronization primitives, mutexes are largely the most used ones. This motivates our study, in this chapter, of the specific case of programs using only this kind of resources. We study their geometric semantics and show that they satisfy two notable properties.

Firstly, we show that the metric semantics is *non-positively curved*, or NPC for short, in the sense of Gromov [Gro87]. We believe that this is an important observation since, in many ways, non-positively curved spaces are "simpler" to work with than general spaces: topological invariants are much simpler, e.g. the homology and homotopy groups have no torsion, their universal covering space is contractible [BH09], etc. Geometrically, such spaces can be characterized as those in which going from one side to the other of a (geodesic) triangle does not take longer than it would in the plane. Examples of spaces which do not have this property are easily given. For instance, consider a triangle drawn on an a sphere, as on the left below:

One can easily be convinced that the shortest path to go from the point x to the point y of the triangle, drawn with dots, is longer than the corresponding one on the plane: the sphere tends to "increase" distances and is thus not NPC. Of course, a similar reasoning would hold with an empty cube as shown on the right. However, if the cube is filled then this is not true anymore, and in fact a filled cube is NPC. Actually, among spaces obtained by gluing cubes (roughly geometric realizations of precubical sets), those which are NPC can be characterized as those in which every border of a cube is filled. This motivates our introduction of the notion of a "non-positively curved precubical complex", whose metric realization is always NPC, using an algebraic counterpart of this property. Interestingly, the resulting notion fundamentally relies on the *cube property*, already encountered (Sections 2.4.2 and 2.5.1), which characterizes systems in which conflict is binary. It is thus not so unexpected that programs using mutexes only should give rise to such spaces, since mutexes precisely express binary exclusions.

Secondly, by definition, two dihomotopic paths are homotopic, but the converse is not generally true. We however show that it always holds for spaces arising as semantics of

programs using only mutexes, and our main tool to show it is our characterization of NPC precubical complexes.

We begin by introducing the notion of non-positively curved precubical set, of which precubical models of programs with mutexes are an instance (Section 4.1), and then relate those to traditional non-positively curved metric spaces (Section 4.2). Finally, we show the coincidence of homotopy and dihomotopy in those models (Section 4.3).

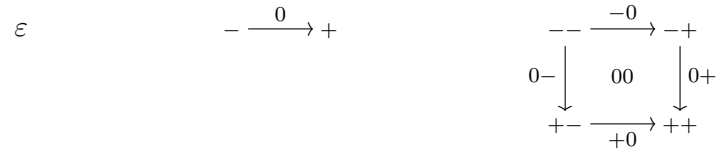
4.1 Non-positively curved precubical sets

4.1.1 A definition

In order to be able to formulate the notion of NPC for precubical sets, we will need the following very useful combinatorial description of the representable precubical sets [Cra95]. We write $Y : \square \to \hat{\square}$ for the Yoneda embedding.

Lemma 97. *Given $n \in \square$, the cubes in Yn are in bijection with strings in $\{-, 0, +\}^n$, the k-cubes in Ynk being the strings containing the letter 0 exactly k times, and given $u \in Ynk$, the face $\partial_i^-(u)$ (resp. $\partial_i^+(u)$) is obtained from u by replacing the i-th letter 0 by $-$ (resp. $+$).*

Example 98. For instance, with $n = 0, 1, 2$, we have



(here ε denotes the empty word).

We have $Ynk = \emptyset$ for $k > n$ and there is only one element in Ynn (the string 0^n). The *standard hollow n-cube* (or the *border of the n-cube*), denoted ∂Yn, is the precubical set obtained from Yn by removing the cell in Ynn. A fact that will sometimes be useful in the following is that the faces of the standard n-cubes naturally index morphisms computing iterated faces of an n-cube: given a precubical set C, an n-cube $x \in C(n)$ and $u \in \{-, 0, +\}^n$, we write

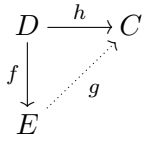
$$\partial^u(x) \quad = \quad \partial_0^{u_0} \circ \partial_1^{u_1} \circ \ldots \circ \partial_{n-1}^{u_{n-1}}(x)$$

where $u_i \in \{-, 0, +\}$ is the i-th letter of u and by convention $\partial_i^0(x) = x$. We also need to introduce the following family of precubical sets.

Definition 99. Given $u \in \{-, +\}^n$, we write Λ^u for the precubical subset of ∂Yn whose cubes $v \in \{-, 0, +\}^n$ are those having at least one letter in common with u (i.e. there exists i, with $0 \leqslant i < n$, such that $v_i = u_i$). The precubical set Λ^u can alternatively be defined from ∂Yn by removing all cubes having the vertex $\overline{u}$ as iterated face, where $\overline{u} = \overline{u_0} \ldots \overline{u_{n-1}}$ with $\overline{-} = +$ and $\overline{+} = -$.

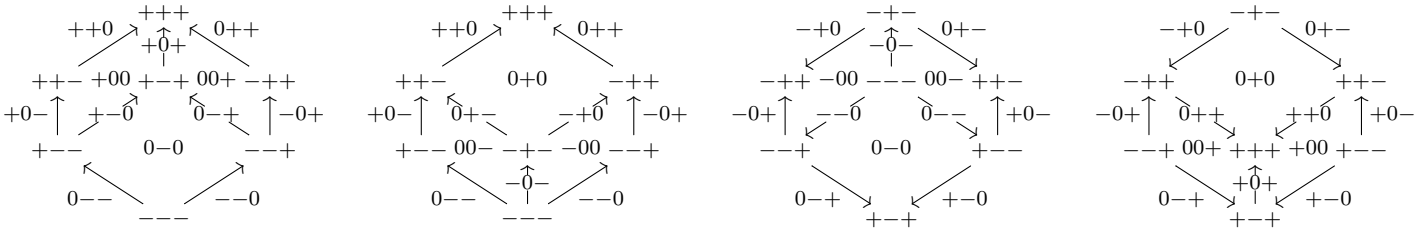
Definition 100. Given a morphism $f : D \to E$ between precubical sets, we say that a precubical set C *lifts* the morphism f, when for every morphism $h : D \to C$, there exists a

unique morphism $g : E \to C$ such that $h = g \circ f$.



One of the fundamental axioms satisfied by NPC precubical sets are the following ones: the first intuitively states that if a precubical set contains half the border of a 3-cube then it also contains the other half of the border, and the other one that if it contains the border of an n-cube then it contains a unique n-cube having it as border.

Definition 101. A precubical set C has the *cube property* when it lifts the canonical inclusions of Λ^{+-+}, Λ^{-+-}, Λ^{---} and Λ^{+++} into $\partial Y3$:



Λ^{+-+} Λ^{-+-} Λ^{---} Λ^{+++}

We say that C has the *n-cube filling property* when it lifts the canonical inclusion $\partial Yn \hookrightarrow Yn$.

The precubical sets of interest in the following can be characterized as follows:

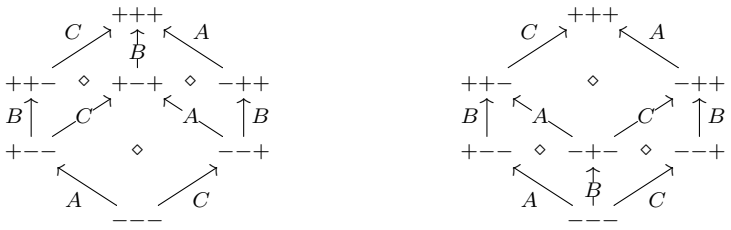
Definition 102 ([MG16])**.** A precubical set is *non-positively curved* (or NPC) when

1. it is geometric,
2. it satisfies the cube property,
3. it satisfies the n-cube filling property for every $n \geqslant 3$.

Proposition 103. *The precubical semantics of a program (with mutexes only) is* NPC.

Proof. Given a program p, it easy to show that the precubical set C_p (before removing forbidden vertices) is NPC: one can check that the required properties are preserved by gluing along vertices and taking tensor product. The precubical semantics is defined as $\check{C}_p = C_p \setminus X$ where X is the set of forbidden vertices. It is easy to show that properties 1. (geometricity) and 3. (lifting of $\partial Yn \hookrightarrow Yn$ for $n \geqslant 3$) above are still satisfied for $\check{C}_p$. However, the cube property requires a little more care. Writing $L = \Lambda^{+-+}$ and $R = \Lambda^{-+-}$, we notice that the hollow 3-cube $\partial Y3$ can be obtained by gluing L and R along their "border" (i.e. by identifying the vertices and edges with the same names in both precubical sets). Suppose given a morphism $L \to \check{C}_p$: we are going to show that it can be extended as a morphism $\partial Y3 \to \check{C}_p$. By abuse of notation, we identify L with its image in $\check{C}_p$ and simply say that $\check{C}_p$ "contains" L. Since C_p satisfies the cube property, it also contains R. Thus, in $\check{C}_p$, L can be completed as a hollow 3-cube unless the vertex $-+-$ is forbidden. We write A, B and C for the actions

respectively labeling the edges of the form $0\epsilon\epsilon'$, $\epsilon 0\epsilon'$ and $\epsilon\epsilon' 0$, with $\epsilon, \epsilon' \in \{-,+\}$: L and R in C_p are respectively labeled as



Suppose that $-+-$ is forbidden. Since none of the vertices in L is forbidden, there exists a resource a such that either $r_p(-+-)(a) = 1$ and B is V_a, or $r_p(-+-) = -1$ and B is P_a. Suppose that we are in the first case (the other case is similar), i.e. $B = V_a$. Notice that we have $r_p(---) = 0$. Necessarily, we have $A = P_a$ (otherwise $r_p(++-) = r_p(-+-) = 1$ and the vertex $++-$ would be forbidden) and $C = P_a$ (otherwise $-++$ would be forbidden). But in this case, we have $r_p(+-+) = -2$ and therefore $+-+$ would be forbidden, contradicting the hypothesis that $\check{C}_p$ contains L. All other cases can be handled by similar reasoning. □

We finally mention a useful generalization of the cube property. In an NPC precubical set, if we have half the border of a cube, then we have the other half by the cube property and therefore we have the cube itself. This can be formally stated as the fact that we have a lifting of the canonical inclusion $\Lambda^u \to Yn$ for $u \in \{-,+\}^3$. In fact, it can be shown that this is also true for higher-dimensional cubes:

Proposition 104 ([MG16])**.** *An* NPC *precubical set C lifts all inclusion morphisms $\Lambda^u \hookrightarrow Yn$ for $n \geqslant 3$ and $u \in \{-,+\}^n$.*

4.1.2 The link condition

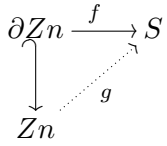
We now show that our definition allows us to show a property which is a combinatorial analogue of Gromov's link condition, which is recalled in Section 4.2. It is generally expressed in terms of a "flagness" condition on the "link" of vertices in the complex. We extend here the definition of link to precubical sets and show that a similar characterization of NPC can be formulated in terms of those links.

Definition 105. Given a precubical set C the *link* of C, written $\mathrm{link}(C)$, is the (augmented) presimplicial set whose n-simplices are pairs (u, y) with $u \in \{-,+\}^n$ and $y \in C(n)$. Given $i \in [n]$, the i-th simplicial face of such an n-simplex (u, y) is given by $\partial_i(u, y) = (u', \partial_i^{u_i}(y))$ where u' is the word u with the i-th letter removed. Given a vertex $x \in C(0)$, the *link* of x is the presimplicial subset $\mathrm{link}(x)$ of $\mathrm{link}(C)$ consisting of simplices having (ε, x) as iterated face.

Example 106. Consider the precubical set C on the left. The link of the vertex x is shown on the right:

Geometric precubical sets which are non-positively curved can be characterized by a lifting condition on the links. We write $Z : \triangle \to \hat{\triangle}$ for the Yoneda embedding of the presimplicial category into presimplicial sets (in order to avoid confusions, the notation Y is reserved here for the Yoneda embedding $Y : \square \to \hat{\square}$). We have $Znk = \emptyset$ for $k > n$ and Znn is reduced to one element (the simplex $[n]$). The *standard hollow n-simplex* ∂Zn is the presimplicial set obtained from Zn by removing the top-dimensional simplex in Zn, and there is a canonical inclusion $\partial Zn \hookrightarrow Zn$.

Definition 107. A presimplicial $S \in \hat{\triangle}$ set is *flag* if, for every natural number $n \geqslant 3$ and morphism $f : \partial Zn \to S$, there exists a unique morphism $g : Zn \to S$ making the following diagram commute, where the vertical arrow is the standard inclusion:



In Example 106, one can see that a point in the link corresponds to an edge in the precubical set, a 1-simplex to a square, and so on. This can be formally stated as follows, and shown by a direct application of the Yoneda lemma:

Lemma 108 ([MG16])**.** *Given a precubical set C and a vertex $x \in C(0)$, there is a bijection between presimplicial morphisms $Zn \to \mathrm{link}(x)$ (resp. $\partial Zn \to \mathrm{link}(x)$) and morphisms $Yn \to C$ (resp. $\square^u \to C$) sending u to x.*

Finally, using this correspondence and Proposition 104, one can show the following characterization of NPC precubical sets.

Theorem 109 ([MG16])**.** *A geometric precubical set C is* NPC *if and only if for every vertex $x \in C(0)$ the presimplicial set* $\mathrm{link}(x)$ *is flag.*

4.2 Non-positively curved spaces

4.2.1 CAT(0) spaces

We now recall the classical notion of NPC space, with the aim of showing that this notion is a topological counterpart of the notion introduced for precubical sets (Definition 102). This notion has led to numerous developments, but we only mention basic definitions here, and refer the reader to standard texts for a more detailed presentation of the subject [Gro87, BH09, GdLH$^+$90]. In the rest of this section, for simplicity, we only consider symmetric generalized metric spaces, since the topology of a space does not depend on its direction.

Definition 110. A *geodesic triangle* $\Delta(x, y, z)$ in a geodesic metric space X consists of three points x, y, z and three geodesics joining any pair of two. A *comparison triangle* for a geodesic triangle $\Delta(x, y, z)$ consists of an isometry $\overline{-} : \Delta(x, y, z) \to \mathbb{R}^2$ whose image is a geodesic triangle $\overline{\Delta}(\overline{x}, \overline{y}, \overline{z})$, where $\mathbb{R}^2$ is equipped with the usual euclidean distance $d_{\mathbb{R}^2}$.

We now recall the definition of non-positively curved space based on the comparison axiom of Cartan, Alexandrof and Topogonov. This is the origin of the name CAT(0), where the 0 refers to the fact that strictly positive or negative curvature can also be defined in a similar way.

Definition 111. A geodesic triangle $\Delta(x, y, z)$ is *CAT(0)* when there exists a comparison triangle $\overline{\Delta}(\overline{x}, \overline{y}, \overline{z})$ such that for any two points $p, q \in \Delta(x, y, z)$, we have $d(p, q) \leqslant d_{\mathbb{R}^2}(\overline{p}, \overline{q})$. A geodesic metric space is *CAT(0)* when every geodesic triangle is CAT(0), and *locally CAT(0)* or *non-positively curved* (or NPC) when every point admits a neighborhood which is CAT(0).

These spaces enjoy many nice properties such as being (locally) uniquely geodesic and contractible. We refer the reader to [BH09] for more details about those.

Other notions of curvature can be defined if we consider other "model spaces" instead of $\mathbb{R}^2$ in which we take our comparison triangles. In particular, we can consider the standard 2-sphere $\mathbb{S}^2$ equipped with the distance d such that $d(x, y) \in [0, \pi]$ is defined by $\cos(d(x, y)) = \langle x, y \rangle$. A space is CAT(1) when for every triangle $\Delta(x, y, z)$ whose diameter is less than 2π, there exists a comparison triangle $\overline{\Delta}(\overline{x}, \overline{y}, \overline{z})$ in $\mathbb{S}^2$, such that we have $d(p, q) \leqslant d_{\mathbb{S}^2}(\overline{p}, \overline{q})$ for every points $p, q \in \Delta(x, y, z)$.

4.2.2 The Gromov link condition

In this section we recall the characterization given by Gromov [Gro87] of CAT(0) cubical complexes, see also [BH09, II.5.4 and II.5.20]. We first introduce some necessary definitions.

Definition 112 ([BH09, 7.14])**.** Given a geodesic metric space X and a point $x \in X$, the *link* $\mathrm{link}_X(x)$ of x in X is the set of geodesics $\gamma : [0, a] \to X$ such that $\gamma(0) = x$, equipped with the compact-open topology, quotiented by the relation identifying two paths which coincide on an interval of the form $[0, \varepsilon[$ for some $\varepsilon > 0$, i.e. the "directions" from x pointing into X. This space can be metrized using the angle between two such directions.

This notion of link can be formally related to the one introduced for precubical sets in Definition 105 by observing that the former is a geometric realization of the latter (where the standard n-simplex is realized as a spherical simplex with edges of length $\pi/2$).

Definition 113. An abstract simplicial complex is *flag* if whenever the complex contains the 1-skeleton of a simplex, it also contains the simplex: given vertices $x_1, \ldots, x_k$ such that $\{x_i, x_j\}$ are simplices for every pair of indices i, j, the set $\{x_1, \ldots, x_k\}$ is also a simplex.

Again, this notion corresponds to the one of Definition 107 through the geometric realization.

Finally, we can explain in which way our conditions for non-positively curved precubical sets (Definition 102) correspond to the traditional geometric one.

Theorem 114 ([MG16])**.** *Given a finite dimensional geometric precubical set C, the following are equivalent:*

(i) $|C|$ is non-positively curved
(ii) in $|C|$ the link of every point is CAT(1)
(iii) in $|C|$ the link of every vertex is a flag complex
(iv) in C the link of every vertex is a flag complex
(v) C satisfies the cube condition

Moreover, the following are equivalent:

(i') $|C|$ is CAT(0)
(ii') $|C|$ is uniquely geodesic
(iii') $|C|$ is non-positively curved and simply connected
(iv') C satisfies the cube condition and is simply connected

Proof. First, notice that because C is supposed to be finite dimensional, its realization is geodesic, see Section 3.2.6. The equivalence between (i), (ii) and (iii) is due to Gromov [Gro87], see also [BH09, Thm. 5.20]. The equivalence between (iii) and (iv) is immediate and the equivalence between (iv) and (v) was shown in Theorem 109. The equivalence between (i'), (ii') and (iii') is due to Gromov [BH09, Thm. 5.5]. The equivalence between (iii') and (iv') follows from the equivalence between (i) and (v). □

4.2.3 Towards more general spaces

The notion of NPC precubical set can be relaxed in many ways, and we would like to investigate the properties of the resulting spaces. In particular, in light of the correspondence between event structures and precubical sets (see Section 2.4.2), the cube property can be subdivided in two conditions: one expresses a *stability* property, roughly the fact that dependency is generated by a partial order and not some more complicated relation, and the other one expresses the fact that conflicts are binary. The spaces satisfying the stability part only are interesting (they correspond to general event structures) and the corresponding spaces seem to satisfy a "directed variant" of the CAT(0) condition. Also, we have investigated spaces generated by programs using mutexes only: can we characterize those generated by resources of capacity at most k? One might expect the resulting precubical sets to satisfy lifting properties starting from dimension $k+1$ instead of 3, but again this is left for future work.

4.3 Homotopy and dihomotopy

As explained in Chapter 1, the dipaths in the precubical semantics $\check{C}_p$ of a program p correspond to executions of the program and the dihomotopy equivalence relation on them (Definition 34) is an approximation to observational equivalence between them, i.e. in order to study a program it is enough to consider its dipaths up to dihomotopy. However, from a topological point of view, the homotopy relation (Definition 34) is much more natural – and studied – than dihomotopy. By definition, two dipaths which are dihomotopic are necessarily homotopic, but the converse is not generally true, as illustrated in Example 35. We show here that for NPC precubical sets however, the two relations coincide.

4.3.1 The fundamental 2-category

We supposed fixed an NPC precubical set. In order to show our property, we will show that any homotopy between two dipaths can be rewritten as a dihomotopy. For this reason, we do not want to quotient dipaths right away, as in the definition of the fundamental category or groupoid (Definition 36), which suggests introducing the following 2-categories.

Definition 115 ([MG16])**.** The *fundamental 2-category* $\vec{\Pi}_2(C)$ associated with C is the 2-category whose

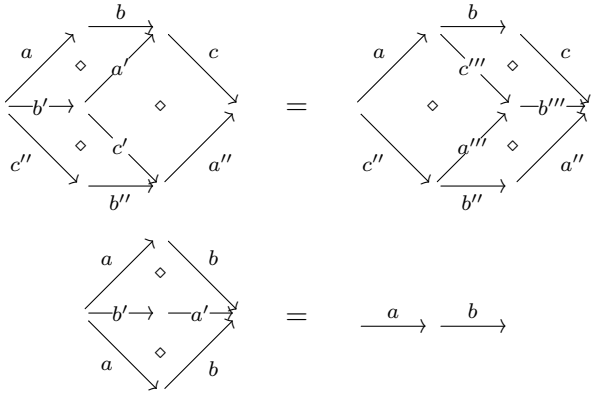
— 0-cells are the vertices of C,
— 1-cells are generated by (non-reversed) edges of C: 1-cells are dipaths in C and composition is given by concatenation,
— 2-cells are freely generated by

$$\gamma^{a,b}_{b',a'} \quad : \quad a \,.\, b \quad \Rightarrow \quad b' \,.\, a'$$

whenever a, b, a', b' are transitions such that $a \,.\, b \diamond b' \,.\, a'$, and quotiented by the smallest congruence (wrt both compositions) such that

$$\begin{aligned}(\gamma^{b',c'}_{c'',b''} \circ \mathrm{id}_{a''}) \circ (\mathrm{id}_{a'} \,.\, \gamma^{a',c}_{c',a''}) \circ (\gamma^{a,b}_{b',a'} \,.\, \mathrm{id}_c) &= (\mathrm{id}_{c''} \,.\, \gamma^{a''',b'''}_{b'',a''}) \circ (\gamma^{a,c'''}_{c'',a'''} \,.\, \mathrm{id}_{b'''}) \circ (\mathrm{id}_a \,.\, \gamma^{b,c}_{c''',b'''}) \\ \gamma^{b',a'}_{a,b} \circ \gamma^{a,b}_{b',a'} &= \mathrm{id}_{a\,.\,b} \end{aligned} \tag{4.1}$$

for transitions such that all the involved morphisms are defined. Graphically,



The horizontal composition will be denoted as concatenation (.), in sequential order, whereas the vertical composition will be denoted as usual ($\circ$), in categorical order, thus following the usual convention for monoidal categories.

Two dipaths $f, g : x \twoheadrightarrow y$ in C are dihomotopic if and only if there exists a 2-cell $\alpha : f \Rightarrow g$ in $\vec{\Pi}_2(C)$. In the 2-category $\vec{\Pi}_2(C)$, if $\gamma^{a,b}_{b',a'}$ is defined, then the transitions a' and b' are uniquely determined from a and b because C is geometric as an NPC precubcial set. Moreover, if $\gamma^{a,b}_{b',a'}$ is defined then $\gamma^{a',b'}_{a,b}$ is also defined, and is an inverse for $\gamma^{a,b}_{b',a'}$ by (4.1): in fact, the 2-category is a category enriched in groupoids.

Definition 116 ([MG16])**.** The *fundamental 2-groupoid* $\Pi_2(C)$ associated with C is the 2-category whose

— 0-cells are the vertices of C,

— 1-cells are generated by edges of C or their reverse: 1-cells are paths in C and composition is given by concatenation,

— 2-cells are freely generated by

$$\gamma^{a,b}_{b',a'} \quad : \quad a \,.\, b \quad \Rightarrow \quad b' \,.\, a'$$

whenever a, b, a', b' are possibly reversed transitions such that $a \,.\, b \diamond b' \,.\, a'$, and

$$\begin{aligned} \eta_a \quad &: \quad \mathrm{id}_x \quad \Rightarrow \quad a \,.\, \overline{a} \\ \varepsilon_a \quad &: \quad \overline{a} \,.\, a \quad \Rightarrow \quad \mathrm{id}_y \end{aligned}$$

whenever $a : x \to y$ is a possibly reversed transition, and quotiented by the smallest congruence (wrt both compositions) such that

$$(\gamma^{b',c'}_{c'',b''} \circ \mathrm{id}_{a''}) \circ (\mathrm{id}_{a'} \,.\, \gamma^{a',c}_{c',a''}) \circ (\gamma^{a,b}_{b',a'} \,.\, \mathrm{id}_c) = (\mathrm{id}_{c''} \,.\, \gamma^{a''',b'''}_{b'',a''}) \circ (\gamma^{a,c'''}_{c'',a'''} \,.\, \mathrm{id}_{b'''}) \circ (\mathrm{id}_a \,.\, \gamma^{b,c}_{c''',b'''}) \tag{4.2}$$

$$\gamma_{a,b}^{b',a'} \circ \gamma_{b',a'}^{a,b} = \mathrm{id}_{a\,.\,b} \tag{4.3}$$

$$(\mathrm{id}_a \,.\, \varepsilon_a) \circ (\eta_a \,.\, \mathrm{id}_a) = \mathrm{id}_a \tag{4.4}$$

$$(\varepsilon_a \,.\, \mathrm{id}_{\overline{a}}) \circ (\mathrm{id}_{\overline{a}} \,.\, \eta_a) = \mathrm{id}_{\overline{a}} \tag{4.5}$$

$$\varepsilon_{\overline{a}} \circ \eta_a = \mathrm{id}_{\varepsilon} \tag{4.6}$$

$$\gamma_{\overline{a'},a'}^{a,\overline{a}} \circ \eta_a = \eta_{\overline{a'}} \tag{4.7}$$

$$\varepsilon_{a'} \circ \gamma_{\overline{a}',a'}^{a,\overline{a}} = \varepsilon_{\overline{a}} \tag{4.8}$$

$$(\gamma_{b,a'}^{a,b'} \,.\, \mathrm{id}_{\overline{a'}}) \,.(\mathrm{id}_a \,.\, \gamma_{b',\overline{a'}}^{\overline{a},b}) \circ (\eta_a \,.\, \mathrm{id}_b) = \mathrm{id}_b \,.\, \eta_{a'} \tag{4.9}$$

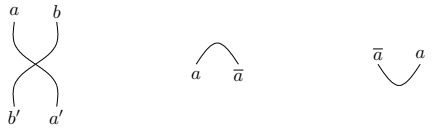
$$(\mathrm{id}_{a'} \,.\, \gamma_{\overline{a'},b}^{b',\overline{a}}) \circ (\gamma_{a',b'}^{b,a} \,.\, \mathrm{id}_{\overline{a}}) \circ (\mathrm{id}_b \,.\, \eta_a) = \eta_{a'} \,.\, \mathrm{id}_b \tag{4.10}$$

$$(\varepsilon_{a'} \,.\, \mathrm{id}_b) \circ (\mathrm{id}_{\overline{a'}} \,.\, \gamma_{a',b}^{b',a}) \circ (\gamma_{\cdot}^{b,\overline{a}}\, \mathrm{id}_a) = \mathrm{id}_b \,.\, \varepsilon_a \tag{4.11}$$

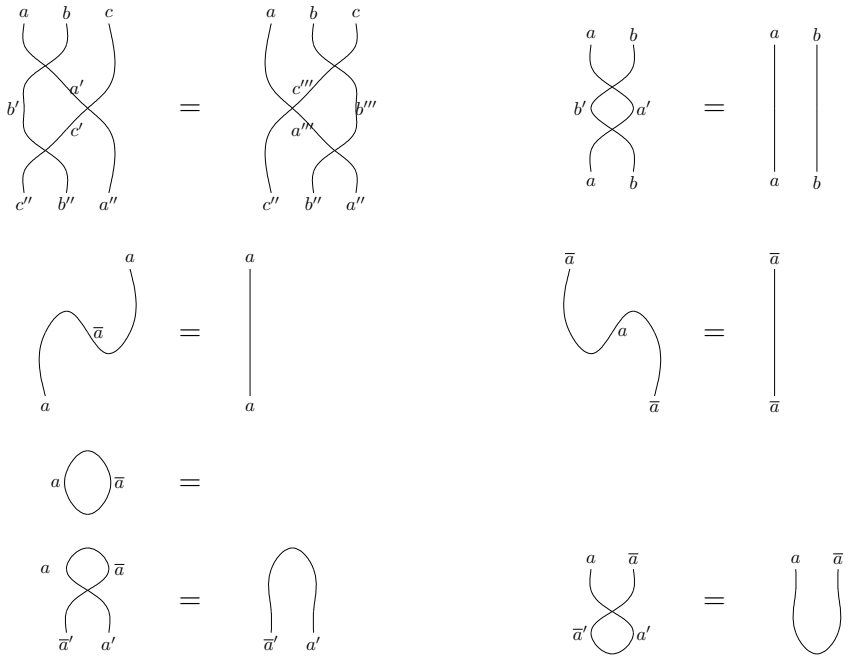
$$(\mathrm{id}_b \,.\, \varepsilon_{a'}) \circ (\gamma_{b,\overline{a'}}^{\overline{a},b'} \,.\, \mathrm{id}_{a'}) \circ (\mathrm{id}_{\overline{a}} \,.\, \gamma_{b',a'}^{a,b}) = \varepsilon_a \,.\, \mathrm{id}_b \tag{4.12}$$

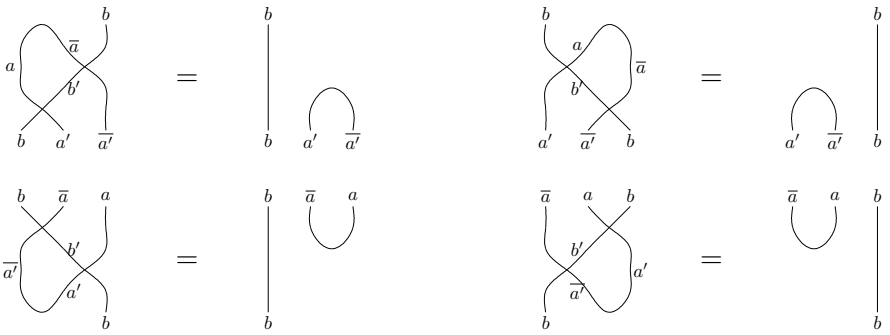
for possibly reversed transitions $a, a', a'', b, b', b'', c, c', c''$ such that all the involved morphisms are defined.

In string diagrammatic notation, the generators $\gamma_{b',a'}^{a,b}$, η_a and ε_a are drawn as



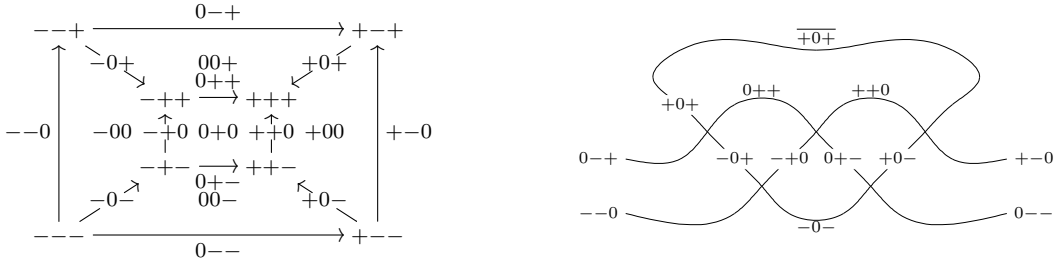
and the relations of Definition 116 can be pictured as





By definition of the generating 2-cells, we have that two paths $f, g : x \twoheadrightarrow y$ in C are homotopic if and only if there exists a 2-cell $\alpha : f \Rightarrow g$ in $\Pi_2(C)$.

Example 117. Consider the hollow cube without bottom C of Example 35, drawn on the left as the precubical set obtained from $\partial Y 3$ by removing the "bottom" face $0{-}0$. This precubical set is not NPC but we can still define a fundamental 2-category, except that some members of the equations may not be defined. The fact that the two paths $f = {--}0\,.\,0{-}+$ and $g = 0{--}\,.\,{+}{-}0$ are homotopic is witnessed by the following 2-cell $\phi : f \Rightarrow g$ in $\Pi_2(C)$ drawn on the right, from left to right instead of top to bottom for space constraints:



4.3.2 Rewriting homotopies as dihomotopies

By definition, the 2-cells of $\Pi_2(C)$ are equivalence classes of 2-cells in the free 2-category generated by the generating 2-cells $\gamma^{a,b}_{b',a'}$, η_a and ε_a, quotiented by the equivalence relation $\equiv$ generated by the relations. An element of such an equivalence class is called a *formal 2-cell*. We say that a formal 2-cell ϕ *rewrites* to a 2-cell ψ, what we write $\phi \Rrightarrow \psi$, when ψ can be obtained from ϕ by iteratively replacing the left member of a relation in some context by the right member of the relation in the same context, where the relation is one of the eleven relations of Definition 116 or three other derivable relations, see [MG16]. For instance the formal 2-cell on the left rewrites to the formal 2-cell on the right using the relation (4.3):

(4.13)

More formally, we can introduce a 3-category with the above free 2-category as underlying 2-category, and 3-cells generated by the relations oriented from left to right, and we would

have $\phi \Rrightarrow \psi$ precisely when there exists a 3-cell from ϕ to ψ in this 3-category. We refer the reader to Chapter 7 (Section 7.2) for the definition of polygraphs, which are a generalization of rewriting systems in which those results can be properly formulated. By suitably rewriting a formal 2-cell, one can always arrive at a formal 2-cell in "canonical form". Those are much easier to handle and will help us show our result.

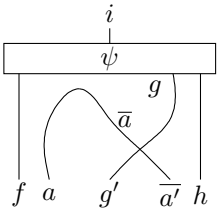
Definition 118. A formal 2-cell ϕ is a *canonical form* when it is of the form $\mathrm{id}_{\mathrm{id}_x}$ for some 0-cell x, which we write Z_x, or there exists a canonical form ψ such that ϕ is of one of the four following forms:

$$\begin{aligned} G^{a,f,g}_{f',a'}\psi &= (\gamma^{a,f}_{f',a'} \,.\, \mathrm{id}_g) \circ (\mathrm{id}_a \,.\, \psi) \\ H^{f,a,g,h}_{g',a'}\psi &= (\mathrm{id}_{f\,.\,a} \,.\, \gamma^{\overline{a},g}_{g',\overline{a'}} \,.\, \mathrm{id}_h) \circ (\mathrm{id}_f \,.\, \eta_a \,.\, \mathrm{id}_{g\,.\,h}) \circ \psi \\ E^{a,f}\psi &= (\varepsilon_a \,.\, \mathrm{id}_f) \circ (\mathrm{id}_{\overline{a}} \,.\, \psi) \end{aligned}$$

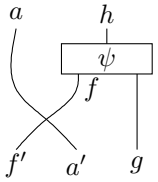
for some transitions a, a' and morphisms f, f', g, g', h. Graphically,

Z_x =

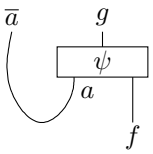
$H^{f,a,g,h}_{g',a'}\psi$ =



$G^{a,f,g}_{f',a'}\psi$ =



$E^{a,f}\psi$ =



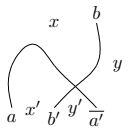
The operations G, H and E on 2-cells are called *operators*. Notice that the operator $E^{a,f}$ transforms a 2-cell $\psi : g \Rightarrow a \,.\, f$ into a 2-cell $E^{a,f}\psi : \overline{a} \,.\, g \Rightarrow f$. Similarly, the "type" for other operators is

$$\begin{array}{rcrcl} G^{a,f,g}_{f',a'} & : & h \Rightarrow f \,.\, g & \rightsquigarrow & a \,.\, h \Rightarrow f' \,.\, a' \,.\, g \\ H^{f,a,g,h}_{g',a'} & : & i \Rightarrow f \,.\, g \,.\, h & \rightsquigarrow & i \Rightarrow f \,.\, a \,.\, g' \,.\, \overline{a'} \,.\, h \\ E^{a,f} & : & g \Rightarrow a \,.\, f & \rightsquigarrow & \overline{a} \,.\, g \Rightarrow f \end{array}$$

Proposition 119 ([MG16])**.** *Every formal 2-cell ϕ rewrites to a canonical form.*

Example 120. The morphism (4.13) is not a canonical form in either of the two formal 2-cells. However, it can be rewritten to the following formal 2-cell (where we have also indicated the 0- and 1-cells):

$H^{\mathrm{id}_x,a,b,\mathrm{id}_y}_{b',a'} G^{a,\mathrm{id}_y,\mathrm{id}_y}_{\mathrm{id}_x,a} Z_y$ =



A formal 2-cell in canonical form is not, in general, a normal form. One can show that a canonical form of the form $XEHY$ (where X and Y are composites of operators and we omit indices) can always be rewritten to a canonical form of the form XGY or $XHEY$, and $XGHY$ to $XHGY$. By severely abusing notations, this can be summarized as

$$EH \quad \Rrightarrow \quad G \text{ or } HE \qquad\qquad GH \quad \Rrightarrow \quad HG \tag{4.14}$$

From these relations, it is easy to show that if a canonical form contains a H (i.e. is of the form XHY) then it can be rewritten to one which contains a H in first position (i.e. of the form HX) and if it contains an E (i.e. is of the form XEY) then it can be rewritten to one of the form XEY where Y does not contain an H. Since a morphism of the form HX always contains a reversed transition in its target, by contraposition if we know that the target does not contain reversed transitions (i.e. a is a dipath), we will be able to show that there is a canonical form without H in it, and similarly for E, from which we will be able to deduce that it is in fact a dihomotopy: in this way we are able to *rewrite any homotopy into a dihomotopy*!

Namely, suppose that two dipaths $f, g : x \to\!\!\!\!\to y$ in C are homotopic, i.e. there exists a 2-cell $\alpha : f \Rightarrow g$ in $\Pi_1(C)$. By Proposition 119, this 2-cell has as representative a formal 2-cell in canonical form. Suppose that this formal 2-cell is not a dihomotopy, i.e. that it is composed of a generator η_a or a generator ε_a. We suppose that it is composed of a generator η_a (the other case is essentially similar, although a bit more involved): this means that α can be expressed as a composite of operators, one of them being H (because H is the only operator containing η_a). By the above discussion, it is equal to a formal 2-cell of the form $\alpha = HX$, where X is a composite of operators. But this is not possible because of the definition of H: α would necessarily contain a reversed transition in its target, which is not the case because the target was supposed to be a dipath. In this way, one sees that α is equal to a formal 2-cell which is a composite of generators not involving η_a or ε_a, i.e. a dihomotopy, and we have:

Theorem 121 ([MG16])**.** *The canonical embedding $\vec{\Pi}_1(C) \to \Pi_1(C)$ is full and faithful, i.e. two dipaths in C are dihomotopic if and only if they are homotopic.*

4.3.3 Extensions

There are many possible extensions of this work, which are mentioned in [MG16].

The theory of compact closed categories. Our axiomatization of the fundamental category is an extension of the theory of compact closed categories. For instance, in the case where C is the terminal 2-dimensional precubical set, $\vec{\Pi}_2(C)$ is the free compact closed category containing an object which is *unidimensional* (this notion, which can be expressed in any compact closed category, characterizes vector spaces of dimension 1 in the category of finite vector spaces and linear maps [MG16]). These relationships with classical algebraic structures should be explored further.

Homological invariants. As mentioned in Section 3.1, finding a good notion of homology for directed spaces is not easy and still an active subject of research. However, in the case of NPC precubical sets homotopy and dihomotopy coincide and we therefore have some hope of being able to extract meaningful information about the directed space using traditional homology. This was in fact one of the original motivations for this work, but it turned out to be more difficult than expected and so is left for future work.

A convergent rewriting system. We have defined a notion of canonical form by hand, but of course it would be more satisfactory if canonical forms could be obtained as the normal forms for some convergent rewriting system, see Chapter 7. In fact, we conjecture that the rewriting system obtained by orienting relations from left to right is convergent. We believe that Guiraud's techniques based on derivations [Gui06] can be used in order to show the termination of the rewriting system. Confluence is quite tedious to check. Because of the "Yang-Baxter" rule (4.2) there is an infinite number of critical pairs as discovered by Lafont [Laf03]. We could still be able to handle those families of critical pairs (as done by Lafont), however this requires beforehand to establish the existence of canonical forms as we did in previous section: it would therefore require strictly more work than done here. As a byproduct of this result, we expect to be able to show that in $\Pi_2(C)$ (and therefore also in $\vec{\Pi}_2(C)$ by Theorem 121) there is at most one homotopy between any two paths (of course, when C satisfies the cube property and other hypothesis).

An axiomatization of homotopies between homotopies. We have seen in Section 3.1 (Theorem 75) that dihomotopy in a precubical set corresponds to dihomotopy in its directed geometric realization. We conjecture that this result extends one dimension higher, i.e. that two formal 2-cells in $\Pi_2(C)$ (resp. $\vec{\Pi}_2(C)$) are equal (i.e. in the same equivalence class modulo the relations) if and only if the corresponding homotopies (resp. dihomotopies) in C are homotopic, thus bringing a geometric justification for our axiomatic definition of $\Pi_1(C)$ and $\vec{\Pi}_2(C)$.

Fundamental n-categories. Finally, it would be interesting to study the higher-dimensional structure of fundamental categories, i.e. define and study the notion of a fundamental n-category for precubical sets.

Chapter 5

A geometric approach to asynchronous computability

In this chapter, we draw and study some links between geometric models presented here and work in the field of fault-tolerant distributed computing. There, people are concerned with designing algorithms and, when possible, solving so-called *decision tasks* on a given distributed architecture, in the presence of faults. The seminal result in this area was established by Fisher, Lynch and Paterson [FLP85], who proved the existence of a simple task that cannot be solved in a message-passing system (or in shared memory [LAA87]) with at most one potential crash. In particular, there is no way in such a distributed system to solve the very fundamental consensus problem: each processor starts with an initial value in local memory, typically a natural number, and should end up with a common value, which is one of the initial values. Later on, Biran, Moran and Zaks developed a characterization of the decision tasks that can be solved by a (simple) message-passing system in the presence of one failure [BMZ88]. The argument uses a "similarity chain", which can be seen as a connectedness result of a representation of the space of all reachable states, called the *protocol complex* [HS99] or the *view complex* [Koz12]. Of course, this argument turned out to be difficult to extend to models with more failures, as higher-connectedness properties of the protocol complex matter in these cases. This technical difficulty was first tackled, using homological considerations, by Herlihy and Shavit [HS93] (and independently [BG93, SZ93]): there are simple decision tasks, such as k-set agreement, a weaker form of consensus, that cannot be solved for $k < n$ in the wait-free asynchronous model, i.e. shared-memory distributed protocols on n processors, with up to $n - 1$ crash failures. Then, the full characterization of wait-free asynchronous decision tasks with atomic reads and writes (or equivalently, with atomic snapshots) was described by Herlihy and Shavit [HS99]: this relies on the central notion of chromatic (or colored) simplicial complexes, and their (chromatic) subdivisions. All these results stem from the contractibility of the "standard" chromatic subdivision, which was completely formalized only recently [Koz12, GMT14, Koz15] and corresponds to the *protocol complex* of distributed algorithms solving layered immediate snapshot protocols. A first contribution of ours was to show formally that the *iterated* subdivision is collapsible, and will be detailed in Section 5.3.

Over the years, the geometric approach to problems in fault-tolerant distributed computing has been very successful, see [HKR14] for a fairly complete up-to-date treatment. One potential limitation however is that for some intricate models, it is extremely difficult to produce their corresponding protocol complex. We believe that the geometric models for concurrency can shed some light on this, through the exploration of links between the geometric

semantics of the synchronization and communication primitives we are considering on a given distributed architecture, and the protocol complex. The interest is that the semantics of such synchronization primitives is much simpler to write down than the protocol complex, which is very error-prone to describe. We advocate that the calculation of protocol complexes can be performed directly from the formal semantics of the underlying synchronization primitives. First ideas in this direction were explored by Goubault [Gou96b, Gou96c, Gou97], although they were limited to simple particular cases.

In order to instantiate this link, we will be considering the simple model of shared-memory concurrent machines with crash failures, where processors compute and communicate through shared locations, and where reads and writes are supposed to be atomic. This model can also be presented as *atomic snapshot protocols* [AAD+93, And93, Lyn96], where processors are executing the following instructions: scanning the entire shared memory (and copying it into their local memory), computing in their local memory, and updating their "own value", i.e. writing the outcome of their computation in a specific location in global memory, assigned to them only.

We begin by recalling the computation model, expressing it as a particular case of the language used in this memoir (Section 5.1), then show how the protocol complex can be recovered from the corresponding geometric model (Section 5.2), and finally show that the iterated chromatic subdivision of the standard simplicial complex is collapsible (Section 5.3).

5.1 Asynchronous computability

This section is mostly based on [GMT15]. It mainly introduces and reformulates well-known definitions [HKR14].

5.1.1 Atomic snapshot protocols

Protocols. In this chapter, we consider only programs of the following specific form, following the *atomic snapshot shared memory* model. As explained above, this specific form has the advantage to allow for the description of the state space in a very convenient way (the protocol complex). A program consists of n processes p_i executed in parallel:

$$p \quad = \quad p_0 \parallel \ \ldots \ \parallel p_{n-1}$$

Each of the processes has a *local memory* cell l_i and a *global memory* cell m_i, containing values in a fixed set $\mathcal{V}$ of values. An element of $\mathcal{V}^n$ will thus be called a *memory* and a *state* is a pair $(l, m) \in \Sigma$ with $\Sigma = \mathcal{V}^n \times \mathcal{V}^n$. We suppose that the set of values is countable, e.g. we manipulate integers as in Chapter 1, so that we have an encoding $\langle x, y \rangle \in \mathcal{V}$ of pairs of values $x, y \in \mathcal{V}$, and more generally an encoding $\langle m \rangle \in \mathcal{V}$ of n-uples of values $m \in \mathcal{V}^n$. Each process p_i alternatively does two actions:

— *update*: it writes in its global memory cell a value depending on its local one,
— *scan*: it writes in its local memory cell a value depending on the previous value of its local memory cell as well as the contents of the global memory.

A process thus has the form

$$p_i \quad = \quad \texttt{while true do (} \ \underbrace{m_i \ \texttt{:=} \ f_i(l_i)}_{\texttt{update}} \ \texttt{;} \ \underbrace{l_i \ \texttt{:=} \ g_i(l_i, m)}_{\texttt{scan}} \ \texttt{)}$$

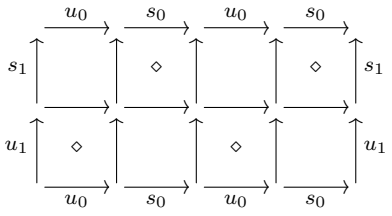
where $f_i : \mathcal{V} \to \mathcal{V}$ and $g_i : \mathcal{V} \times \mathcal{V}^n \to \mathcal{V}$ are arbitrary fixed functions, and a *protocol* π is entirely determined by the families of functions f_i and g_i of the various processes. A protocol is *full-information* when $f_i = \mathrm{id}_{\mathcal{V}}$ for every $i \in [n]$, i.e. every process fully discloses its local state in the global memory. The fact that, in the scan phase, the process reads the whole global memory is the reason why it is called *atomic snapshot*. It is well known [Lyn96] that those are equivalent, with respect to their expressiveness in terms of fault-tolerant decision tasks they can solve, to the protocols based on atomic registers with atomic reads and writes, where processes read one global memory cell at a time (some further admissible simplifications of protocols are mentioned in Section 5.1.3). Also, notice that there is no synchronization whatsoever between processes (for instance, a process cannot wait for another one to compute its value), which is why such a protocol is called *wait-free*.

Failures. We are interested here to what programs can compute in the presence of *failures*. We thus suppose that at any moment a process might *die* which means that it stops executing actions, and it stops modifying its local or global memory. The tasks to perform we are interested in will thus have to specify what the acceptable values are for any set of *alive* processes: we are only interested in the values computed by processes which did not die. In order to formally specify failures, we would need to modify the execution model presented in Chapter 1, hopefully in a way which is clear to the reader. Notice that because our model is asynchronous, if, after scanning, a process p_i does not see any change in the global memory cell of a process p_j, it cannot decide whether the process p_j has died or is simply slow performing its update, and this will be the cause of much of the trouble of designing correct protocols.

Execution traces. In order to simplify notations, we write u_i (resp. s_i) for the update (resp. scan) action of the i-th process, m_i`:=`$f_i(l_i)$ (resp. l_i`:=`$g_i(l_i, \langle m \rangle)$):

$$p_i \quad = \quad \texttt{while true do (}\ u_i\ \texttt{;}\ s_i\ \texttt{)}$$

Notice that as written above the program is not coherent (any two executions are equivalent since there is no use of synchronization primitives). In order to be so, one should at least use blocking sections in order to express the fact that the actions u_i and s_j are incompatible, since u_i writes on m_i and s_j reads m (and thus m_i), and we implicitly suppose that this is the case in the following. Thus, with two processes respectively executing only two and one round of the loop, the precubical semantics will be



(5.1)

We write $\mathcal{A}_i = \{u_i, s_i\}$ and $\mathcal{A} = \bigcup_{i \in [n]} \mathcal{A}_i$ for the set of actions. A word in $\mathcal{A}_i^*$ is *well-bracketed* if it belongs to $(u_i s_i)^*$ and by extension a word $u \in \mathcal{A}$ is well-bracketed if $\mathrm{proj}_i(u)$ is well-bracketed for every $i \in [n]$ where $\mathrm{proj}_i : \mathcal{A}^* \to \mathcal{A}_i^*$ is the obvious projection, keeping only the letters in $\mathcal{A}_i$. The *execution traces* are (in bijection with) well-bracketed words over $\mathcal{A}$ and two traces are equivalent if and only if they are related by the smallest congruence $\sim$ such that $u_j u_i \sim u_i u_j$ and $s_j s_i \sim s_i s_j$ for every $i, j \in [n]$ such that $i \neq j$. As explained in

Chapter 1, given a protocol, we have, for every well-bracketed word $u \in \mathcal{A}^*$, a *denotational semantics* $[\![u]\!] : \Sigma \to \Sigma$ which associates the state resulting from the execution of the trace u with every initial state (l, m), and given two traces u, v such that $u \sim v$, we have $[\![u]\!] = [\![v]\!]$. A state $(l', m') \in \Sigma$ is *reachable* when there there is an initial state $(l, m) \in \mathcal{I}^n \times \mathcal{V}^n$ (see below) and an execution trace u such that $[\![u]\!](l, m) = (l', m')$. Notice that the first thing a process does is to update the global memory, without reading it, so actually the semantics of an execution trace only depends on the local memory l: given an execution trace u, we have $[\![u]\!](l, m) = [\![u]\!](l, m')$ for all memories $l, m, m' \in \mathcal{V}^n$.

5.1.2 Tasks

From now on, we suppose given two distinguished sets of values $\mathcal{I}$ and $\mathcal{O}$, respectively called *input* and *output values*, the elements of $\mathcal{V} \setminus (\mathcal{I} \cup \mathcal{O})$ being called *intermediate values*. The elements of $\mathcal{I}^n \times \mathcal{V}^n \subseteq \Sigma$ are called *initial states*. We suppose that when a process p_i has computed an output value, by which we mean $l_i \in \mathcal{O}$, it will not change it anymore: for every $x \in \mathcal{O}$ and $m \in \mathcal{V}^n$, we suppose that $g_i(x, m) = x$. We also suppose fixed an element $\perp \in \mathcal{I} \cap \mathcal{O}$ standing for an *unknown value*: a process containing $\perp$ in its local memory can be considered as having died immediately after the beginning, or as no longer participating in the computation. Given a memory $l \in \mathcal{V}^n$, we write $\partial_i(l)$ for the memory obtained from l by replacing the i-th value by $\perp$, which is called a *face* of the memory: the notation and terminology are motivated by the fact that memories can be structured into simplicial complexes, as we will see in Section 5.1.5.

The problems we are interested in solving are called "tasks", and are specified by a relation stating given an initial local memory state, which are the acceptable local memory states after an execution of the protocol which is long enough so that every process has decided upon an output value. The case where a process dies at the beginning of the execution is modeled as its local state l_i being $\perp$, and we impose that the acceptable states in this case should be coherent with those in the case where the process did not die: if a process p finishes without hearing from process q, it does not necessarily mean that q has crashed, this latter process might just be slow, and therefore p has to react in the same way in both situations.

Definition 122 ([HKR14])**.** A *task* Θ is a relation $\Theta \subseteq \mathcal{I}^n \times \mathcal{O}^n$ such for every $(l, l') \in \Theta$ and $i \in [n]$,

1. $l_i = \perp$ if and only if $l'_i = \perp$,
2. there exists $l'' \in \mathcal{O}^n$ such that $(l, l'') \in \Theta$ and $(\partial_i(l), \partial_i(l'')) \in \Theta$.

A geometric intuition of the two conditions of the definition is given in Remark 130 below. The *domain* of a wait-free task Θ is $\operatorname{dom}\Theta = \{l \in \mathcal{I}^n \mid \exists l' \in \mathcal{O}^n, (l, l') \in \Theta\}$ and its *codomain* is $\operatorname{codom}\Theta = \{l' \in \mathcal{O}^n \mid \exists l \in \mathcal{I}^n, (l, l') \in \Theta\}$.

Example 123. In the *binary consensus* problem each process starts with a value in $\{0, 1\}$ and should end in the same set, thus $\mathcal{I} = \mathcal{O} = \{0, 1, \perp\}$, in such a way that in the end all the values chosen by the different processes are the same, and chosen among the initial values of the alive processes. For instance, with $n = 2$, the corresponding task is

$$\Theta \quad = \quad \{(b\perp, b\perp), (\perp b, \perp b), (bb', bb), (b'b, bb) \mid b, b' \in \{0, 1\}\}$$

In the case $n = 2$, we will also consider the variant called *binary quasi-consensus*, which restricts the output so that it cannot happen that p_1 decides 0 and p_0 decide 1 at the same time: the corresponding task is $\Theta \setminus \{(10, 10)\}$.

The fact that a protocol agrees with such a specification is formalized as follows. We write $\mathcal{A}^\omega$ for the set of infinite sequences of actions in $\mathcal{A}$. The notion of well-bracketed sequence can be extended to those in the expected way.

Definition 124 ([HKR14])**.** A protocol *solves* a task Θ when for every $l \in \operatorname{dom}\Theta$, and well-bracketed infinite sequence of actions $u \in \mathcal{A}^\omega$ which is *fair*, i.e. the projection on $\mathcal{A}_i$ is infinite for each $i \in [n]$, there exists a finite prefix u' of u such that $(l, l') \in \Theta$ where l' is the local memory after executing u', i.e. $(l', m') = [\![u']\!]_\pi (l, \bot^n)$.

In an execution, the number r_i of *rounds* for a process is the number of times the loop has been executed in an execution. If we suppose that there is only a finite number of possible initial values, by König's lemma, there is an upper bound on the number of rounds necessary for each process to reach an output value, and since executing a process longer does not change the decided output value, we can suppose that all the processes run exactly r rounds before deciding an output value.

5.1.3 Variants of protocols

In order to reason about protocols it is often convenient to restrict as much as possible the protocols we consider as well as their possible executions, without changing task solvability, in order to make the state-space as small and regular as possible, so that it is easier to reason about.

A first easy remark is that if a protocol π solves a task Θ then the protocol π' such that $f^{\pi'} = \mathrm{id}_{\mathcal{V}}$ and $g^{\pi'} = g^\pi \circ (\mathrm{id}_{\mathcal{V}} \times \prod_i f^\pi)$ also solves the task, and we can therefore restrict to *full-information* protocols. The other usually assumed restrictions are more subtle since they concern executions, and would require the modification of the execution model in order to be formally introduced. A protocol is *layered* if the loops of the various processes are synchronized: no process starts its $(k+1)$-th round before every process has ended its k-th round. A layered protocol is *clean memory* if the contents of the whole global memory are reset to $\bot^n$ at each round (sometimes layered protocols are implicitly supposed to be so, but not in this manuscript). A protocol is *immediate snapshot* when it is restricted to the executions such that after a scan has been performed, no update can be performed unless no other scan can be performed: the rounds of two processes are either concurrent or sequentially ordered, in the first case the immediate snapshot assumption amounts to suppose that all the reads are performed concurrently and then all the write are performed concurrently. For instance, with three processes, the execution $u_0u_1s_1s_0u_2s_2$ is immediate snapshot, but $u_0u_1s_0u_2s_1s_2$ is not: after the prefix $u_0u_1s_0$ the scan s_1 can be performed so that doing u_2 is not allowed. These assumptions are not restrictive: a task is solvable if and only if it is solvable by a clean-memory layered immediate snapshot full-information protocol [HS99].

In some of the following examples, we will assume that $\operatorname{dom}\Theta$ contains only the memory l such that $l_i = i$, and its faces, and call this a protocol with *standard input*. Moreover, the protocol complex (see Section 5.1.5) for an arbitrary task can be obtained by suitably renaming and amalgamating protocol complexes with standard input, and we will therefore study protocol complexes mostly in this case.

5.1.4 The view protocol

One can build a category of full-information protocols as follows.

Definition 125 ([GMT15])**.** Consider the category whose objects are protocols π operating on values $\mathcal{V}$ as objects, and a morphism $\phi : \pi \to \pi'$ witnesses the fact that π can simulate π': it consists of functions $\phi_i : \mathcal{V} \to \mathcal{V}$ and $\phi'_i : \mathcal{V} \to \mathcal{V}$ (respectively translating local and global memories), indexed by $i \in [n]$, such that ϕ_i is the identity when restricted to $\mathcal{I}$, and the following diagrams commute for $i \in [n]$:

$$\begin{array}{ccc} \mathcal{V} & \xrightarrow{f_i^\pi} & \mathcal{V} \\ {\scriptstyle \phi_i}\downarrow & & \downarrow{\scriptstyle \phi'_i} \\ \mathcal{V} & \xrightarrow[f_i^{\pi'}]{} & \mathcal{V} \end{array} \qquad \begin{array}{ccc} \mathcal{V} \times \mathcal{V}^n & \xrightarrow{g_i^\pi} & \mathcal{V} \\ {\scriptstyle \phi_i \times \prod_i \phi'_i}\downarrow & & \downarrow{\scriptstyle \phi_i} \\ \mathcal{V} \times \mathcal{V}^n & \xrightarrow[g_i^{\pi'}]{} & \mathcal{V} \end{array}$$

The category **Prot** of full-information protocols is the subcategory whose morphisms ϕ are such that $\phi'_i = \mathrm{id}_{\mathcal{V}}$ for every $i \in [n]$.

The definition of this category is not entirely satisfactory since it requires morphisms ϕ_i to be defined even on values that never occur in reachable memories. If we slightly modify the definition of morphisms and impose that ϕ_i should be suitable partial functions, it can be shown that the category **Prot** admits an initial object [GMT15]: the *view protocol* $\pi^\triangleleft$, also called the *generic protocol in normal form* [HS99], defined as follows. We have $f_i(x) = x$ for $x \in \mathcal{V}$ (i.e. the protocol is full-information) and $g_i(x, m) = \langle x, \langle m \rangle \rangle$ for $x \in \mathcal{V}$ and $m \in \mathcal{V}^n$: when reading the global memory, the protocol stores (an encoding as a value of) the pair constituted of its current local memory x and (an encoding as a value of) the global memory m it has read. A *view* for a process i is a value that can occur as the local memory l_i for some reachable state (l, m) and we write $\mathrm{Views}_i(\mathcal{I}^n)$ for the set of views that can be obtained starting from $\mathcal{I}^n$. The view protocol roughly exchanges its "full history" at each step, one can therefore expect that it is the "most general" one. This is shown in [HS99] and can be categorically reformulated as follows:

Proposition 126 ([GMT15])**.** *The view protocol is initial in the category of full-information protocols. In particular, given any protocol π, there exists a unique family of functions $\phi_i : \mathrm{Views}_i(\mathcal{I}^n) \to \mathcal{V}$ indexed by $i \in [n]$, such that $\phi_i(x) = x$ for $x \in \mathcal{I}$, and such that for every $\langle x, \langle m \rangle \rangle \in \mathrm{Views}_i(\mathcal{I}^n)$,*

$$\phi_i\left(\langle x, \langle m \rangle \rangle\right) \quad = \quad g_i^\pi\left(\phi_i(x), \left(f_j^\pi \circ \phi_j\left(m_j\right)\right)_{j \in [n]}\right)$$

Suppose that a task Θ is solvable by some protocol, in a given number r of rounds. Of course, the above results generalize to the case where we consider views which are reachable not from any local state in $\mathcal{I}^n$ but only those in $\mathrm{dom}\,\Theta$. By the initiality of the view protocol, this means that we can associate an output value with each view obtained after executing r rounds of each process, in such a way that after each execution the specification Θ is satisfied. Because of this, it is enough to study the structure of the views after r rounds, as we will do in the next section.

5.1.5 The protocol complex

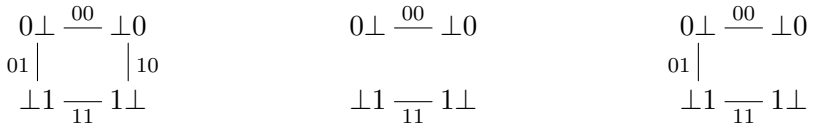
Since the set of possible inputs (resp. outputs) is closed under face operations, one can encode them as an abstract simplicial complex whose vertices are the possible inputs for the various processes and the simplices indicate when they are *coherent*, which means that they can occur in a same input.

Definition 127. An *(abstract) simplicial complex* $(\underline{K}, K)$ consists of a set $\underline{K}$ of vertices together with a set K of finite subsets of $\underline{K}$, called *simplices*, such that K is non-empty, contains the vertices as singletons, and is closed under taking subsets. A simplex σ is a *face* of a simplex τ when $\sigma \subseteq \tau$.

Definition 128. The *input complex* $K^-(\Theta)$ is the smallest abstract simplicial complex with $[n] \times (\mathcal{I} \setminus \{\bot\})$ as set of vertices, which contains, for any initial local memory $l \in \operatorname{dom} \Theta$ a simplex $\sigma = \{(i, x) \in [n] \times \mathcal{V} \mid l_i = x \neq \bot\}$. The *output complex* $K^+(\Theta)$ is defined similarly from $\operatorname{codom} \Theta$.

Notice that those complexes have another structure: their vertices are *colored* by process numbers and simplices contain only vertices of distinct colors. Constructions on these will be detailed in Section 5.3.1.

Example 129. In binary consensus and quasi-consensus tasks of Example 123, with $n = 2$, the input complexes are the same and shown on the left, and the output complexes are respectively shown in the middle and the right:



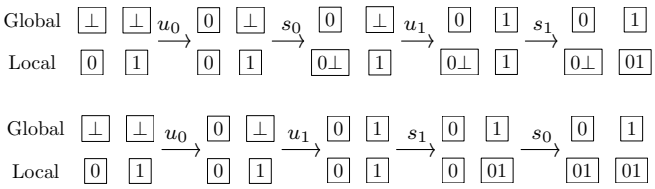
For the vertices, instead of writing (i, x), we write $\bot \ldots \bot x \bot \ldots \bot$ (with $x \neq \bot$ at the i-th position): for instance, we write $1\bot$ instead of $(0, 1)$, meaning that the process 0 starts with value 1. This has the advantage of generalizing to a notation for all simplices as illustrated above. The corresponding tasks are depicted in Example 133.

Remark 130. In Definition 122, the two conditions for defining a task Θ can be interpreted as follows. The first one imposes that Θ relates k-simplices of the input complex with k-simplices of the output complex. Given a simplex σ of the input complex, the simplices in relation by Θ with σ induce a subcomplex of the output complex. The second condition says that faces of σ are in relation with faces of this subcomplex.

The discussion in the previous section suggests that the structure of the possible views contains all the information about which decisions a protocol can make, since it is the universal one, and thus to introduce the following complex.

Definition 131. Given a number r of rounds, the *r-iterated (full-information) protocol complex* K^r is the smallest abstract simplicial complex whose vertices are pairs (i, l_i) where $i \in [n]$ and l is a local memory resulting from the execution of r rounds for each process in the view protocol, i.e. a view, such that for each such local memory l, there is a simplex $\{(0, l_0), \ldots, (n-1, l_{n-1})\}$. This construction of course extends to the case where the number of rounds is not the same for all processes, and is coded by $r \in \mathbb{N}^n$.

Example 132. The local views for each process are determined by the operational semantics, as in the two following examples with two processes, starting from the standard input:



There is a third potential outcome of the computation, symmetric to the first case, in which process 1 would do its update and scan before process 0 does. Putting this together, according to Definition 131, we get the well known protocol complex for one round and two processes [HS99]:

$$0,(0\bot) \text{ —— } 1,(01) \text{ —— } 0,(01) \text{ —— } 1,(\bot 1)$$

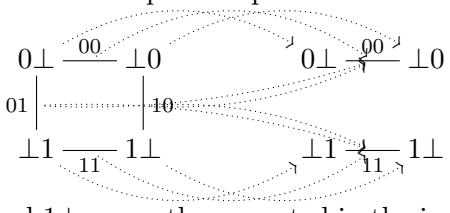
The encoding of the local states, i.e. vertices in the graph above, is as follows. The identifier of the process whose local view is the number before the comma, e.g. the state $0,(0\bot)$ above is the local view of process 0. The group of numbers or $\bot$ within parentheses, e.g. $(0\bot)$ in the state above, is a condensed notation for the local state where $l_0 = \langle 0, \langle 0, \bot\rangle\rangle$, see Section 5.1.1. Similarly, state $1,(01)$ denotes the local view of processor 1, with local state such that $l_1 = \langle 1, \langle 0, 1\rangle\rangle$.

The previous considerations are the starting point of the work of Herlihy [HS99], who (considerably) extended those. One can show that there is a protocol implementing a task if and only if there is a color-preserving simplicial map

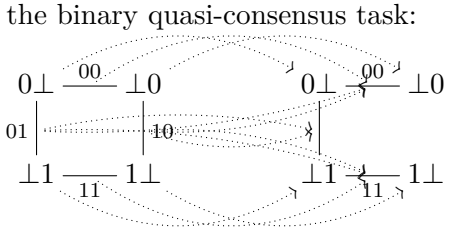
$$f \quad : \quad K^r \quad \to \quad K^+(\Theta) \tag{5.2}$$

from the r-iterated protocol complex for some number r of rounds to the output complex, which respects the task in the sense that the output associated to a given input should be an image in Θ (see the example below). This is actually sometimes even taken as the definition of a protocol. In the case of layered immediate snapshot protocols, this r-iterated protocol complex can be described as some form of subdivision of the input complex, called the *chromatic subdivision* (see Theorem 153), which is such that the subdivision $\chi(\Delta^n)$ of the standard n-simplex Δ^n is k-connected for every $k \in \mathbb{N}$. This implies that the (iterated) subdivision $\chi^r(K^-(\Theta))$ of the input complex preserves the connectivity properties of the input complex $K^-(\Theta)$. Now, it is a well-known fact that a simplicial map, such as (5.2), preserves the connectivity of the space, and one can thus obtain impossibility results roughly as follows: if the input complex is k-connected, while the output complex is not, we know that there can be no simplical map (5.2) and thus no protocol solving the task. Many other invariants have also been considered in order to show impossibility results [HKR14].

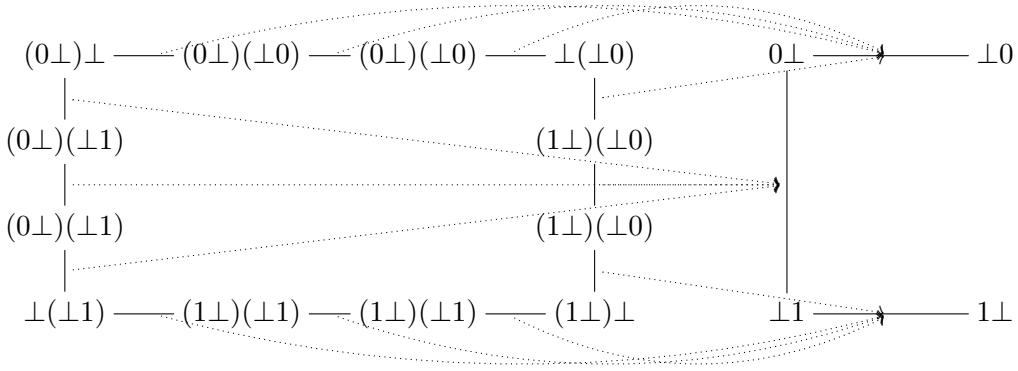
Example 133. The binary consensus task described in Example 123 can be represented by dotted arrows from the input to the output complex:



Notice that the vertices $\bot 0$ and $1\bot$ are path-connected in the input complex, and will remain so after subdivisions. Since their images (respectively $\bot 0$ and $1\bot$) are not connected in the output complex, there is no simplicial map from a subdivision of the input complex to the output complex which respects the task, and therefore no protocol implementing this task. The situation is different for the binary quasi-consensus task:



For reasons similar to before, there is no simplicial map which respects the task from the input complex to the output complex. However, there exists one after one subdivision of the input complex and therefore the task can be implemented by a protocol:



(for clarity, we have only shown the images of the simplicial map on edges).

5.2 A geometric construction of the protocol complex

This section is mostly based on [GMT15].

We have seen that the protocol complex is a central construction for studying asynchronous protocols. We show here how to reconstruct it from the geometric semantics. Our hope is that this should be of help for designing protocol complexes for other primitives where it is difficult to directly come up with the right notion of protocol complex, whereas the geometric semantics is already available. For this reason, we believe that it is interesting to understand the details of the correspondence instead of using too abstract reasoning, which might not be available for other models.

5.2.1 The geometric semantics

The geometric semantics of asynchronous read-write atomic snapshot protocols can be given as follows. Since we want to describe the state-space where the loops have been executed a given number of rounds, the resulting spaces are acyclic and the geometric semantics can either be given in terms of d-spaces, or in the simpler model of pospaces. We suppose given $r \in \mathbb{N}^n$ specifying the number of rounds for each of the n processes.

Definition 134 ([GMT15])**.** The geometric semantics X_r^n is the directed space

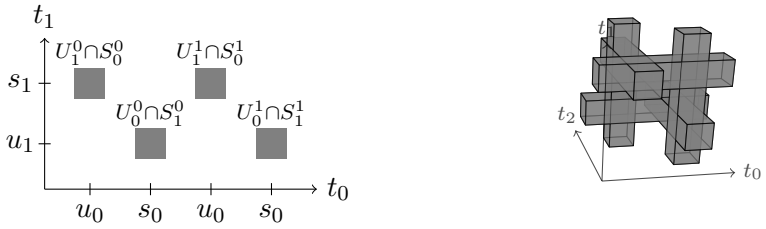
$$X_r^n \quad = \quad \prod_{i\in[n]} [0, r_i] \setminus \bigcup_{\substack{i,j\in[n] \\ k\in[r_i],\ l\in[r_j]}} U_i^k \cap S_j^l$$

endowed with the product topology and product order induced by $\mathbb{R}^n$, where $n, r_i \in \mathbb{N}$, $u, s \in \mathbb{R}$ with $0 < u < s < 1$, and

— $U_i^k = \left\{ x \in \prod_{i\in[n]}[0, r_i] \;\middle|\; x_i = k + u \right\}$ stands for the region where the i-th process updates the global memory into its local memory for the k-th time,

— $S_j^l = \left\{ x \in \prod_{i\in[n]}[0, r_i] \;\middle|\; x_j = l + s \right\}$ stands for the region where the i-th process scans the global memory with its local memory for the l-th time.

For instance, the spaces $X^2_{(2,1)}$ and $X^2_{(1,1,1)}$ are respectively



The geometric semantics defined here corresponds to the one introduced in Chapter 3. For instance, the space on the left is essentially the directed geometric realization of (5.1).

In [GMT15], we have shown how to explicitly associate a dipath $f_u : I \to X^n_r$ with every execution trace $u \in \mathcal{A}^*$ of r rounds and we have shown constructively that two equivalent dipaths are dihomotopic. Of course, the hard part is now to show the converse property, i.e. that given two execution traces $u, v \in \mathcal{A}^*$, a dihomotopy $f_u \sim f_v$ implies $u \sim v$. This is done by first showing that both execution traces up to equivalence and dipaths up to dihomotopy are in bijection with colored interval orders:

Definition 135. Let $(I_x)_{x\in X}$ be a family of intervals on the real line $(\mathbb{R}, \leqslant)$. This family induces a poset $(X, \preceq)$, where $\prec$ is defined by $x \prec y$ whenever $s < t$ for every $s \in I_x$ and $t \in I_y$, and such a poset is called an *interval order* [Fis70]. We denote by $x \| y$ the *independence* relation: $x \| y$ whenever $\neg(x \prec y)$ and $\neg(y \prec x)$.

An $[n]$-*colored* interval order consists of an interval order $(X, \preceq)$ and a *labeling* function $\ell : X \to [n]$ such that two elements with the same label are comparable.

The intuition behind our construction is that every pair of update and scan defines an "interval", colored by the process performing those. This can be formalized by explicitly constructing the maps of the following theorem.

Theorem 136 ([GMT15])**.** *One can construct maps as schematically depicted below:*

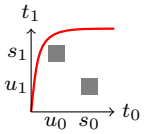
$$\begin{array}{ccccc} \textit{traces in } \mathcal{A}^* & & & & \textit{dihomotopy classes} \\ \textit{up to equivalence} & \xleftarrow{\quad \textit{bijection} \quad} & [n]\textit{-colored} & \underset{\textit{section}}{\overset{\textit{retraction}}{\rightleftarrows}} & \textit{of dipaths in } X^n_r \\ & & \textit{interval orders} & & \end{array}$$

the pair on the right being a deformation retract.

Proof. We only briefly describe the first map. We associate a labeled interval order $(X, \preceq)$ with any execution trace $u \in \mathcal{A}^*$ as follows. We denote by r_i the number of occurrences of u_i in u, and by u_i^k and s_i^k the respective k-th occurrence of u_i and s_i. The set X is then defined as $X = \{(i,k) \mid k \in [r_i],\ i \in [n]\}$. Any embedding of u in the real line induces an interval order by setting $I_{(i,k)} = [u_i^k, s_i^k]$. More precisely, X is then endowed with the partial order such that $(i,k) \prec (i',k')$ whenever $s_i^k < u_{i'}^{k'}$, i.e. s_i^k occurs before $u_{i'}^{k'}$. We can label this interval order $(X, \preceq)$ by the projection $\ell : (i,k) \mapsto i$, and hence produce an $[n]$-colored interval order since u is supposed to be well-bracketed. The inverse of this map can be obtained by suitably "linearizing" a partial order. □

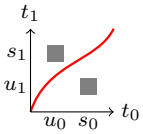
As an illustration, in $X^2_{(1,1)}$, the following dipaths, which represent the three possible

dihomotopy classes, correspond to the following interval orders and execution traces:



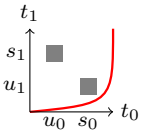
$[u_1, s_1] \prec [u_0, s_0]$

$u_1 s_1 u_0 s_0$



$[u_0, s_0] \parallel [u_1, s_1]$

$u_0 u_1 s_0 s_1$



$[u_1, s_1] \prec [u_0, s_0]$

$u_0 s_0 u_1 s_1$

5.2.2 Recovering the protocol complex

The protocol complex captures the views of the various processes and their coherence, i.e. when they can occur simultaneously in an execution. In particular, the maximal simplices correspond to the views of all the processes and we can therefore expect that they correspond to executions up to equivalence. This provides us with a way to generate the protocol complex: we start from the dihomotopy classes of dipaths, we consider them as maximal simplices, and close them under faces. By Theorem 136, it is equivalent to consider colored interval orders instead of dihomotopy classes, which turns out to be simpler technically.

First, the view can be extracted from a labeled interval order as follows. The definition is technical so we omit it here, see [GMT16], and write V_i^k for the interval order corresponding to the view of the k-th scan of the i-th process. We are now in a position to give a new equivalent combinatorial description of the protocol complex, using interval orders:

Definition 137. The *interval order complex* is the simplicial complex whose
- vertices are $((i,k), V_i^k)$ where i stands for the i-th process, k for the round number,
- maximal simplices are $\{((0,r_0), V_0^{r_0}), \dots, ((n,r_n), V_n^{r_n})\}$ such that there is an interval order as in the previous proposition whose restriction to (i, r_i) is $V_i^{r_i}$.

In that case we say that it is the interval order complex on r rounds and for $n+1$ processes.

Example 138. Consider again the one round, two process case of Example 132. We have represented below the protocol complex, and decorated its maximal simplices, i.e. edges, with the corresponding dipaths modulo dihomotopy above, and the corresponding interval order below:

$$0,(0\bot) \;\; \underset{0\prec 1}{\square} \;\; 1,(01) \;\; \underset{0 \;\; 1}{\square} \;\; 0,(01) \;\; \underset{0\succ 1}{\square} \;\; 1,(\bot 1)$$

The local view of process 0 which is $0,(0\bot)$ comes from the restriction to 0 of the interval order $_{0\prec 1}$, subscript of the leftmost edge in the graph above: an interleaving trace corresponding to this interval order is $u_0 s_0$ leading to local state $(0\bot)$ on process 0. Similarly, $1,(01)$ corresponds to the local state $l_1 = (01)$ for process 1, both for the restriction $_{0\prec 1}$ of $_{0\prec 1}$ to $\mathcal{V}_1^1$ (corresponding to a trace $u_0 s_0 u_1 s_1$, as in the trace $\square$ superscript of the edge on the left of the graph above) and for the restriction $_{0 \;\; 1}$ of $_{0 \;\; 1}$ to V_1^1 (corresponding to a trace $u_0 u_1 s_0 s_1$ for instance, as in the trace $\square$ superscript of the middle edge of the graph above).

Example 139. An example of interval order complex with traces corresponding to the execution for 2 processes, 2 rounds is depicted at Figure 5.1. Note that this is not the classical iterated chromatic subdivision in three parts at each round [HKR14], i.e. a 9 edge complex, because the protocols we are considering are not supposed to be layered. In Figure 5.2, we show the interval order complex for 3 processes and 1 round. Note again that we do not have

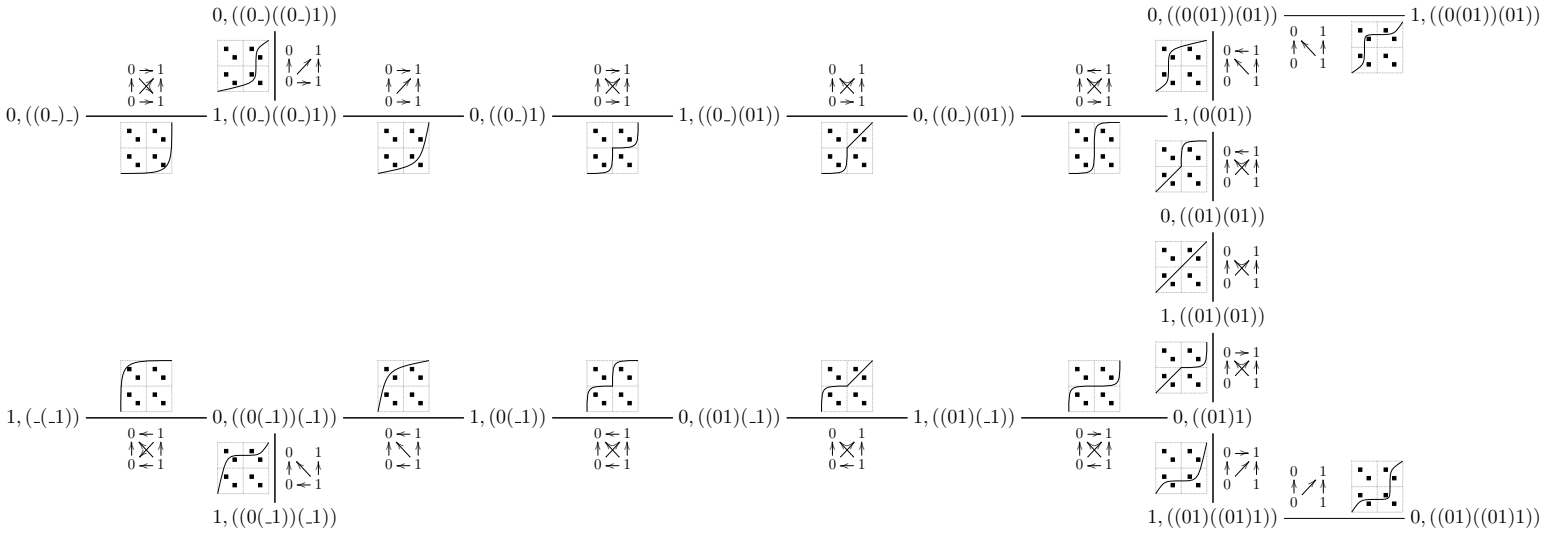


Figure 5.1 – Interval order complex, together with corresponding traces, of 2 processes, 2 rounds.

exactly the same picture as in [HKR14]: to the 13 triangles in [HKR14], we have to add the 6 extra blue triangles that make the complex not faithfully representable as a planar shape, which correspond to non-immediate snapshot executions. For instance, the upper left blue triangle is labeled with the interval order where 0 is not comparable to both 1 and 2, and 2 is less than 1. An interleaving trace (up to equivalence) corresponding to this interval order is given on the same figure: $u_0u_2s_2u_1s_1s_0$.

In order to compare our construction with the traditional one, we need to restrict to interval orders corresponding to layered immediate snapshot executions. The following theorem shows that once we have done this, we recover the usual notion of protocol complex, which corresponds to a chromatic subdivision by Theorem 153 recalled in the next section.

Theorem 140 ([GMT15])**.** *Layered immediate snapshot executions correspond to interval orders such that $J \prec K$ and I is not comparable with J implies $I \prec K$. The subcomplex of the interval order complex on one round, restricted to immediate snapshot executions, is isomorphic to the chromatic barycentric subdivision of the input complex.*

5.3 Collapsibility of the protocol complex

This section is mostly based on [GMT14].

As already mentioned in the previous section, the protocol complex for layered immediate snapshot executions corresponds to a particular subdivision of the input complex, which is a colored simplicial complex called the *chromatic subdivision*. Moreover, one of the main properties of this subdivision is that it preserves the connectivity of the complex. This is shown in [HS99] (Corollary 4.13), in a non-constructive way, by considering the various possible executions generating the protocol complex. Here, we considerably rework the definition of the chromatic subdivision, expressing it in more abstract terms, which allows us to constructively

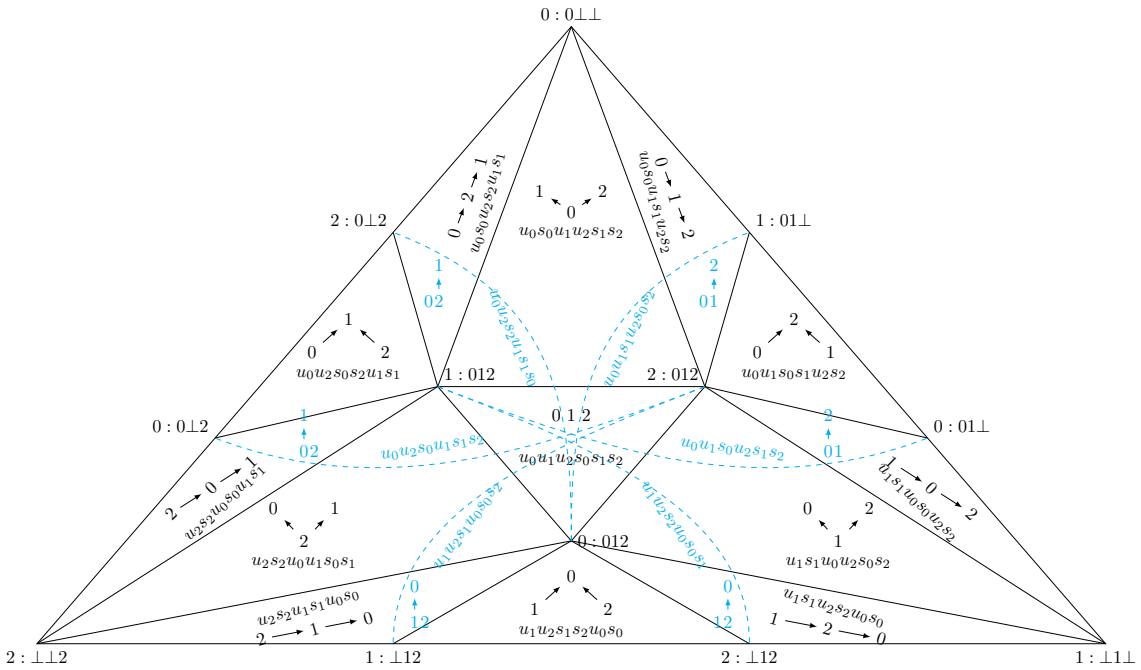


Figure 5.2 – Interval order complex with traces of 3 processes, 1 round.

show that it is contractible when applied to standard simplices, and thus preserves connectivity. Moreover, our proof is of purely combinatorial nature, not referencing the execution model of protocols. The same result was proved independently by Kozlov [Koz15], however our proof has the advantage of being easily extended to the case of iterated subdivisions.

5.3.1 Labeled simplicial complexes

The notion of (abstract) simplicial complex was already recalled in Definition 127.

Definition 141. We write **SC** for the category of simplicial complexes with morphisms $f : K \to K'$ the functions $f : \underline{K} \to \underline{K'}$ between sets of vertices preserving simplices and locally injective: for every simplex $\sigma \in K$, the restriction of f to σ is injective.

Instead of working with simplicial complexes, all the developments performed in this section could have been done in a suitable presheaf category, as explained in Section 5.3.5. However, simplicial complexes are easier to manipulate, and give rise to much simpler notations, which is why we use them here. The standard simplices are of course of particular interest in the category of simplicial complexes, since they constitute the analogue of representable objects. Given a finite set I, we write Δ^I for the *standard I-simplicial complex*, with I as vertices and subsets of I as simplices. We often write Δ^n instead of $\Delta^{[n]}$. Given a simplicial complex K and a simplex $\sigma \in K$, we write $K \setminus \sigma$ for the largest subcomplex of K not containing σ. This allows us to also define *horns* Λ^I_p, with $p \in I$, as $\Lambda^I_p = \Delta^I \setminus \sigma$ where σ is the simplex $I \setminus \{p\}$. In fact, there is a small indexing problem, since Δ^0 should in fact be considered as a (-1)-simplex (we are considering augmented simplicial sets). In order for the pictures below to make sense, we implicitly suppose that I is non empty and

shift dimensions by one (in case of doubt, the article [GMT14], on which this section is based, does have correct indices).

Writing **Inj** for the category of finite sets and injections, we have a functor $! : \mathbf{Inj} \to \mathbf{SC}$ sending a finite set I to Δ^I and a function to itself. This allows us to define labeled simplicial complexes by a slice construction. Since we will only need to take labels in $\mathbb{N}$, we define

Definition 142. We write $\mathbf{CSC} = \mathbf{SC}/!$ for the category of *colored simplicial complexes*.

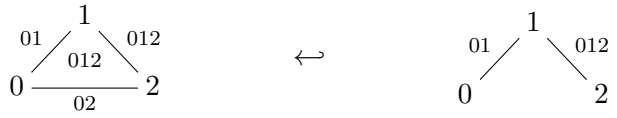
An object in this category consists of a simplicial complex K together with a morphism $\ell : K \to !\,\mathbb{N}$. Because morphisms are locally injective, this amounts to a function $\ell : \underline{K} \to \mathbb{N}$ such that any two vertices $x, y \in \sigma$ in a simplex σ have distinct colors $\ell(x) \neq \ell(y)$. In particular, any standard simplicial complex Δ^n is canonically colored by the inclusion $\ell : [n] \to \mathbb{N}$.

5.3.2 Collapsibility

The main tool we are going to use in order to show that a space is contractible, and thus n-connected for every $n \geqslant 0$, is the notion of collapse introduced by Whitehead [Whi39]. Collapsibility is a stronger property than contractibility since there are contractible spaces which are not collapsible (the Bing house with two rooms is an example of this): collapses constitute a tractable way of "reducing" a complex without changing its homotopy type, but those do not form a complete set of operations to generate homotopy equivalence (in fact, in the more general setting of finite CW-complexes, the Whitehead torsion measures the obstruction of a homotopy equivalence being generated by collapses or their inverses [Whi50]). These are analogue in the setting of simplicial complexes of Tietze-equivalence which will be discussed in Chapters 7 and 8.

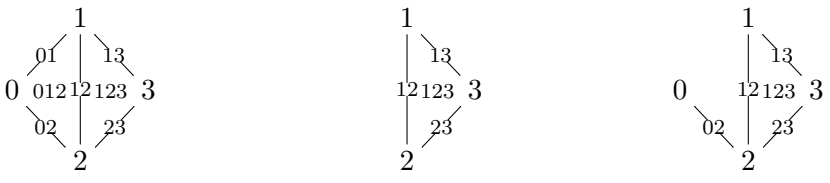
Definition 143. A simplex τ is a *free face* of a simplex σ in a simplicial complex K when $\tau \subsetneq \sigma$ (τ is a strict face of σ), σ is a maximal simplex and the only maximal simplex having τ as face. In this situation, the inclusion $(K \setminus \tau) \hookrightarrow K$ is called a *collapse step*. A collapse step is *elementary* when $\dim \tau = \dim \sigma - 1$. A *collapse* is a morphism composed of collapse steps and a complex is *collapsible* when there exists a collapse $\Delta^0 \to K$, which "reduces K to a point".

The prototypical example of a collapse is to replace a simplex Δ^I by a horn Λ^I_p. For instance, we can replace Δ^3 by Λ^3_1 without changing the homotopy type:



(the corresponding inclusion is from the right to the left).

Example 144. Consider the following simplicial complex, whose set of vertices is $\{0, 1, 2, 3\}$ and whose maximal simplices are $\{0, 1, 2\}$ and $\{1, 2, 3\}$, depicted on the left. The simplices $\{0\}$ and $\{0, 1\}$ are free faces of $\{0, 1, 2\}$, respectively giving rise to the following collapses, in the middle and the right. On the contrary, the simplices $\{1\}$ and $\{1, 2\}$ are not free faces.



It is well-known that restricting to elementary collapses is not restrictive [Whi39, GMT14]:

Lemma 145. *Every collapse can be obtained as a composition of elementary collapse steps.*

5.3.3 The chromatic subdivision

In this section, we adapt the usual definition of the barycentric subdivision to the colored case. The abstract definition we based ours on is folklore, see for instance [GJ09]. We first begin by introducing a category of graphs (distinct from the usual presheaf definition such as the one of Section 2.1) that will be used in the following.

Definition 146. We write **Graph** for the category of *graphs* $G = (V_G, E_G)$, with $E_G \subseteq V_G \times V_G$, which are irreflexive: $(x, y) \in E_G$ implies $x \neq y$. A morphism $f : G \to H$ consists of a function $f : V_G \to V_H$ such that for every $(x, y) \in E_G$, we have $(f(x), f(y)) \in E_H$.

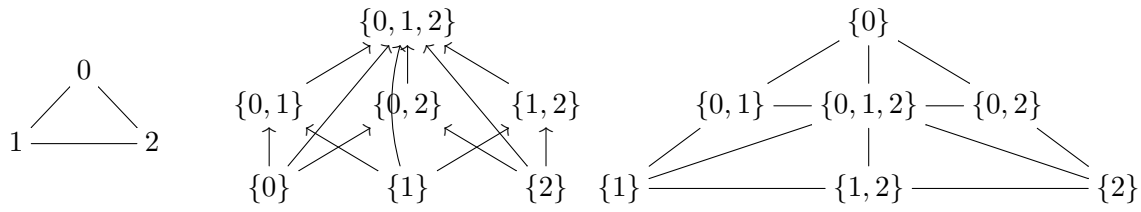
A useful construction when studying a simplicial complex is its *poset of faces* which is the poset of non-empty simplices of the complex ordered by inclusion. In Definition 147, we introduce a variant of this definition, which we call the *graph of elements* of the complex (by analogy with the category of elements of a presheaf such as a presimplicial set). We introduce this variant because its generalization to colored simplicial complexes (see Definition 151) naturally gives rise to graphs with cycles (see Example 152) which do not correspond to posets anymore: this also explains the peculiar definition of graphs we consider in Definition 146, which can be seen as a generalization of the notion of poset allowing some cycles.

Definition 147. The *graph of elements* $\mathrm{El}(K)$ of a simplicial complex K is the graph whose elements are the non-empty simplices of K and there is an edge $\tau \to \sigma$ whenever $\tau \subsetneq \sigma$. This construction extends to a functor $\mathrm{El} : \mathbf{SC} \to \mathbf{Graph}$.

Definition 148. The *nerve of a graph* $G = (V_G, E_G)$ is the simplicial complex NG whose vertices are those of G and simplices are sets of the form $\{x_1, \ldots, x_n\}$ with an edge $x_i \to x_j$ for any $i < j$. This construction extends to a functor $N : \mathbf{Graph} \to \mathbf{SC}$.

Definition 149. The *barycentric subdivision* functor is $\chi = N \circ \mathrm{El}$.

Example 150. Consider Δ^2 (on the left), the associated graph $\mathrm{El}(\Delta^2)$ is pictured in the middle and $\chi(\Delta^2) = N(\mathrm{El}(\Delta^2))$ on the right:



The previous definitions can be adapted to the colored case as follows, providing an abstract reformulation of the usual definition of the standard chromatic subdivision [HS99]. We define a functor $! : \mathbf{Inj} \to \mathbf{Graph}$ which associates, with a set X, the graph with X as set of vertices and pairs $(x, y) \in X \times X$ with $x \neq y$ as edges. The category of *colored graphs* is the slice category $\mathbf{Graph}/!\mathbb{N}$.

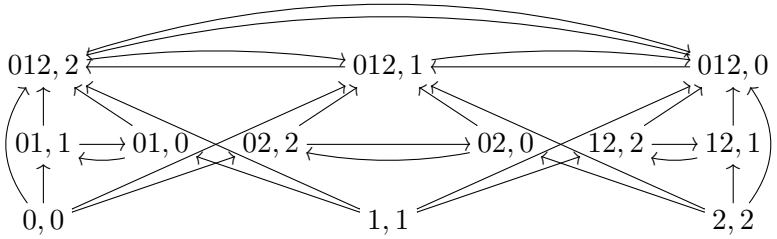
Definition 151. We define a functor $\mathrm{El} : \mathbf{SC}/!\mathbb{N} \to \mathbf{Graph}/!\mathbb{N}$ which, with every simplicial complex K, associates the graph whose vertices are pairs (σ, i) where $\sigma \in K$ and $i \in \ell_K(\sigma)$, labeled by i, and there is an edge $(\tau, i) \to (\sigma, j)$ whenever

1. $i \neq j$,
2. $\tau \subseteq \sigma$,
3. $j \notin \ell_K(\tau)$ or $\sigma \subseteq \tau$.

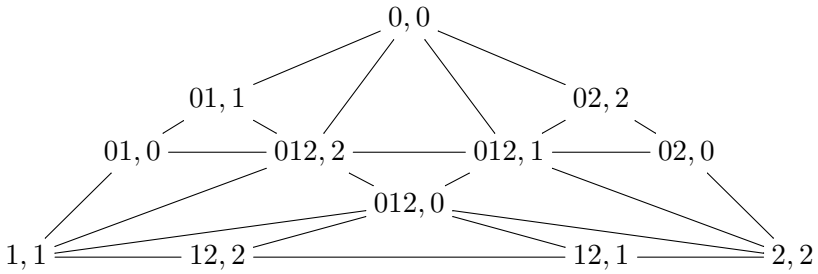
In the other direction, we define a functor $N : \mathbf{Graph}/!\mathbb{N} \to \mathbf{SC}/!\mathbb{N}$ which, with a colored graph (G, ℓ), associates the simplicial complex with V_G as set of vertices, labeled by ℓ, simplices being sets of the form $\{x_1, \ldots, x_n\}$ such that there is an edge $x_i \to x_j$ whenever $i < j$. The *standard chromatic subdivision* functor is

$$\chi \quad = \quad N \circ \mathrm{El} \quad : \quad \mathbf{CSC} \quad \to \quad \mathbf{CSC}$$

Example 152. Consider the labeled complex Δ^2. The colored graph $\mathrm{El}(\Delta^2)$ is:



(the second component of vertices indicate the color) and $\chi(\Delta^2) = N(\mathrm{El}(\Delta^2))$ is



(all the triangles are filled).

We can now formally state the reason of our interest in this construction:

Theorem 153 ([HS99])**.** *The full-information protocol complex of layered immediate snapshot protocols with r rounds is the chromatic subdivision of the input complex iterated r times.*

This can be understood as follows in the case where the input is standard, i.e. the input complex is the standard simplicial complex Δ^n. By applying Definition 151, the simplicial complex $\chi(\Delta^n)$ can be explicitly described as follows. Its vertices are pairs (V, i) with $V \subseteq [n]$ and $i \in V$ and simplices are the sets $\sigma = \{(V_0, i_0), \ldots, (V_d, i_d)\}$ which are, for every $p, q \in [d+1]$,

1. *well-colored*: $i_p = i_q$ implies $p = q$,
2. *ordered*: $V_p \subseteq V_q$ or $V_q \subseteq V_p$
3. *transitive*: $i_p \in V_q$ implies $V_p \subseteq V_q$

(these three points correspond precisely to the three points of Definition 151). Here, $i \in V_j$ means that the process j sees what the update of the i-th process has written, i.e. this corresponds to an execution where s_j occurs after u_i, and the conditions can be interpreted as:

1. we only consider coherence of views of distinct processes,
2. in an execution, given two processes, the scan of the first has to occur after the update of the other, or the contrary, or both (it cannot happen that none of the two processes sees what the other has written),
3. if a process sees another, then the latter cannot see more than the former.

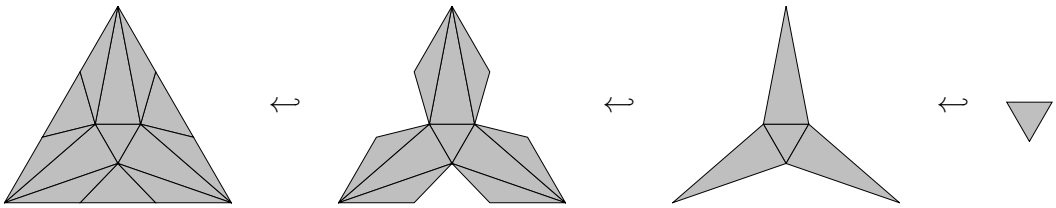
The last condition is maybe the most intriguing. For instance, the execution trace $u_0u_1s_0u_2s_1s_2$ violates this condition since, after its execution, the views V_i of processes i are respectively $V_0 = \{0, 1\}$, $V_1 = \{0, 1, 2\}$ and $V_2 = \{0, 1, 2\}$: we have $1 \in V_0$ but not $V_1 \subseteq V_0$. The reason for this is that the execution trace is not immediate snapshot. It can actually be shown that if we remove the third condition, we obtain precisely the full-information protocol complex for layered (non-immediate snapshot) protocols.

5.3.4 The standard chromatic subdivision of the standard colored simplicial complex is collapsible

One of the main result of ours in this section is the following one:

Theorem 154 ([GMT14])**.** *The standard chromatic subdivision $\chi(\Delta^n)$ of the standard n-simplex is collapsible.*

The proof is quite subtle and technical, and requires one to show many intermediate results about the interaction between collapse and elementary geometric operations (taking the star of a simplex, the (colored) join of two simplicial complexes, etc.). It is performed by constructing an explicit sequence of collapse steps, which can be illustrated on $\chi(\Delta^2)$, depicted in Example 152. Notice that there is an inclusion $\Delta^2 \hookrightarrow \chi(\Delta^2)$ sending the standard 2-simplicial complex to the central 2-simplex of $\chi(\Delta^2)$, whose vertices are those of the form $(012, i)$ with $i \in [2]$. Our goal is to show that this inclusion is a collapse, which will be enough to conclude since Δ^2 can easily be shown to be collapsible. This can be done by the following sequence of collapse steps:



5.3.5 Chromatic presimplicial sets

In order to be able to use the machinery presented in Section 2.1, we briefly explain how colored simplicial complexes can be embedded into a presheaf category.

Definition 155 ([GMT14])**.** The *chromatic presimplicial category* $\triangle_!$ has objects non-empty subsets I of $\mathbb{N}$ and morphisms are inclusions. Presheaves over this category are called *chromatic presimplicial sets.*

Remark 156. Notice that this category is really "close" to the category $\triangle$ of presimplicial sets. Given $I \subseteq \mathbb{N}$ there is a unique increasing bijection $\iota_I : I \to [\#I]$, where $\#I$ is the cardinal of I, and we can define a functor $\triangle_! \to \triangle$ sending I to $\#I$ and a morphism $f : I \to J$ to $\iota_J \circ f \circ \iota_I^{-1}$. This enables us for instance to easily define the geometric realization of a chromatic presimplicial set as the realization along the composite functor $\triangle_! \to \triangle \to \mathbf{Top}$.

We define a functor $U : \mathbf{CSC} \to \hat{\triangle}_!$ which, with a simplicial complex $K \in \mathbf{CSC}$, associates the chromatic presimplicial set UK such that the image of an object I and a morphism $J \subseteq I$ are respectively

$$UK(I) = \{\sigma \in K \mid \ell(\sigma) = I\} \qquad \begin{array}{rcl} UK(J \subseteq I) : UK(I) & \to & UK(J) \\ \sigma & \mapsto & \{x \in \sigma \mid \ell(x) \in J\} \end{array}$$

Conversely, we define a functor $F : \hat{\triangle}_! \to \mathbf{CSC}$ which, with a presimplicial set $P \in \hat{\triangle}_!$, associates the simplicial complex with $\underline{FP} = \coprod_{i \in \mathbb{N}} P(\{i\})$, a vertex $x \in P(\{i\})$ being labeled by i, and set of simplices

$$FP \quad = \quad \{P\left(\{i\} \subseteq I\right)(\sigma) \mid \sigma \in P(I) \text{ and } i \in I\}$$

Proposition 157. *The functor F is left adjoint to U and the induced comonad $F \circ U$ is the identity. The functor U is thus an embedding.*

A definition of a collapse (step) can easily be formulated in this setting by mimicking the one for simplicial complexes (Definition 143), and one can show that in this category,

Lemma 158 ([GMT14])**.** *Collapses are stable under pushouts with coinitial morphisms. In particular, elementary collapse steps are generated by pushouts of inclusions $\Lambda^I_p \to \Delta^I$ of horns into standard simplicial complexes along coinitial morphisms.*

5.3.6 Contractibility of the iterated chromatic subdivision

We can now prove that the iterated chromatic subdivision $\chi^r(\Delta^n)$ of the standard simplicial complex is contractible, as follows. First, one can notice that the functor $\chi : \hat{\triangle}_! \to \hat{\triangle}_!$ is entirely determined by the images of the representable objects, i.e. the standard simplices.

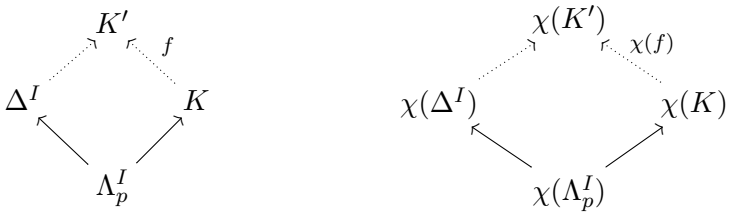
Proposition 159 ([GMT14])**.** *We write $\chi^\Delta : \triangle_! \to \hat{\triangle}_!$ for the functor obtained as the "restriction of χ to representables", i.e. $\chi^\Delta = \chi \circ Y$ where $Y : \triangle_! \to \hat{\triangle}_!$ is the Yoneda embedding: this functor associates $\chi(\Delta^I)$ with I and the obvious embedding $\chi(\Delta^J) \to \chi(\Delta^I)$ with an inclusion $J \subseteq I$. Then the cocontinuous extension of χ^Δ as a functor $\hat{\triangle}_! \to \hat{\triangle}_!$ is the functor χ.*

Using the previous proposition, it can be shown that

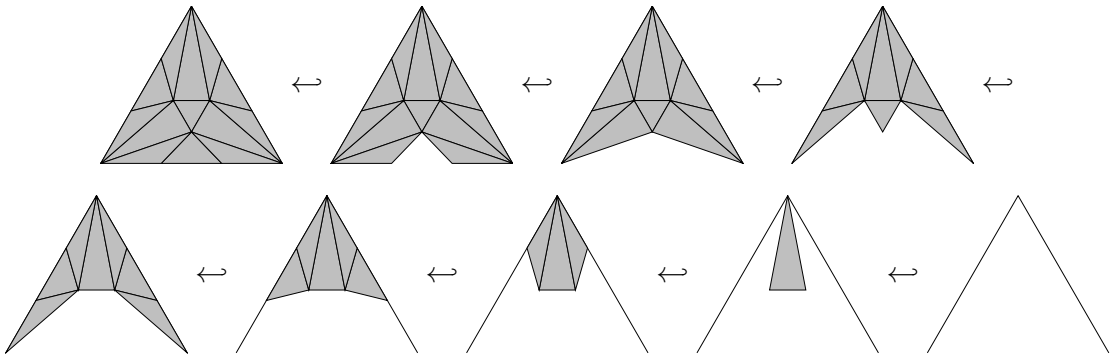
Proposition 160 ([GMT14])**.** *The functor χ preserves collapses.*

Proof. By Lemma 145, it is enough to show that χ preserves elementary collapse steps f, which can be obtained as pushouts along inclusions $\Lambda^I_p \hookrightarrow \Delta^I$ by Lemma 158, as the morphism f on the pushout diagram on the left below. Since the functor χ is a realization along χ^Δ, it

is left adjoint to the nerve associated with χ^Δ and thus preserves colimits. In particular, the pushout diagram on the left is sent to the pushout diagram on the right:



If we can show that the inclusion $\chi(\Lambda^I_p) \hookrightarrow \chi(\Delta^I)$ is collapse, we will be able to conclude by Lemma 145: $\chi(f)$ will be a collapse as a pushout of a collapse. This is indeed the case and can be shown as a variant of the proof of Theorem 154. We only illustrate the fact that $\chi(\Lambda^2_p) \hookrightarrow \Delta^2$ is a collapse by decomposing it as a sequence of (elementary) collapse steps:



□

From this, we can almost immediately deduce:

Theorem 161 ([GMT14])**.** *Given a collapsible colored simplicial complex K, $\chi(K)$ is also collapsible. In particular, iterated chromatic subdivisions of the standard simplex $\chi^r(\Delta^n)$ are collapsible.*

In particular, the full-information protocol complex of layered immediate-snapshot protocols with standard input over r rounds is $\chi^r(K)$, and is thus collapsible and k-connected for every $k \in \mathbb{N}$. In [GMT14], we claimed that our methods should be general enough to show a similar result in the non-layered case and more. Indeed, Benavides and Rajsbaum have extended the above methods to show that the iterated subdivision of the standard simplicial complex corresponding to the read-write model is collapsible [BR15].

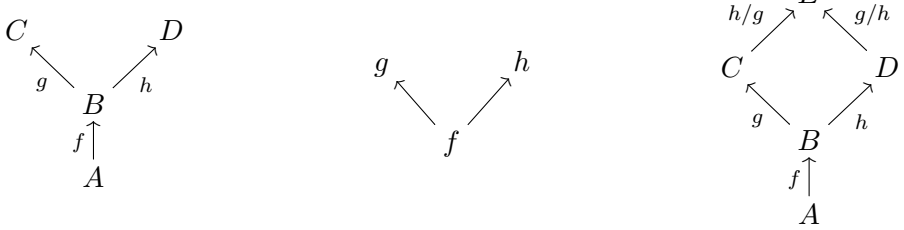
Chapter 6

A categorical theory of patches

This chapter is mostly based on [MG13].

Version control systems (or VCS, such as subversion, `git` or Darcs) offer a very concrete and interesting case of concurrent model. These are used to store documents (mainly text), which are concurrently edited by multiple people. The servers containing those documents are called *repositories*. These VCS mainly implement two operations: when a user is happy with the modifications he has brought to his local copy the document, he can *commit* them, which stores the modifications on the server; he can also *update* his files by pulling new modifications performed by others from the server. For efficiency reasons, the whole file is not transmitted each time, but only a *patch* storing the differences between the current and the last version.

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 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:



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. This suggests somehow considering a category with files as objects and morphisms as patches, and modeling residuals as pushouts in this category. To the best of our knowledge, our article [MG13] is the first to propose this simple idea. Most usual VCS (such as `git`) do not seriously take residuation of patches into account and only implement heuristics (e.g.

recursive 3-way merge) in order to be able to apply multiple concurrent patches, Darcs however tries to do this properly by "commuting patches", but considering the commutation relation expressing $g.(h/g) \sim h.(g/h)$ instead of the residuation operation does not easily allow us to consider universal properties. A notion of residuation is also investigated in the community of *operational transformations* [RNRG96], various formalizations for those have been proposed, but their categorical properties are rarely studied.

As expected, not every pair of coinitial morphisms have a pushout in the category of files: this reflects the fact that two patches can be conflicting (for instance if two users modify the same line of a file), otherwise the patches are said to be *compatible*. Representing and handling such conflicts in a coherent way is one of the most difficult issues in implementing a VCS (as witnessed for instance by the various proposals for Darcs: mergers, conflictors, graphictors, etc. [R$^+$13]). We believe that instead of trying various possible axiomatizations for the operations on patches and waiting for someone to come up with a corner case, one should start from the universal properties the model should satisfy and then work out the implementation details. In this case, our proposal is to compute a free completion under pushouts of the category of files. This category can easily be shown to exist for abstract reasons, and our main contribution is to work out a concrete description of it, which results in a new representation of files with conflicts, roughly as a preordered set of lines.

Among other works concerned with this problem, we would like to mention other interesting approaches using inverse semigroups in [Jac09], homotopy type theory [AMLH14]. Houston described ideas similar to ours in a series of blog posts [Hou12, Hou14]. Finally, to our great excitement, the present work has also been used as the theoretical basis for a new experimental version control system named *Pijul* [BLM15].

We introduce a category modeling files and patches (Section 6.1) and provide a concrete description of its free conservative finite cocompletion (Section 6.2). Finally, we briefly mention some possible extensions of this work (Section 6.3).

6.1 A category of files and patches

We begin by investigating 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 $L = \{a, b, \ldots\}$ of lines (typically, L is the set of words over some alphabet, such as the UTF-8 characters). A *file* is a word $A = a_0 a_1 \ldots a_{n-1}$ over lines, which will be represented here as a function $A : [n] \to L$, where $n \in \mathbb{N}$ is the number of lines in the file. A morphism $f : A \to B$ between two files $A : [m] \to L$ and $B : [n] \to L$ is an injective increasing partial function $f : [m] \to [n]$ such that $\forall i \in [m], B \circ f(i) = A(i)$ whenever $f(i)$ is defined. Such morphism is called a *patch*.

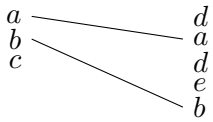
Definition 162 ([MG13])**.** The category $\mathcal{L}$ has files as objects and patches as morphisms.

Notice that the category $\mathcal{L}$ is isomorphic to the simplicial category when L is reduced to one element, so that it can be thought of as a labeled variant of it.

The category $\mathcal{L}$ is strictly monoidal, the tensor $A \otimes B$ of two files $A : [m] \to L$ and $B : [n] \to L$, being the file $A \otimes B : [m + n] \to L$ being defined as $(A \otimes B)(i) = A(i)$ if $i < m$ and $(A \otimes B)(i) = B(i - m)$ otherwise, the unit being the empty file $I : [0] \to L$, 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 (which is why we defined morphisms in this way):

Proposition 163 ([MG13])**.** *The category $\mathcal{L}$ is the free monoidal category containing L as (subcollection of) objects and containing, for every line $a \in L$, morphisms $\eta_a : I \to a$ (insertion of a line a) and $\varepsilon_a : a \to I$ (deletion of a line a) such that $\varepsilon_a \circ \eta_a = \mathrm{id}_I$ (deleting an inserted line amounts to doing nothing).*

Example 164. 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 $f : [3] \to [5]$ such that $f(0) = 1$ and $f(1) = 4$ and $f(2)$ is undefined. Graphically,



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.

6.2 The free conservative cocompletion

6.2.1 The general approach

Suppose that A is a file which is edited by two users, respectively applying patches $f_1 : A \to A_1$ and $f_2 : A \to A_2$ to the file. For instance,

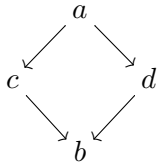
$$a \quad c \quad c \quad b \quad \xleftarrow{f_1} \quad a \quad b \quad \xrightarrow{f_2} \quad a \quad b \quad c \quad d \tag{6.1}$$

Now, each of the two users imports the modification from the other one. The resulting 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 (6.1). Now, it can be noticed that not every diagram in $\mathcal{L}$ has a pushout. For instance, the diagram

$$a \quad c \quad b \quad \xleftarrow{f_1} \quad a \quad b \quad \xrightarrow{f_2} \quad a \quad d \quad b \tag{6.2}$$

does not admit a pushout in $\mathcal{L}$. In this case, the two patches f_1 and f_2 are said to be *conflicting*.

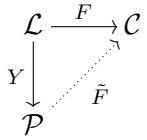
In order to represent the state of files after applying two conflicting patches, we investigate the definition of a category $\mathcal{P}$ which is obtained by completing the category $\mathcal{L}$ under all pushouts. Since, this completion should also contain an initial object (i.e. the empty file), we are actually defining the category $\mathcal{P}$ as the free completion of $\mathcal{L}$ 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 [Mac98]. 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
<<<<<<< HEAD
c
=======
d
>>>>>>> 5c55...
b
```
(6.3)

which would intuitively model the pushout of the diagram (6.2) if it existed, indicating 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 $\mathcal{L}$ reflects the fact that its objects are files whose lines are linearly ordered, whereas the objects of $\mathcal{P}$ can be thought as files whose lines are only partially ordered. More formally, the category is defined as follows.

Definition 165. The category $\mathcal{P}$ is the *free finite conservative cocompletion* of $\mathcal{L}$: it is (up to equivalence of categories) the unique finitely cocomplete category together with an embedding functor $Y : \mathcal{L} \to \mathcal{P}$ preserving existing finite colimits, such that for every finitely cocomplete category $\mathcal{C}$ and functor $F : \mathcal{L} \to \mathcal{C}$ preserving existing finite colimits, there exists, up to unique isomorphism, a unique functor $\tilde{F} : \mathcal{P} \to \mathcal{C}$ preserving all finite colimits and satisfying $\tilde{F} \circ Y = F$:



Above, the term *conservative* refers to the fact that we preserve colimits which already exist in $\mathcal{L}$ (we will only consider such completions here). Since the category $\hat{\mathcal{L}}$ is the free cocompletion of $\mathcal{L}$ (see Section 2.1), one can expect that the finite conservative cocompletion can be obtained as a full subcategory of $\hat{\mathcal{L}}$. The following folklore theorem, often attributed to Kelly, shows that it is indeed the case:

Theorem 166 ([Kel82, AR94])**.** *The conservative cocompletion of the category $\mathcal{L}$ is equivalent to the full subcategory of $\hat{\mathcal{L}}$ whose objects are presheaves which preserve finite limits, i.e. the image of a limit in $\mathcal{L}^{\mathrm{op}}$ (or equivalently a colimit in $\mathcal{L}$) is a limit in* **Set** *(and limiting cones are transported to limiting cones). The finite conservative cocompletion $\mathcal{P}$ can be obtained by further restricting to presheaves which are finite colimits of representables.*

In the above theorem, the Yoneda functor $Y : \mathcal{L} \to \hat{\mathcal{L}}$, can be shown to corestrict into a functor $Y : \mathcal{L} \to \mathcal{P}$, providing the unit of the completion [AR94].

Extracting a concrete description of the category $\mathcal{P}$ from the above proposition is a challenging task, because we a priori need to characterize firstly all diagrams admitting a colimit in $\mathcal{L}$, and secondly all presheaves in $\hat{\mathcal{L}}$ which preserve those diagrams. We introduce here a general methodology to build such a category.

6.2.2 The case of unlabeled insertions

In order to ease our presentation, we first consider here the variant of the model where only insertions of lines are allowed and the set L of lines contains only one element. The case of multiple lines will be obtained as a labeled variant using a slice construction (Section 6.2.3) and the extension to deletions of lines can be obtained by slightly generalizing the computations done here (Section 6.2.5)

In this section, we suppose that L is reduced to an element (so that we will not mention the labels), and still denote by $\mathcal{L}$ the subcategory of the previously defined category $\mathcal{L}$ restricted to total injective increasing functions as morphisms. Proposition 163 can easily be adapted to show that $\mathcal{L}$ is now the free monoidal category containing a morphism $\eta : 0 \to 1$, and the category can also be described as the free category generated by morphisms $s_i^n : n \to n+1$,

for $i \in [n+1]$, subject to the relations $s_i^{n+1} \circ s_j^n = s_{j+1}^{n+1} \circ s_i^n$ for $0 \leqslant i < j < n$. The second description can be obtained from the first one by defining $s_i^n = \mathrm{id}_i \otimes \eta \otimes \mathrm{id}_{n-i}$. Notice that the category $\mathcal{G} = \; 1 \rightrightarrows 2$ described in Section 2.1, such that $\hat{\mathcal{G}}$ is the category of graphs, is the full subcategory of $\mathcal{L}$ on the objects 1 and 2: the object 1 is the object of vertices, 2 is the object of edges, and $s_0^1, s_1^1 : 1 \to 2$ respectively correspond to source and target maps (we sometimes omit the superscript for those maps). This functor induces a restriction functor $I^* : \hat{\mathcal{L}} \to \hat{\mathcal{G}}$ which will allow us to consider vertices, edges, paths, etc. in a presheaf of $\hat{\mathcal{L}}$.

In order to simplify computations, it will be useful to describe an object of $\mathcal{L}$ as a colimit of objects of the subcategory $\mathcal{G}$. As explained in Section 2.1, the inclusion functor $I : \mathcal{G} \to \mathcal{L}$ induces a *nerve functor* $N_I : \mathcal{L} \to \hat{\mathcal{G}}$ such that $N_I(n) = \mathcal{L}(I-, n)$. An easy computation shows that the image $N_I(n)$ of $n \in \mathcal{L}$ is a graph with n vertices, so that its objects are isomorphic to $[n]$, and there is an arrow $i \to j$ for every $i, j \in [n]$ such that $i < j$. For instance,

$N_I(3) = \; 0 \rightrightarrows 1 \rightrightarrows 2$ $\qquad\qquad$ $N_I(4) = \; 0 \rightrightarrows 1 \rightrightarrows 2 \rightrightarrows 3$

It is therefore easy to check that this embedding is full and faithful, i.e. morphisms in $\mathcal{L}$ correspond to natural transformations in $\hat{\mathcal{G}}$. The functor $I : \mathcal{G} \to \mathcal{L}$ is thus dense in the following sense, see [MM92] for details.

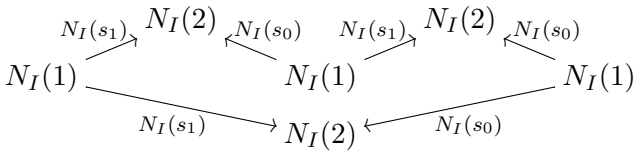
Definition 167. A functor $F : \mathcal{C} \to \mathcal{D}$ is *dense* if it satisfies one of the two equivalent conditions:

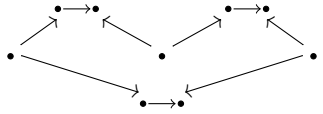
(i) the associated nerve functor $N_F : \mathcal{D} \to \hat{\mathcal{C}}$ is full and faithful,

(ii) every object of $\mathcal{D}$ is canonically a colimit of objects in $\mathcal{C}$: for every $D \in \mathcal{D}$,

$$D \quad \cong \quad \mathrm{colim}(\mathrm{El}(N_F D) \xrightarrow{\pi} \mathcal{C} \xrightarrow{F} \mathcal{D}) \tag{6.4}$$

The second equivalent condition essentially means that every object is the realization of its nerve, i.e. $D = R_F \circ N_F(D)$, except that the realization functor R_F might not be defined everywhere because the category $\mathcal{D}$ is not supposed to be cocomplete. The formula (6.4) is supposed to make this clear, but the category $\mathrm{El}(N_F D)$ can be replaced by the equivalent comma category F/D, whose definition is simpler. 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





We now investigate some properties of the free finite conservative cocompletion $\mathcal{P}$ of $\mathcal{L}$. Since the functor I is dense, every object of $\mathcal{L}$ is a finite colimit of objects in $\mathcal{G}$, and $\mathcal{G}$ does not have any non-trivial colimit. One could thus expect the free conservative finite cocompletion $\mathcal{P}$ of $\mathcal{L}$ to be the same as the free finite cocompletion of the category $\mathcal{G}$. We will see that this is not the case because the image in $\mathcal{L}$ of a non-trivial diagram in $\mathcal{G}$ might still have a colimit. By Theorem 166, the category $\mathcal{P}$ is the full subcategory of $\hat{\mathcal{L}}$ of presheaves preserving limits, which we now describe explicitly. This category will turn out to be equivalent to a full subcategory of $\hat{\mathcal{G}}$ (Theorem 173). We should first remark that those presheaves satisfy the following properties:

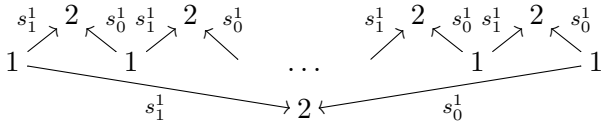
Proposition 168 ([MG13])**.** *Given a presheaf $P \in \hat{\mathcal{L}}$ which is an object of $\mathcal{P}$,*

1. *the underlying graph of P is finite,*
2. *for each non-empty path $x \twoheadrightarrow y$ there exists exactly one edge $x \to y$ (in particular 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 $P \in \mathcal{P}$, which is seen as a continuous presheaf in $\hat{\mathcal{L}}$ by Theorem 166. Given any natural number $n > 1$, the object $n+1$ is the colimit in $\mathcal{L}$ of the diagram

(6.5)

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



(6.6)

with $n+1$ occurrences of the object 1 also admits the object $n+1$ as colimit, we should have $P(n+1) \cong P(n+1) \times P(2)$ between any two vertices x and y, i.e. for every non-empty path $x \twoheadrightarrow y$ there exists exactly one edge $x \to y$. Also, since the object 0 is initial in $\mathcal{L}$, 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. □

Conversely, we wish to show that the conditions mentioned in the above proposition exactly characterize the presheaves in $\mathcal{P}$ among those in $\hat{\mathcal{L}}$. In order to prove this, by Theorem 166, we have to show that presheaves P satisfying these conditions preserve finite limits in $\mathcal{L}$, i.e. that for every finite diagram $D : \mathcal{J} \to \mathcal{L}$ admitting a colimit we have $P(\operatorname{colim} D) \cong \lim(P \circ D^{\mathrm{op}})$. It seems quite difficult to characterize the diagrams admitting a colimit in $\mathcal{L}$, however the following lemma shows that it is enough to check diagrams "generated" by a graph which admits a colimit.

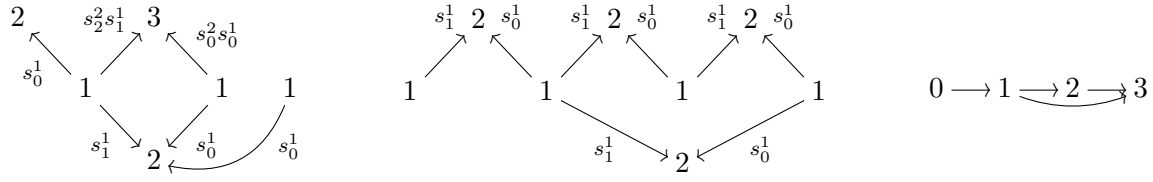
Lemma 169 ([MG13])**.** *A presheaf $P \in \hat{\mathcal{L}}$ preserves finite limits if and only if it sends the colimits of diagrams of the form*

$$\mathrm{El}(G) \xrightarrow{\pi_G} \mathcal{G} \xrightarrow{I} \mathcal{L} \tag{6.7}$$

to limits in **Set***, where $G \in \hat{\mathcal{G}}$ is a finite graph such that the above diagram admits a colimit. Such a diagram in $\mathcal{L}$ is said to be* generated by the graph G*.*

Proof. In order to check that a presheaf $P \in \hat{\mathcal{L}}$ preserves finite limits, we have to check that it sends colimits of finite diagrams in $\mathcal{L}$ *which admit a colimit* to limits in **Set**, and therefore we have to characterize diagrams which admit colimits in $\mathcal{L}$. Suppose given a

diagram $K : \mathcal{J} \to \mathcal{L}$. Since I is dense, every object of $\mathcal{L}$ is a colimit of a diagram involving only the objects 1 and 2 (see Definition 167). 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_0^1 and s_1^1 as arrows (these are the only non-trivial arrows in $\mathcal{L}$ whose source and target are 1 or 2), in which every object 2 is the target of exactly one arrow s_0^1 and one arrow s_1^1. For instance, the diagram in $\mathcal{L}$ below on the left admits the same colimits as the diagram in the middle, which is shown on the right.



Any such diagram K is obtained by gluing a finite number of diagrams of the form $1 \xrightarrow{s_1^1} 2 \xleftarrow{s_0^1} 1$ along objects 1, and is therefore of the form $\mathrm{El}(G) \xrightarrow{\pi} \mathcal{G} \xrightarrow{I} \mathcal{L}$ for some finite graph $G \in \hat{\mathcal{G}}$: 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_1^1 and s_0^1 admitting it as target. For 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** [Par73, SW73]. □

Among diagrams generated by graphs, those admitting a colimit can be characterized as follows.

Lemma 170 ([MG13])**.** *Given a graph $G \in \hat{\mathcal{G}}$, the associated diagram* (6.7) *admits a colimit in $\mathcal{L}$ if and only if there exists $n \in \mathcal{L}$ and a morphism $f : G \to N_I n$ in $\hat{\mathcal{L}}$ such that every morphism $g : G \to N_I m$ in $\hat{\mathcal{L}}$, with $m \in \mathcal{L}$, factorizes uniquely through $N_I n$:*

$$G \underset{g}{\overset{f}{\rightrightarrows}} N_I n \dashrightarrow N_I m$$

Proof. Follows from the existence of the partially defined left adjoint R_I to N_I, in the sense of [Par73], given by the fact that I is dense (see Definition 167). □

From the previous lemma, one can obtain the following, more concrete, characterization of diagrams admitting colimits:

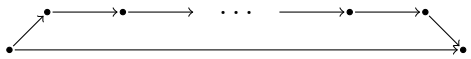
Lemma 171 ([MG13])**.** *A finite graph $G \in \hat{\mathcal{G}}$ induces a diagram* (6.7) *in $\mathcal{L}$ which admits a colimit if and only if it is*

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

We can finally deduce that the conditions we have found for colimit-preserving presheaves provide in fact a complete characterization.

Proposition 172 ([MG13])**.** *The free conservative finite cocompletion $\mathcal{P}$ of $\mathcal{L}$ is equivalent to the full subcategory of $\hat{\mathcal{L}}$ whose objects are presheaves P satisfying the conditions of Proposition 168.*

Proof. By Lemma 169, the category $\mathcal{P}$ is equivalent to the full subcategory of $\hat{\mathcal{L}}$ whose objects are presheaves preserving limits of diagrams of the form (6.7) generated by some graph $G \in \hat{\mathcal{G}}$ which admits a colimit, i.e. by Lemma 171 the finite graphs which are acyclic and connected. We write G_n for the graph with $[n]$ as vertices and edges $i \to (i+1)$ for $0 \leqslant i < n-1$. It can be shown that any acyclic and connected finite graph can be obtained from the graph G_n, for some $n \in \mathbb{N}$, by iteratively adding an edge $x \to y$ for some vertices x and y such that there exists a non-empty path $x \twoheadrightarrow y$. Namely, suppose given an acyclic and connected finite graph G. The relation $\leqslant$ on its vertices, defined by $x \leqslant y$ whenever there exists a path $x \twoheadrightarrow y$, 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 $x \to y$ with $x \leqslant y$, otherwise it would not be acyclic. Since by Proposition 168, see (6.6), the diagram generated by a graph of the form



is preserved by presheaves in $\mathcal{P}$ (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 $\mathcal{P}$ preserve diagrams generated by graphs G_n. This follows again by Proposition 168, see (6.5). □

One can notice that a presheaf $P \in \mathcal{P}$ 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: $P \cong I_*(I^*P)$. We can therefore simplify the description of the cocompletion of $\mathcal{L}$ as follows:

Theorem 173 ([MG13])**.** *The free conservative finite cocompletion $\mathcal{P}$ of $\mathcal{L}$ is equivalent to the full subcategory of the category $\hat{\mathcal{G}}$ of graphs, whose objects are finite graphs such that for every non-empty path $x \twoheadrightarrow y$ there exists exactly one edge $x \to y$. Equivalently, it can be described as the category whose objects are finite sets equipped with a transitive relation $<$, and functions respecting the relation.*

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

Proposition 174 ([MG13])**.** *With the last above description, the pushout of a diagram in $\mathcal{P}$*

$$(B, <_B) \xleftarrow{f} (A, <_A) \xrightarrow{g} (C, <_C)$$

is $B \sqcup C/\sim$ with $B \ni b \sim c \in C$ whenever there exists $a \in A$ with $f(a) = b$ and $f(a) = c$, equipped with the transitive closure of the relation inherited by $<_B$ and $<_C$.

6.2.3 Labeled files

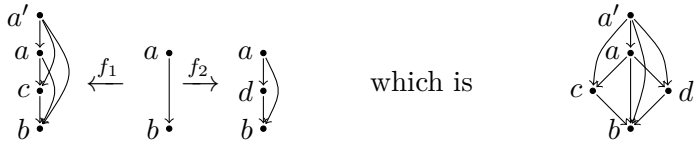
The construction can be extended to the labeled case by using a comma category construction. The forgetful functor $\hat{\mathcal{L}} \to \mathbf{Set}$ sending a presheaf P to the set $P(1)$ admits a right adjoint $! : \mathbf{Set} \to \hat{\mathcal{L}}$. Given $n \in \mathbb{N}^*$ the elements of $!\,L(n)$ are words u of length n over L, with $!L(s_i^{n-1})(u)$ being the word obtained from u by removing the i-th letter. The free conservative finite cocompletion $\mathcal{P}$ of $\mathcal{L}$ is the slice category $\mathcal{L}/!\,L$, whose objects are pairs (P, ℓ) consisting of a finite presheaf $P \in \hat{\mathcal{L}}$ together with a *labeling* morphism $\ell : P \to !\,L$ of presheaves. Alternatively, the description of Theorem 173 can be straightforwardly adapted by labeling the elements of the objects by elements of L (labels should be preserved by morphisms), thus justifying the use of labels for the vertices in the following examples.

Remark 175. One could also want to allow changing the contents of a line in patch: merging files, the patch changing a line a to a' and the patch deleting a and inserting a' instead do not behave in the same way. In this case, we would consider the comma category $\mathcal{L}/!$. In this case, we would get a similar description for $\mathcal{P}$ except that morphisms are not required to preserve labels. We did not do so here because traditional VCS do not allow for such operations.

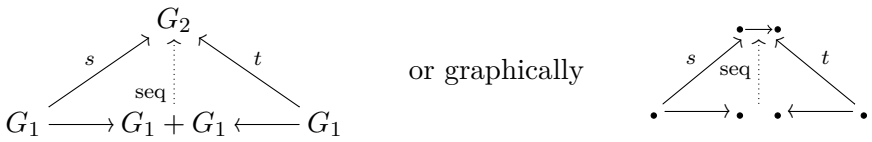
6.2.4 Examples of pushouts

In this section, we give some examples of merging, i.e. pushouts, of patches.

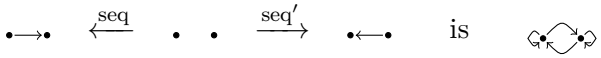
Example 176. Suppose that starting from a file ab, one user inserts a line a' at the beginning and c in the middle, while another one inserts a line d in the middle. After merging the two patches, the resulting file is the pushout of



Example 177. 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 $s, t : G_1 \to G_2$ for the two morphisms in $\mathcal{P}$. Since $\mathcal{P}$ is finitely cocomplete, there is a coproduct $G_1 + G_1$ which gives, by the universal property, an arrow $\mathrm{seq} : G_1 + G_1 \to G_2$:

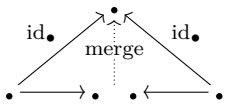


that we call the *sequentialization morphism*. This morphism corresponds to the following patch: given two possibilities for a line, a user can decide to turn them into two consecutive lines. We also write $\mathrm{seq}' : G_1 + G_1 \to G_2$ for the morphism obtained similarly by exchanging s and t in the above cocone. Now, the pushout of



which illustrates how cyclic graphs appear in $\mathcal{P}$ during the cocompletion of $\mathcal{L}$.

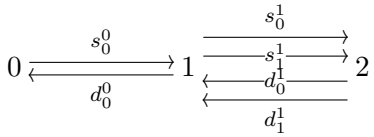
Example 178. With the notations of the previous example, by taking the coproduct of two copies of $\mathrm{id}_{G_1} : G_1 \to G_1$, there is a universal morphism $G_1 + G_1 \to G_1$, which illustrates how two independent lines can be merged by a patch (in order to resolve conflicts).



6.2.5 Patches with deletions

The methodology should generalize in order to compute the free conservative finite cocompletion of the category $\mathcal{L}$ as introduced in Definition 162, i.e. with deletions of lines. The

category $\mathcal{G}$ we should now consider is the full subcategory of $\mathcal{L}$ on the objects 0, 1 and 2, and we still write $I : \mathcal{G} \to \mathcal{L}$ for the inclusion functor. This category is the free category on the graph on the left subject to the relations on the right:



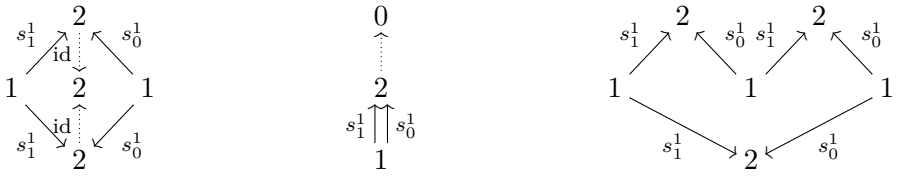
$$\begin{array}{lcl} s_0^1 s_0^0 & = & s_1^1 s_0^0 \\ d_0^0 d_0^1 & = & d_0^0 d_1^1 \end{array} \qquad \begin{array}{lcl} d_0^0 s_0^0 & = & \mathrm{id}_1 \\ d_0^1 s_0^1 & = & \mathrm{id}_2 \\ d_1^1 s_1^1 & = & \mathrm{id}_2 \end{array} \qquad \begin{array}{lcl} d_1^1 s_0^1 & = & s_0^0 d_0^0 \\ d_0^1 s_1^1 & = & s_0^0 d_0^0 \end{array}$$

Moreover, Proposition 168 can be generalized as follows:

Proposition 179. *Given a presheaf $P \in \hat{\mathcal{L}}$ which is an object of $\mathcal{P}$,*

1. *the sets $P(1)$ and $P(2)$ are finite,*
2. *$P(0)$ is reduced to one element $\star$,*
3. *given vertices $x, y \in P(1)$ there is at most one edge $x \to y$ in $P(2)$,*
4. *the only vertex $x \in P(1)$ such that there is an edge $x \to x$ in $P(2)$ is $\bot = P(d_0^0)(\star)$,*
5. *$P(n)$ is isomorphic to the set of n-uples of vertices $(x_i)_{i \in [n]}$ such that there is an edge $x_i \to x_j$ for every $i, j \in [n]$ with $i < j$.*

Proof. The first point follows from the fact that every object in $\mathcal{P}$ should be obtained as a finite colimit of representables. The other points follow from the preservation of some colimits in $\mathcal{L}$. Respectively, we have that 0 is initial in $\mathcal{L}$, and



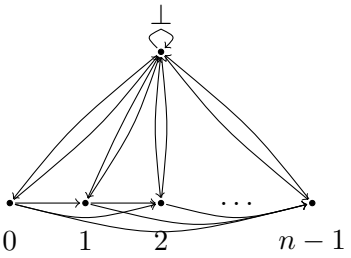
2 is the colimit of the diagram on the left, 0 is the coequalizer shown in the middle, and n is the colimit of a suitable diagram (we have shown the diagram for $n = 3$ on the right). □

The restriction of $\hat{\mathcal{G}}$ to presheaves satisfying the above conditions can be seen as the category of graphs with partial morphisms. The situation is a variant of the following well-known one. A *pointed set* $(A, \star)$ consists of a set together with a distinguished element $\star \in A$, a morphism between those is supposed to preserve the distinguished element, and we write **pSet** for the resulting category. We also write **PSet** for the category of sets and partial functions. We can define functor $F : \mathbf{PSet} \to \mathbf{pSet}$, sending a set A to $(A \sqcup \{\star\}, \star)$ and a partial function $f : A \to B$ to its total extension $Ff : A \sqcup \{\star\} \to B \sqcup \{\star\}$ sending an element on which f is not defined to $\star$. This functor is easily seen to be an equivalence of categories: a partial function is the same as a total function which might return a "default value". In the case of an element $P \in \hat{\mathcal{G}}$ with $P(0) = \{\star\}$, the presheaf P can be seen as a graph with default values as follows: the elements of $P(1)$ (resp. $P(2)$) are the vertices (resp. edges) and $P(0)$ can simply be ignored since it is trivial. The vertex $\bot = P(d_0^0)(\star)$ plays the role of a default value for vertices, and for every vertex $x \in P(1)$ there is an edge $P(d_0^1)(x) : \bot \to x$ and an edge $P(d_1^1)(x) : x \to \bot$ playing the role of a default value for edges whenever the image of the source or the target of an edge is not defined.

Example 180. Given a natural number $n \in \mathcal{L}$, the graph $N_I n \in \hat{\mathcal{G}}$ is such that

$N_I n 0 = \mathcal{L}(I0, n) \cong \{\star\}$
$N_I n 1 = \mathcal{L}(I1, n) \cong [n] \sqcup \{\bot\}$
$N_I n 2 = \mathcal{L}(I2, n) \cong \{(i, j) \in [n] \times [n] \mid i < j\} \sqcup \{(\bot, i) \mid i \in [n]\} \sqcup \{(i, \bot) \mid i \in [n]\} \sqcup \{(\bot, \bot)\}$

It can thus be pictured as



Proposition 181 ([MG13])**.** *Consider the full subcategory of $\hat{\mathcal{G}}$ whose objects are presheaves P such that $P(0)$ is reduced to one element $\star$, there is at most one edge $\bot \to \bot$, and for every $x \in P(0)$ there is at most one edge $\bot \to x$ and at most one edge $x \to \bot$, where $\bot = P(d^0_0)(\star)$. This category is equivalent to the category of graphs and "partial morphisms": the functions on objects and morphisms are partial, and the image of an edge is defined if and only if both the image of its source and target are.*

In particular, it is easy to check that a morphism $f : N_I m \to N_I n$ in $\hat{\mathcal{G}}$ is the same, under the correspondence of the previous proposition, as a partial increasing function $[m] \to [n]$, and thus that the functor I is dense. We believe that the conditions given in Proposition 179 are sufficient to characterize the presheaves preserving the colimits existing in $\mathcal{L}$ and that the rest of the proof goes through as in Section 6.2.2.

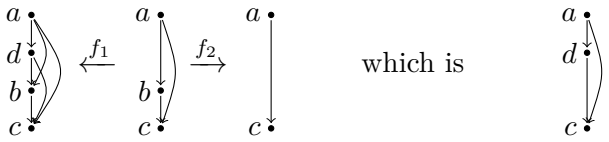
Conjecture 182. *The free conservative finite cocompletion of $\mathcal{L}$ is the full subcategory $\mathcal{P}$ of $\mathcal{L}$ whose objects are presheaves satisfying the conditions of Proposition 179.*

Finally, the extension to the labeled case should be the following one.

Conjecture 183. *The free conservative finite cocompletion of $\mathcal{L}$ is the category $\mathcal{P}$ whose objects are triples $(A, <, \ell)$ where A is a set of lines, $<$ is an irreflexive relation on A, and $\ell : A \to L$ associates a label with each line, and morphisms $f : (A, <_A, \ell_A) \to (B, <_B, \ell_B)$ are partial functions $f : A \to B$ such that for every $a, a' \in A$ both admitting an image under f, we have $\ell_B(f(a)) = \ell_A(a)$, and $a <_A a'$ implies $f(a) <_B f(a')$.*

In this category, the natural way of modeling the insertion of a line in a file $(A, <, \ell)$ is to add a new element x to A, with the desired label, and extend the relation so that $y < x$ for every line before x and $x < y$ for every line after x. The deletion of a line is obtained by removing the corresponding element of A and restricting $<$ and ℓ.

Example 184. 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 $\mathcal{P}$) is the pushout of



i.e. the file *adc*. Notice that the morphism f_2 is partial: b has no image.

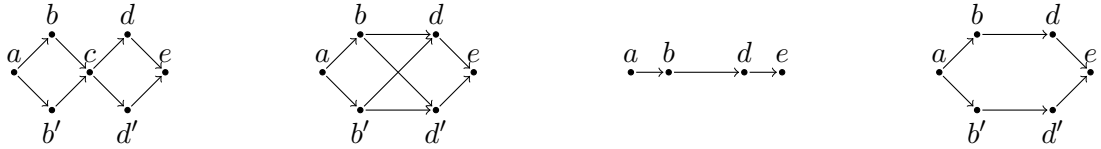
Contrarily to what is stated in Conjecture 183, we have claimed in [MG13, Theorem 20] that the category resulting from the cocompletion could be described as the category whose objects are triples $(A, <, \ell)$, where $<$ is a transitive relation (which may contain loops), and morphisms are partial functions preserving the relation and labels, and that pushouts are computed by the obvious generalization of Proposition 174. However, while writing this memoir, we became aware of the following counterexample found by Houston [Hou14], which shows that our claim was incorrect. We believe that the version presented here is the right answer, but we prefer to state it as a conjecture for now until the result is published. We have traced back the error to an incorrect computation of pushouts: the diagram on the left

does not admit 3 as pushout (contrarily to the case of Section 6.2.2 where $\mathcal{L}$ has total morphisms). For instance, the cocone on the right such that $f(0)$ (resp. $f'(0)$, resp. $f''(0)$) is 1 (resp. undefined, resp. 0) does not induce a universal arrow $g : 3 \to 2$. Intuitively, if it existed, the morphism g should be such that $g(0) = 1$, $g(1)$ is undefined and $g(2) = 0$, but such a morphism would not be strictly increasing.

Houston's counterexample is as follows. Consider the file *ace* and the following three possible patches from this file:

1. insert b and d in order to obtain the file *abcde*,
2. insert b' and d' in order to obtain the file $ab'cd'e$,
3. delete c in order to obtain the file *ae*.

The file obtained by applying both first and second patches is shown on the left, and if we then also apply the third patch we obtain the file in the middle left. The file obtained by applying both first and third patches is shown in the middle right, and if we then also apply the second patch we obtain the file on the right.



For clarity, we have not drawn the transitive edges: the relation $<$ is the transitive closure of the adjacency relation of the graphs. The file in the middle left and the one on the right are not equal, whereas they should be since pushouts are supposed to be associative. Of course, in the category $\mathcal{P}$ described in Conjecture 183, the two sequences of pushouts give rise to the same answer, which is the transitive closure of the diagram on the right above.

6.3 Future work

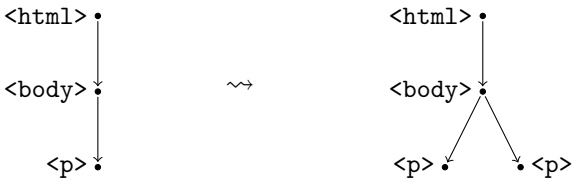
Handling structured documents. We believe that the interest of our methodology lies in the fact that it adapts easily to other more complicated base categories $\mathcal{L}$ than the two investigated here. We plan to investigate variants of the model extended in order to cope with

different file types (containing text, or more structured data such as XML trees) and multiple files (which can be moved, deleted, etc.).

A natural way to model XML files is to consider, for $\mathcal{L}$, the category whose objects are trees with vertices labeled in L, and morphisms are partial injective functions preserving dependencies and labels. For instance, adding a paragraph to an HTML file as in

```
<html>                                  <html>
  <body>                                  <body>
    <p>Hello world!</p>          ⇝          <p>Hello world!</p>
  </body>                                   <p>How are you today?</p>
</html>                                   </body>
                                        </html>
```

would be modeled by the following morphism of labeled trees:

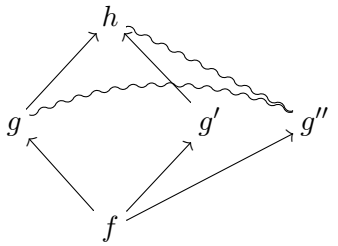


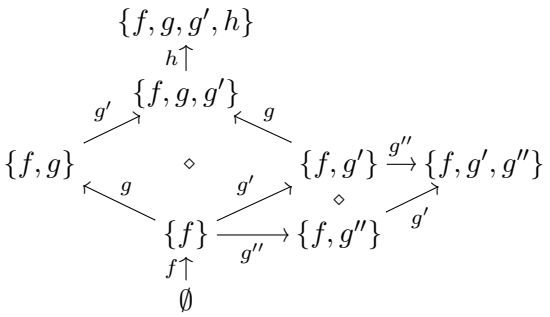
Similarly, a file system could be modeled as the comma category $\mathbf{Set}_+/\mathcal{L}$, i.e. sets of files labeled in actual files, where $\mathbf{Set}_+$ is the category of partial injective functions. The suitable cocompletion of those categories remains to be investigated.

Modeling repositories. A *repository* is the server containing all the patches pushed by the various participants. These patches are naturally ordered by their order of creation: a patch depends on the patches which were already applied in the local copy of the user when he has created it. For instance, suppose that there are three users u_0, u_1 and u_2 and the following scenario happens:

— u_0 creates a patch f, which is then imported by users u_1 and u_2,
— concurrently u_0 creates a patch g, u_1 creates a patch g' and u_2 creates a patch g'',
— u_0 imports the patch g' and creates a patch h,
— in the end all the users upload their patches on the repository.

We suppose that the patches g and g' are compatible, as well as g' and g'', i.e. they have a pushout in $\mathcal{L}$, but g and g'' are not compatible, as well as g'' and h. The partial order of the patches, as well as their compatibility, is naturally encoded as an event structure, see Section 2.4.2. For instance, the one corresponding to the situation described above is depicted on the left:





The associated asynchronous graph (see Section 2.4.2) is depicted on the right. Its vertices correspond to the possible states of conflict-free repositories and the states of all possible repositories can be obtained as the vertices of the asynchronous graph associated with the event structure obtained from the one above by removing conflict: the effect of the cocompletion is to formally remove all possible conflicts. For each such state of repository, the corresponding file can be obtained, starting from the empty file, by applying (residuals of) patches according to any path from $\emptyset$ to the state. We believe that most usual operations on repositories can be modeled in this context: cherry-picking (importing only one patch from another repository), using branches, removing a patch, etc.

Adding inverses. It would also be interesting to explore axiomatically the addition of inverses for patches, in order to model the reverting of an already applied patch. Notice that if we start from a category $\mathcal{L}$ which is a connected groupoid, it already has finite colimits and the completion will not change it. This means that inverses have to be added after conflicts have been resolved: one has to be able to distinguish between "positive patches" (those doing something) and their inverses (which are reverting preceding patches). Otherwise any file is a good resolution for a conflict since there is a morphism between any two files anyway.

Algorithmic operations. Once the theoretical setting will be clearly established, we plan to investigate algorithmic issues; in particular, how to efficiently represent and manipulate the conflicting files, which are objects in $\mathcal{P}$. Some work in this direction has already been undertaken by Becker, Lepigre and Meunier, during the implementation of the version control system Pijul [BLM15], which is based on the patch theory described in this chapter.

Part II

Higher-Dimensional Rewriting

Chapter 7

Higher-dimensional rewriting systems

We present a generalization of rewriting systems and their techniques to higher dimensions: *polygraphs*, that we will often simply be calling *higher-dimensional rewriting systems*. Those were independently introduced by Street [Str76, Pow91] (under the name of *computad*) and Burroni [Bur93] for the following motivations. Firstly, string rewriting systems can be thought of as variants of presentations of monoids, with oriented relations, which often helps deciding the word problem. Since a monoid is a category with only one object, can we generalize the notion of rewriting system in "higher dimension" in order to obtain an oriented presentation of an n-category? Secondly, we would like to see a clear inductive pattern emerge: an n-dimensional rewriting system can also be considered as a signature for an $(n+1)$-dimensional rewriting system. Thirdly, the notion of CW-complex, i.e. roughly spaces built up by gluing disks, has turned out to be central in (algebraic) topology: what is an algebraic counterpart of these topological structures?

The resulting setting is very rich and still an active subject of research. Many of the properties of rewriting or homotopy in this context are still to be discovered; we try here to sketch a brief panorama of the field, omitting many details. We will mostly need only the definitions in low dimension (in Chapter 8 and 9 in particular), but our work is a part of this general context, and we believe that it is necessary to present it in order to properly motivate what we have done. We begin by recalling string rewriting systems before generalizing them. Unless otherwise stated, the rewriting systems we consider are implicitly supposed to be finite in order to simplify some formulations, since we will not need the general case.

We begin by recalling basic properties of string rewriting systems (Section 7.1) and then show how this traditional rewriting setting can be extended to higher dimensions by polygraphs (Section 7.2).

7.1 String rewriting systems

We begin by recalling the main properties of string rewriting systems. This presentation is of course far from being exhaustive, we refer to [BN99, BKd03] for details.

7.1.1 Presentations of monoids

String rewriting systems have been originally introduced by Thue in order to study word problems in monoids [Thu14]: from our perspective, these can provide small descriptions (presentations) of monoids, on which many computations can be performed, such as computing the homology of the monoid.

Definition 185. A *string rewriting system*, or SRS, $P = (P_1, s, t, P_2)$ consists of a set P_1, called the *alphabet* of *generators*, or *signature*, and a set P_2 of *rules* together with two functions $s, t : P_2 \to P_1^*$ associating with each rule its *source* and *target* respectively, which are words over P_1.

We often write $\rho : u \Rightarrow v$ for a rule $\rho \in P_2$ with $s(\rho) = u$ and $t(\rho) = v$, and $\langle P_1 \mid P_2 \rangle$ for an SRS (implicitly, the elements of P_2 come with their source and target and we sometimes omit the names of the rules when they are not relevant). We write $u \Rightarrow v$ whenever there exists a rule $\rho : u \Rightarrow v$, and $\overset{*}{\Leftrightarrow}$ for the smallest congruence (wrt to concatenation) containing $\Rightarrow$. We sometimes write $\Rightarrow_P$ instead of $\Rightarrow$ (and so on) when we need to distinguish between various rewriting systems. The monoid $\|P\|$ *presented* by such a string rewriting system is $\|P\| = P_1^*/P_2$, the quotient of the free monoid over P_1 by the relation $\overset{*}{\Leftrightarrow}$, and P is said to be a *presentation* of any monoid isomorphic to $\|P\|$.

Example 186. Consider the string rewriting system $P = \langle a, b \mid \gamma : ba \Rightarrow ab \rangle$. The monoid it presents is (isomorphic to) $\mathbb{N} \times \mathbb{N}$ equipped with componentwise addition.

Notice that a monoid always admits a presentation, as shown in the following lemma. However, we are interested in ones which are reasonably small and have good properties in order to be able to perform computations on those.

Lemma 187. *Given a monoid $(M, \cdot, 1)$, its* standard presentation *P^M, which is the* SRS *with $P_1^M = M \setminus \{1\}$ and $P_2^M = \{u_1 u_2 \to v \mid u_1 \cdot u_2 = v\}$, is a presentation of M.*

7.1.2 Tietze transformations

Two SRS P and Q are said to be *Tietze-equivalent* when they present the same monoid, i.e. $\|P\| \cong \|Q\|$. Given an SRS P, the following operations produce a new SRS P':

1. *adding a definable generator*: given $a \notin P_1$ and $u \in P_1^*$, we consider the SRS

$$P' \quad = \quad \langle P_1, a \mid P_2, a \Rightarrow u \rangle$$

2. *adding a derivable relation*: given $u, v \in P_1^*$ such that $u \overset{*}{\Leftrightarrow}_P v$, consider the SRS

$$P' \quad = \quad \langle P_1 \mid P_2, u \Rightarrow v \rangle$$

It is easy to see that these operations, called *Tietze transformations*, produce an SRS which is Tietze-equivalent to the original one, as well as their converses, i.e. going from P' to P. Tietze has shown that they actually form a complete set of operations generating the Tietze equivalence:

Proposition 188 ([Tie08])**.** *Two string rewriting systems P and Q are Tietze-equivalent if and only if P can be transformed into Q by using the two above operations and their converses.*

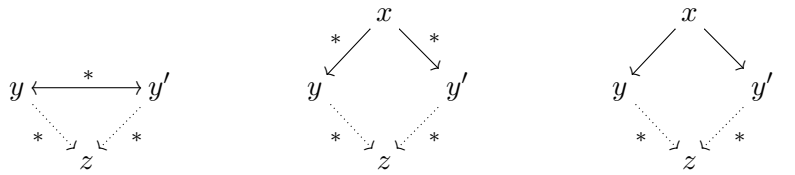
7.1.3 Confluence in abstract rewriting systems

In order to study the properties of a rewriting system, one can abstract away from the structure of words, which suggests considering the following framework [Hue80]:

Definition 189. An *abstract rewriting system*, or ARS, (P_0, s, t, P_1) is a graph, with P_0 as set of vertices, P_1 as set of edges and $s, t : P_1 \to P_0$ as source and target functions.

We write $x \to y$ whenever there exists an edge with x as source and y as target, $\overset{*}{\to}$ for the reflexive transitive closure of $\to$ and $\overset{*}{\leftrightarrow}$ for the equivalence relation generated by $\to$. In the case where $x \overset{*}{\to} y$, we say that x *rewrites* to y. By analogy with previous case, we write $\|P\|$ for the quotient set $P_0/\overset{*}{\leftrightarrow}$, which is called the *set presented* by the ARS. We now recall the basic properties considered in rewriting theory:

Definition 190. An ARS is
- *terminating* when there is no infinite sequence of vertices $(x_i)_{i\in\mathbb{N}}$ with $x_i \to x_{i+1}$ for every $i \in \mathbb{N}$,
- *Church-Rosser* when for every vertices y, y' such that $y \overset{*}{\leftrightarrow} y'$ there exists a vertex z such that $y \overset{*}{\to} z$ and $y' \overset{*}{\to} z$,
- *confluent* when for vertices x, y, y' such that $x \overset{*}{\to} y$ and $x \overset{*}{\to} y'$ there exists a vertex z such that $y \overset{*}{\to} z$ and $y' \overset{*}{\to} z$,
- *locally confluent* when for x, y, y' such that $x \to y$ and $x \to y'$ there exists a vertex z such that $y \overset{*}{\to} z$ and $y' \overset{*}{\to} z$,
- *convergent* when both terminating and confluent.

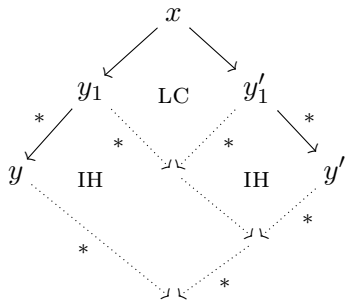


Church-Rosser confluence local confluence

Clearly, Church-Rosser implies confluence, and is in fact equivalent to it. Moreover, confluence implies local confluence, but the converse is not true. Newman's lemma shows that termination is enough to ensure that this is the case.

Lemma 191 ([New42])**.** *A terminating and locally confluent abstract rewriting system is confluent (and Church-Rosser).*

Proof. Since it is terminating, the relation $\to$ is well-founded, and by well-founded induction on it, we show that the local confluence property at a vertex x implies the confluence property at x. The base cases are immediate. Otherwise, we have a diagram of the form



which can be closed using either the local confluence hypothesis (LC) or the induction hypothesis (IH). □

A vertex x such that there exists y with $x \to y$ is called *reducible*, and a *normal form* when it is not reducible. In a terminating and (locally) confluent ARS, a normal form $\hat{x}$ can be canonically associated with each vertex x as follows. If we start from x and construct a sequence $x = x_0 \to x_1 \to x_2 \to \dots$ of vertices, this sequence will necessarily be finite, the last vertex is a normal form written $\hat{x}$, and because of confluence it only depends on x, not on the chosen path. This provides a canonical representative of equivalence classes of vertices under the relation $\overset{*}{\leftrightarrow}$: two vertices x and y are such that $x \overset{*}{\leftrightarrow} y$ if and only if $\hat{x} = \hat{y}$.

Any string rewriting system P induces an abstract rewriting system Q with $Q_0 = P_1^*$, $Q_1 = P_2$, and same source and target functions, which enables us to apply the previous definitions and lemma to SRS. Given two words $u, v \in P_1^*$, the *word problem* consists in determining whether they have the same image under the quotient morphism $P_1^* \to \|P\|$. Thus, when the SRS is finite and convergent, the word problem is decidable by computing normal forms as above: the elements of the monoid $\|P\|$ are in bijection with normal forms and two words in P_1^* are in the same equivalence class if and only if they have the same normal form.

7.1.4 Reduced rewriting systems

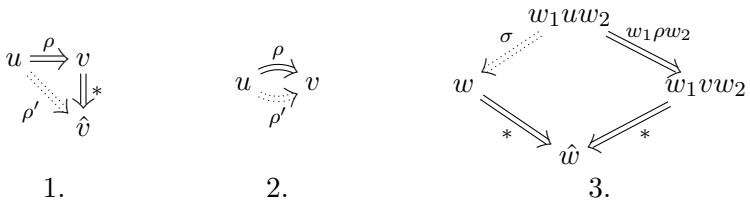
As a first application of Tietze transformations, it can be shown that one can restrict to reduced SRS without loss of generality. Computations (of resolutions for instance, see below) are easier to perform on SRS with this assumption.

Definition 192. An SRS P is *reduced* when for every rule $\rho : u \Rightarrow v$ in P_2, u is a normal form wrt $P \setminus \{\rho\}$ and v is a normal form wrt P.

Proposition 193 ([KB70, Mét83, Squ87])**.** *Any convergent* SRS *is Tietze-equivalent to a reduced and convergent one.*

Proof. A convergent SRS P can be transformed into a reduced one by successively applying the following operations, in this order, until they cannot be applied anymore:

1. if P_2 contains a rule $\rho : u \Rightarrow v$ with v reducible, replace it by $\rho' : u \Rightarrow \hat{v}$,
2. if P_2 contains two parallel rules $\rho, \rho' : u \Rightarrow v$, remove ρ',
3. if P_2 contains a rule $\rho : u \Rightarrow v$ and a rule $\sigma : w_1 u w_2 \Rightarrow w$, remove σ.



1. 2. 3.

(the above diagrams should make it clear that the transformations do not change the generated relation on words in P_1^*). The transformations can be checked to preserve finiteness, termination and confluence of the SRS. □

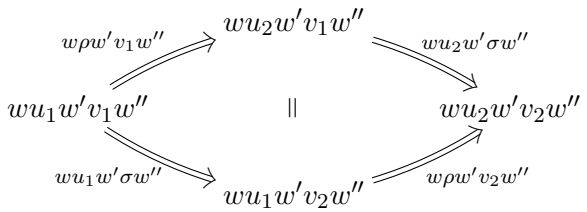
7.1.5 Rewriting paths

Any abstract rewriting system P generates a free category P^*: the free category on the graph P with vertices as objects and paths as morphisms. Notice that we have $x \xrightarrow{*} y$ if and only if there exists a morphism from x to y: the category allows us to consider the various rewriting paths between two vertices, as opposed to only wondering if a path exists.

For string rewriting systems a category of rewriting paths can be defined similarly, and has more structure: it is equipped with a monoidal tensor product induced by concatenation of words. More precisely, suppose fixed a set P_1 of generators. Any monoidal category $\mathcal{C}$ with P_1^* as underlying monoid of objects induces an SRS with the morphisms of $\mathcal{C}$ as rewriting rules, and this operations extends to a functor between suitable categories which admits a left adjoint. Thus, any SRS P induces a *free monoidal category* P^* and we write P_2^* for its set of morphisms. More explicitly, the category P^* has P_1^* as set of objects and morphisms are freely generated by triples (u, ρ, w), often simply written $u\rho w$ and called *rewriting steps*, with $u, w \in P_1^*$ and $\rho : v \Rightarrow v' \in P_2$ with type

$$u\rho w \quad : \quad uvw \quad \Rightarrow \quad uv'w$$

quotiented by the *exchange law*:



$$\text{for all} \begin{cases} u_1 \overset{\rho}{\Rightarrow} u_2 \text{ and } v_1 \overset{\sigma}{\Rightarrow} v_2 \text{ in } P_2 \\ w,\ w' \text{ and } w'' \text{ in } P_1^*. \end{cases} \tag{7.1}$$

There are obvious left and right actions of P_1^* on P_2^*, respectively sending $(u, w\rho w')$ to $uw\rho w'$ and $(w\rho w', u)$ to $w\rho w'u$, which extend to all morphisms in a way compatible with composition and identities. The tensor product of two morphisms $f : u \Rightarrow u'$ and $g : v \Rightarrow v'$ in P_2^* is then defined by $fg = u'g \circ fv = fv' \circ ug$. From a rewriting point of view, the morphisms in P_2^* correspond to *rewriting paths*, where the paths do not take into account the order of rewriting steps at disjoint positions in a word.

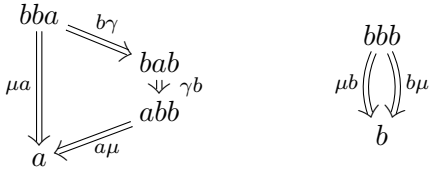
7.1.6 Critical branchings

In string rewriting systems, obstructions to confluence are generated by minimal ones which are called critical branchings, as we now recall. We suppose fixed an SRS P. A *branching* of P is a pair $(f : u \Rightarrow v, g : u \Rightarrow w)$ of morphisms of P_2^* with a common source; it is *local* when f and g are both rewriting steps. A local branching is *independent* when it is of the form $(f'v, ug')$ or $(vf', g'u)$ for some rewriting steps $f' : u \Rightarrow u', g' : v \Rightarrow v'$, and *overlapping* when it is not independent and f and g are distinct. A branching is *critical* when it is overlapping and minimal for the order generated on branchings by $(f, g) \preccurlyeq (ufv, ugv)$, for any words u and v. A critical branching is sometimes also called a *critical pair*. A branching $(f : u \Rightarrow v, g : u \Rightarrow w)$ is *confluent* when there exist a pair of 2-cells $f' : v \Rightarrow u'$ and $g' : w \Rightarrow u'$ in P_2^*. Notice that P is (locally) confluent when all of its (local) branchings are confluent. In order to test local confluence, it is enough to test it on critical branchings:

Lemma 194. *An* SRS *is locally confluent if and only if all its critical branchings are confluent.*

Example 195. Consider again the presentation given in Example 186: $P = \langle a, b \mid ba \Rightarrow ab \rangle$. The rewriting system has no critical branchings and the normal forms are words of the form $a^m b^n$ for $m, n \in \mathbb{N}$, which shows that this is indeed a presentation of $\mathbb{N} \times \mathbb{N}$.

Example 196. As a slightly more involved example than the previous one, consider the variation $P = \langle a, b \mid \gamma : ba \Rightarrow ab, \mu : bb \Rightarrow 1 \rangle$. The two critical branchings are confluent:



and the normal forms are words of the form $a^m b^n$ with $m \in \mathbb{N}$ and $n \in \{0, 1\}$, which shows that $\|P\|$ is isomorphic to $\mathbb{N} \times (\mathbb{N}/2\mathbb{N})$.

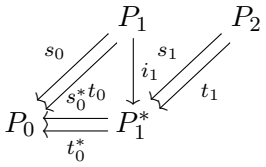
7.1.7 Presentations of categories

If we think of monoids as the particular cases of categories with only one object, and of monoidal categories as the particular cases of 2-categories with only one 0-cell, it is natural to generalize the notion of presentation from monoids to categories.

Definition 197. A *presentation of a category*, also called a *1-dimensional rewriting system* or *categorical rewriting system* or a *2-polygraph*,

$$P \quad = \quad (P_0, s_0, t_0, P_1, s_1, t_1, P_2)$$

consists of three sets P_0, the objects generators, P_1, the morphism generators and P_2, the rewriting rules, together with functions $s_0, t_0 : P_1 \to P_0$, associating with each morphism generator its source and target, and functions $s_1, t_1 : P_2 \to P_1^*$ associating with each rewriting rule its source and target:



such that $s_0^* \circ s_1 = s_0^* \circ t_1$ and $t_0^* \circ s_1 = t_0^* \circ t_1$. The graph $P_0 \underset{t_0}{\overset{s_0}{\leftleftarrows}} P_1$ is often called the *signature* of the presentation, and P_1^* denotes the morphisms of the category it generates, i.e. its paths, $i_1 : P_1 \to P_1^*$ is the canonical injection, sending an edge to the corresponding path of length one and $s_0^*, t_0^* : P_1^* \to P_0$ are the respective canonical extensions of s_0 and t_0, associating with each path its source and target. To be precise, one should also consider the structure of category on $P_0 \underset{t_0^*}{\overset{s_0^*}{\leftleftarrows}} P_1^*$ as part of the data of a 2-polygraph.

Notice that we recover Definition 185 of SRS as the particular case where P_0 is reduced to one element. All the previous developments generalize to this setting with suitable minor modifications: $\|P\|$ is now a presented category, and P^* is now a 2-category with P_2^* as set of 2-cells. This explains why we have denoted the morphisms of P^* with double arrows in various previous diagrams, and we will allow ourselves to either speak of objects/morphisms or 1-cells/2-cells in a monoidal category.

7.2 Polygraphs

7.2.1 Definition

An inductive pattern for the definition of polygraph should be clear from the following remark in previous definition of categorical rewriting system. We define a *1-polygraph* as being an abstract rewriting system, i.e. a graph. Now, notice that the set of "terms" P_1^* in a 2-polygraph P is freely generated by a graph, i.e. a 1-polygraph. This suggests the following inductive definition (imprecisely stated here, see [Bur93] for a proper detailed definition):

Definition 198. A 0-polygraph is a set. Given $n \in \mathbb{N}$, an $(n+1)$-*polygraph* consists of an n-polygraph P, which generates an n-category with P_n^* as set of n-cells, together with a set P_{n+1} of *rewriting rules* and two functions $s_n, t_n : P_{n+1} \to P_n^*$ associating with each rule a source and a target in a "globular way", i.e. such that $s_{n-1}^* \circ s_n = s_{n-1}^* \circ t_n$ and $t_{n-1}^* \circ s_n = t_{n-1}^* \circ t_n$ when $n > 0$.

An $(n+1)$-polygraph P freely generates an $(n+1)$-category P^* and presents an n-category $\|P\|$, obtained from P^* by identifying any two n-cells which are the respective source and target of an $(n+1)$-cell. We will sometimes consider ∞-*polygraphs*, which are obtained as the inductive limit of this construction. For concision, we do not present the morphisms of polygraphs, but those of course do form a reasonable category.

An $(n+1)$-polygraph thus provides us with a notion of *presentation of an n-category*, i.e. an *n-dimensional rewriting system*. The extension of traditional rewriting techniques is the subject of active ongoing work [Bur93, Laf03, GM09, Mim10a, GM12b, GM13, GM12c, GGM15], which is far from being a simple generalization of well-known rewriting techniques. For instance, Lafont was the first to observe that a finite 2-dimensional rewriting system can give rise to an infinite number of critical pairs [Laf03], contrarily to the case of string and term rewriting systems. By elaborating on this observation, Guiraud and Malbos were able to show that a convergent 2-dimensional rewriting system does not necessarily have finite derivation type [GM09] (in the sense of Definition 202 below, generalized to 2-categories), contrarily to the case of presentations of categories recalled in the next section.

Example 199. As an example of a 2-dimensional rewriting system, consider the simplicial category $\triangle$ with natural numbers as objects and increasing functions $f : [m] \to [n]$ as morphisms $f : m \to n$ (this is a variant of the presimplicial category presented in Section 2.1), studied in [Mac98, Laf03]. This category is monoidal, with tensor product being given by addition on objects and by "juxtaposition" on morphisms. It admits a presentation with $P_1 = \{1\}$, $P_2 = \{\mu : 1^{\otimes 2} \Rightarrow 1, \eta : 1^{\otimes 0} \Rightarrow 1\}$ and

$$P_3 = \{ \quad \alpha : \mu \circ (\mu \otimes \mathrm{id}_1) \Rrightarrow \mu \circ (\mathrm{id}_1 \otimes \mu), \quad \lambda : \mu \circ (\eta \otimes \mathrm{id}_1) \Rrightarrow \mathrm{id}_1, \quad \rho : \mu \circ (\mathrm{id}_1 \otimes \eta) \Rrightarrow \mathrm{id}_1 \quad \}$$

(implicitly, we are presenting a 2-category and P_0 is thus reduced to one element). The same methodology as before can be used in order to show that $\|P\| \cong \triangle$, i.e. that P is indeed a presentation of $\triangle$: the critical branchings are confluent and the normal forms are in bijection with morphisms in $\triangle$. From this presentation, many properties can be deduced, for instance the fact that monoidal functors from $\triangle$ to a monoidal category $\mathcal{C}$ correspond to monoids in $\mathcal{C}$.

Remark 200. As noticed by Lafont [Laf03], in order to show that a 2-dimensional rewriting system P is a presentation of a 2-category $\mathcal{C}$, one does not necessarily need to have a convergent rewriting system: it is enough to "invent" a notion of canonical form for morphisms in P_2^*

modulo the congruence generated by P_3, show that every element of P_2^* is equivalent to a canonical form, and that canonical forms are in bijection with 2-cells in $\mathcal{C}$. This is in fact the technique we have been following in Chapter 4 in the proof of Theorem 121, see also [Mim14] for other examples.

Inherent to the definition of polygraphs, is the generation of the free n-category P^* an n-polygraph generates. Sometimes, it is desirable to generate an n-category with more structure, typically with inverses. For instance, a notion of presentation of a groupoid, which includes the widely used notion of group presentation as particular case, can be defined as in Definition 197, except that P_1^* now denotes the morphisms in the free groupoid generated by the underlying graph, i.e. the non-directed paths. For this, and other, reasons, a more general notion of polygraph was introduced by Batanin [Bat98a] which parametrizes the free constructions by a monad on globular sets. We will only need a particular case here: given $n, p \in \mathbb{N}$ with $p \leqslant n$, we call an (n,p)-*polygraph* P a "polygraph" in which the definition has been modified so that cells are invertible in every dimension k with $k > p$. The terminology comes from the fact that the generated n-category is in fact an (n,p)-category, i.e. has k-cells invertible in dimension $k > p$. Similarly, an $(n+1,p)$-polygraph P presents an (n,p)-category $\|P\|$. In particular, an (n,n)-polygraph is an n-polygraph in the previous sense. In order to be clearer, we write $P_k^\top$ instead of P_k^* for the set of freely generated k-cells when the dimension k is supposed to be invertible.

7.2.2 Coherent presentations

In order to obtain more precise invariants of monoids, or categories, one can be interested not only in the rewriting paths, but also in how those rewriting paths relate to each other and how a relation between those can be generated. For this reason, one can be interested in coherent presentations of categories.

Definition 201. A *coherent presentation* of a category $\mathcal{C}$ is a $(3,1)$-polygraph P such that

1. the category presented by the underlying $(2,1)$-polygraph of P is $\mathcal{C}$,
2. in the $(3,1)$-category P^* generated by P there is a 3-cell between any pair of parallel 2-cells or, equivalently, in the $(2,1)$-category $\|P\|$ presented by P there is at most one 2-cell between any pair of 1-cells.

A typical invariant one can compute using this notion is the following one:

Definition 202. Given a $(2,1)$-polygraph P presenting a category $\mathcal{C}$, we say that P has *finite derivation type*, or FDT, when it can be extended as a coherent presentation with a finite number of 3-cells.

This notion is an invariant of the presented category in the sense that if a category $\mathcal{C}$ admits a finite presentation with FDT then any finite presentation of $\mathcal{C}$ has FDT [SOK94], and in this case we simply say that $\mathcal{C}$ has FDT.

Squier has shown that if a category $\mathcal{C}$ admits a convergent presentation P, then it can be completed into a coherent presentation by considering each critical branching, arbitrarily closing it by confluence, and adding a 3-cell with source and target corresponding to this diagram. Since a finite rewriting system has only finitely many critical branchings, the following theorem can be deduced:

Theorem 203 ([SOK94])**.** *A category $\mathcal{C}$ admitting a finite convergent presentation has* FDT.

Example 204. The presentation P of Example 196 for $N \times (\mathbb{N}/2\mathbb{N})$ can be completed into a coherent one by adding as 3-cells

$$P_3 \quad = \quad \{ \quad A : a\mu \circ \gamma b \circ b\gamma \Rrightarrow \mu a, \quad B : \mu b \Rrightarrow b\mu \quad \}$$

This is a fundamental result for the following reason. A finitely presented monoid M which is not FDT but has decidable word problem has also been exhibited by Squier [SOK94, Laf95]: by the previous theorem, this means that there is no hope of finding a convergent presentation for M. Another way to state this is that we cannot always use convergent rewriting to decide the word problem.

Apart from considering finiteness, we will see in next section that more quantitative invariants can also be extracted. For those reasons, we are not interested in knowing whether there exists a coherent presentation (in fact every presentation can always be extended into a coherent one by adding 3-cells of all possible shapes), but rather in constructing ones which are as small as possible. This will be the subject of Chapter 8.

7.2.3 Resolutions and homology

We have seen that by considering the "higher-dimensional structure" of monoids, one can obtain useful invariants on categories. There is no reason to stop at dimension 3, as we would like to briefly mention now. The notion of coherent presentation can be generalized in arbitrary dimension by $(\infty, 1)$-polygraphic resolutions, as defined by Guiraud and Malbos [GM12c], which are a variant of Métayer's polygraphic resolutions [Mét03], which themselves generalize, in the polygraphic context, the well-known notion of projective resolution of a module.

Definition 205 ([Mét03, GM12c])**.** An $(\infty, 1)$-polygraph P is a *resolution* of a category $\mathcal{C}$ if

1. the category presented by the underlying $(2, 1)$-polygraph of P is $\mathcal{C}$,
2. in the $(\infty, 1)$-category P^* generated by P, there is a $(k + 1)$-cell between any pair of parallel k-cells for $k \geqslant 2$.

Given $n \geqslant 2$, a category is FDT_n if it admits a resolution which is finite in every dimension $k \leqslant n$.

Notice that FDT_3 is what we have simply been calling FDT in previous section. Those resolutions are analogue to projective resolutions, in the sense that they can be shown to be cofibrant replacements in the folk model structure on $(\infty, 1)$**-Cat** [LMW10, AM11, GM12c], in which categories generated by $(\infty, 1)$-polygraphs are cofibrant objects. The coherent presentation of a category $\mathcal{C}$ constructed by Squier can in fact be extended into a resolution by having $(n+1)$-generators induced by critical n-uples (in particular, 3-generators are generated by critical branchings, as before), see [GM12c], and the category is thus FDT_∞.

Those properties are of homotopical nature: the polygraphs precisely encode the shapes and boundaries of the cells. However, as is well-known from algebraic topology, by considering properties of homological nature, one can often obtain interesting information (even though it is much less precise than the homotopical one) which is much easier to compute. In this setting, it can be done as follows [Mét03]. With each polygraph P, one can associate a chain complex $\mathbb{Z}P$ of free $\mathbb{Z}$-modules, called its *abelianization*, and defined as

$$\mathbb{Z}P \quad = \quad \cdots \xrightarrow{\partial_2} \mathbb{Z}P_2 \xrightarrow{\partial_1} \mathbb{Z}P_1 \xrightarrow{\partial_0} \mathbb{Z}P_0$$

where the definition of boundary maps ∂_n derives from the source and target maps s_n and t_n of the polygraph P. Given a category $\mathcal{C}$, between any two resolutions P and Q there is a morphism of resolutions $f : P \to Q$, which is unique up to chain homotopy, and therefore their abelianizations are homotopically equivalent, in the usual sense of chain complexes. From it follows that the homology does not depend on the choice of resolution:

Definition 206. Given a category $\mathcal{C}$, its *homology* $H_*(\mathcal{C})$ is defined as $H_*(\mathbb{Z}P)$ where P is any polygraphic resolution of $\mathcal{C}$.

In fact, this definition can be shown to coincide with the traditional definition of homology in the case of monoids [LM09].

Finally, we would like to briefly explain how these homological invariants can be useful. We have mentioned above that, given a convergent presentation P of a category $\mathcal{C}$, we have a resolution of $\mathcal{C}$ where P_0 is the set of objects of $\mathcal{C}$, P_1 gives generators for morphisms of $\mathcal{C}$, P_2 gives the rewriting rules, P_3 the critical branchings, etc. From this we deduce that

— the rank of $H_0(\mathcal{C})$ is a lower bound on the number of object generators of a presentation of $\mathcal{C}$,
— the rank of $H_1(\mathcal{C})$ is a lower bound on the number of morphism generators of a presentation of $\mathcal{C}$,
— the rank of $H_2(\mathcal{C})$ is a lower bound on the number of relations of a presentation of $\mathcal{C}$,
— etc.

In fact, Squier's first result [Squ87] (which was then shown to be implied by Theorem 203) stated that a finite convergent rewriting system has homology groups of finite rank in dimension up to 3, thus enabling him to show that a particular monoid did not have a convergent presentation.

Example 207. Consider the monoid with the following presentation

$$M \quad = \quad \langle a, b, b', c \mid \rho_0 : ab'bbc \Rightarrow 1, \rho_1 : abb'bc \Rightarrow 1, \rho_2 : abbb'c \Rightarrow 1 \rangle$$

which is clearly convergent, and write P for the resolution described above: in particular, we have $P_1 = \{a, b, b', c\}$, $P_2 = \{\rho_0, \rho_1, \rho_2\}$ and $P_3 = \emptyset$ (there are no critical branchings). In the chain complex $\mathbb{Z}P$, the map $\partial_2 : \mathbb{Z}P_3 \to \mathbb{Z}P_2$ is zero because P_3 is empty, and the map $\partial_1 : \mathbb{Z}P_2 \to \mathbb{Z}P_1$ is defined by $\partial_2(\rho_i) = a + 2b + b' + c$ for every index i. Therefore $H_2(M)$ is of rank 2 and we know that any presentation of M necessarily has at least two relations, a fact which would have been difficult to establish directly.

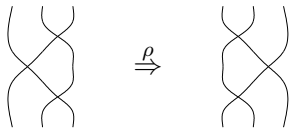
Chapter 8

A homotopical completion procedure

This chapter is mostly based on [GMM+13].

In this chapter we will be interested in the following question: given a presented monoid M, how can we build a reasonably small coherent presentation of it? When starting from a convergent presentation of the monoid, a coherent presentation is easily given by Newman's lemma: it is enough to add a 3-cell corresponding to a confluence diagram for each critical pair. When the rewriting system is not convergent, the Knuth-Bendix completion algorithm [KB70] can be used in many cases in order to complete it into a convergent rewriting system and thus obtain a coherent presentation. Instead of performing this in two steps (first completing the rewriting system and then extracting a convergent presentation), we have proposed an enhanced completion procedure which combines both, which we call a *homotopical completion* algorithm.

Adding generators. The Knuth-Bendix procedure only exploits one kind of transformation in order to complete a rewriting system: given a critical pair $v \overset{f}{\Leftarrow} u \overset{g}{\Rightarrow} w$ it adds a rule $h : v \Rightarrow w$ (or its converse), which is derivable since $h = f^{-1} \circ g$. In particular, adding it does not change the monoid presented by the rewriting system: this is an instance of a Tietze transformation. However, there is another kind of Tietze transformation, adding definable generators, which is rarely exploited by the completion procedure (although some heuristics have been investigated in the context of group theory [Sim94] and term rewriting systems [KB70, Les84]). Could the procedure be improved by also adding new generators during completion? On the theoretical level, an affirmative answer has been brought by Kapur and Narendran [KN85] who considered the usual Artin presentation P of the monoid $\mathbf{B}_3^+$ of positive braids with 3 strands (with its alternative graphical representation on the right):

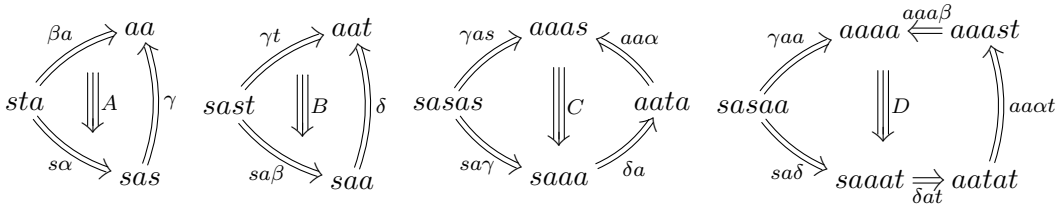
$tst \overset{\rho}{\Rightarrow} sts$  with $s =$  and $t =$ (8.1)

They show that there exists no finite convergent string rewriting system, *with the same generators* s and t, that presents the monoid $\mathbf{B}_3^+$. However, they consider the string rewriting system Υ with three generators s, t and a new generator a (standing for the product st) and

two relations $st \Rightarrow a$ and $sts \Rightarrow tst$. This rewriting system Υ is Tietze-equivalent to the rewriting system P, but applying the Knuth-Bendix completion procedure on it terminates, giving rise to the convergent rewriting system Υ' with s, t, a as generators, and rules

$$ta \overset{\alpha}{\Rightarrow} as \qquad st \overset{\beta}{\Rightarrow} a \qquad sas \overset{\gamma}{\Rightarrow} aa \qquad saa \overset{\delta}{\Rightarrow} aat \tag{8.2}$$

Thus, adding a superfluous generator has made completion possible. The reason why the completed rewriting system Υ' is Tietze-equivalent to the original rewriting system P can be understood by considering its four confluent critical branchings:



(8.3)

The cell $A : (\beta a) \Rrightarrow (s\alpha) \circ \gamma$ witnesses the fact that the rule γ is superfluous since $\gamma = (s\alpha)^{-1} \circ (\beta a)$ and, similarly, the cell B proves that $\delta = (sa\beta)^{-1} \circ (\gamma t)$ is superfluous. Finally, the rule β witnesses the fact that the generator a is superfluous (it is equivalent to st). We are left with the rule α where a has been substituted by st, i.e. $\rho : tst \Rightarrow sts$ in (8.1). As we will see, this example is far from being isolated, thus justifying the use of Tietze transformations as a central concept to study existing extensions and refinements of completion procedures, such as Pedersen's morphocompletion [Ped89], or to introduce new ones.

A homotopical reduction procedure. The additional information contained in coherent presentations can also help one to reduce a presentation by removing superfluous generators, rules and homotopy generators. For instance, we have already mentioned that the cell A in (8.3) indicates that the rule γ is superfluous. Similarly, the rule β indicates that the generator a is superfluous since it is equivalent to the product st, and we will see that superfluous homotopy generators in (8.3) can be also removed by computing critical triples of the rewriting system. All these operations of removing superfluous data from a presentation are again examples of Tietze transformations. Based on these, we introduce here a *homotopical reduction procedure* for coherent presentations which minimizes a coherent presentation, such as one obtained from our homotopical completion procedure. The general idea of this work is thus to give ways to mutate presentations using Tietze transformations in order to come up with presentations satisfying various properties: convergence, coherence, minimality, etc.

Applications in algebra and representation theory. Coherent presentations also appear as a fundamental structure in representation theory (in particular through the examples of Artin and plactic monoids). One of the motivations of the results presented here is to apply constructive rewriting methods to compute coherent presentations for these algebraic structures arising in geometry. For instance, Tits' theorem [Tit81] states that an Artin group has a coherent presentation where coherence cells are given by its parabolic subgroups of rank 3 (a similar result holds for Artin monoids). The original proof relies on geometry, we give here a constructive methodology that has since been used to obtain a coherent presentation for any Artin monoid and group [GGM15]. We also apply our completion methods to the plactic monoid, used in the representation theory of semisimple Lie algebras [Lit96].

Plan of the chapter. We begin by studying in more detail the notion of coherent presentation of a monoid introduced in the previous chapter (Section 8.1), then we introduce our coherent completion and reduction algorithm (Section 8.2), and finally illustrate some of its applications on examples (Section 8.3).

8.1 Coherent presentations of monoids

8.1.1 Definition

As explained in Chapter 7, a presentation of a monoid is a particular instance of a 2-polygraph P, with P_0 being reduced to one element, P_1 being the set of generating letters and P_2 being the set of rules. This presentation freely generates a 2-category (which is also a monoidal category) with P_2^* as set of 2-cells, and also a $(2,1)$-category with $P_2^\top$ as set of 2-cells ($P_2^\top$ is roughly P_2^* with inverses added).

Example 208. Consider the presentation P with $P_1 = \{s, t, a\}$ as generators, and whose rules in P_2 are the four rules of (8.2). The following composite of rewriting steps is a morphism in P_2^*: $(sa\gamma) \circ (\delta a) \circ (aa\alpha) : sasas \Rightarrow aaas$; it occurs in the border of the cell C in (8.3). Similarly, the composite $(s\alpha) \circ \gamma \circ (\beta a)^{-1} : sta \Rightarrow sta$ is a morphism in the groupoid $P_2^\top$, which is the border of the cell A in (8.3).

Here, we will say that a $(3,1)$-polygraph is an *extended presentation*: it consists of a presentation P together with a set P_3 of *homotopy generators* and two functions $s_2, t_2 : P_3 \to P_2^\top$ specifying their source and target. The *homotopy relation* generated by such an extended presentation is the equivalence relation $\equiv_{P_3}$ (or simply $\equiv$) on parallel 2-cells which is stable under context and composition:

— for any f and g in $P_2^\top$ and any u and v in P_1^*, $f \equiv g$ implies $ufv \equiv ugv$,
— for any $h : u' \Rightarrow u$, $f, g : u \Rightarrow v$ and $k : v \Rightarrow v'$ in $P_2^\top$, $f \equiv g$ implies $h \circ f \circ k \equiv h \circ g \circ k$.

Definition 209. A *coherent presentation* is a extended presentation P such that the homotopy relation generated by P_3 relates every pair of parallel 2-cells.

Example 210. The presentation P of Example 208 can be extended into a coherent presentation with the three diagrams A, B and C of (8.3) as its set of 3-cells. For instance, we have $s_2(A) = \beta a$ and $t_2(A) = s\alpha \circ \gamma$. Notice that, since the 2-cells of $P_2^\top$ are invertible, different choices for the source and target of 3-cells could still give a coherent presentation, such as $s_1(A) = (\beta a)^{-1} \circ (s\alpha)$ and $t_1(A) = \gamma^{-1}$.

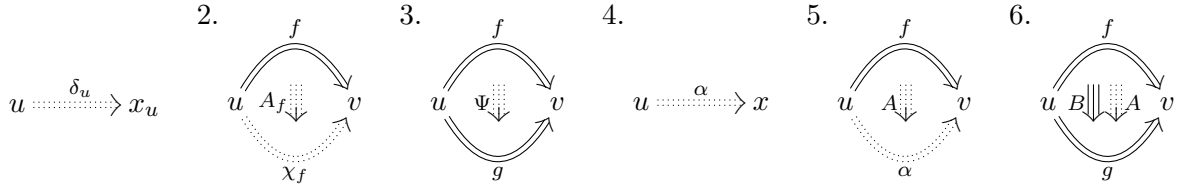
In the same way that rewriting systems present monoids, a coherent presentation presents a 2-category. Namely, given a coherent presentation P, one can define a 2-category, denoted by $P_2^\top / P_3$, as the 2-category $P_2^\top$ whose 2-cells have been quotiented by the homotopy relation $\equiv_{P_3}$. Notice that this 2-category always has its 2-cells invertible.

8.1.2 Transformations of coherent presentations

Starting from a non-convergent presentation of a monoid, the Knuth-Bendix procedure provides a (convergent) presentation on the same set of generators, but a monoid can also admit other presentations with different sets of generators. The notion of Tietze transformation [Tie08], recalled in Section 7.1.2, describes a complete set of elementary transformations

on presentations, leaving the presented monoid unchanged. In [GGM15], a corresponding notion has been introduced for extended presentations, defined as the composites of the following elementary transformations:

1. **add a generator** $\mathcal{T}_u^+$: for u in P_1^*, add x_u to P_1 and $\delta_u : u \Rightarrow x_u$ to P_2,
2. **add a relation** $\mathcal{T}_f^+$: for $f : u \Rightarrow v$ in $P_2^\top$, add $\chi_f : u \Rightarrow v$ to P_2 and $A_f : f \Rrightarrow \chi_f$ to P_3,
3. **add a 3-cell** $\mathcal{T}_{(f,g)}^+$: for $f \equiv_{P_3} g$ in $P_2^\top$, add $\Psi : f \Rrightarrow g$ to P_3,
4. **remove a generator** $\mathcal{T}_x^-$: for $\alpha : u \Rightarrow x$ in P_2, with $x \in P_1$ and $u \in (P_1 \setminus \{x\})^*$, remove x and α and replace x by u in the relations and 3-cells and α by 1_u in the 3-cells,
5. **remove a relation** $\mathcal{T}_\alpha^-$: for $A : f \Rrightarrow \alpha$ in P_3, with $\alpha \in P_2$ and $f \in (P_2 \setminus \{\alpha\})^*$, remove α and A and replace α by f in the 3-cells,
6. **remove a 3-cell** $\mathcal{T}_A^-$: for $A : f \Rrightarrow g$ in P_3 with $f \equiv_{P_3 \setminus \{A\}} g$, remove A.



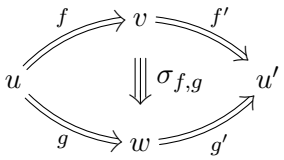
A *Tietze transformation* is a finite composite of elementary Tietze transformations. The notion of Tietze-equivalence on presentations can be generalized to extended presentations: P and Υ are *Tietze-equivalent* extended presentations if they are Tietze-equivalent as presentations and when there is an equivalence of categories $(P_2^\top/P_3) \cong (\Upsilon_2^\top/\Upsilon_3)$, see [GGM15]. In particular, coherent presentations of a same monoid are Tietze-equivalent. As in the case of presentations, we have for extended presentations:

Theorem 211 ([GGM15])**.** *Two finite extended presentations are Tietze-equivalent if, and only if, there exists a Tietze transformation between them.*

8.1.3 Computing coherent presentations

For a string rewriting system, Newman's lemma (see Lemma 191) states that confluence is equivalent to local confluence, which in turn is equivalent to confluence of critical pairs (see Lemma 194). This result can be reformulated as follows in the setting of coherent presentations.

A *family of generating confluences of* P is a set of 3-cells over $P_2^\top$ that contains, for every critical branching (f, g) of P, one 3-cell whose shape is



If P is confluent, it always admits at least one family of generating confluences. Given a convergent presentation P, we denote by $\mathcal{S}(P)$ the extended presentation obtained from P by adjoining a chosen family of generating confluences of P. The presentation $\mathcal{S}(P)$ is only

defined up to that choice, but two families of generating confluences give Tietze-equivalent extended presentations [GM12c]. Squier proved the following result, often called the *Squier (homotopical) theorem*, from which Theorem 203 can in fact be proved.

Theorem 212 ([SOK94])**.** *Let P be a (finite) convergent presentation of a monoid $\mathbf{M}$. The extended presentation $\mathcal{S}(P)$ is a (finite) coherent and convergent presentation of $\mathbf{M}$.*

Several examples of this construction are given in [Laf95]. Squier proved that the property, for a finite presentation of a monoid $\mathbf{M}$, to be extendable into a finite coherent presentation is an invariant of $\mathbf{M}$, that is, one given finite presentation of $\mathbf{M}$ is extendable into a finite coherent presentation if, and only if, all of them are [SOK94]. However, there are finitely presented decidable monoids with no finite coherent presentation (such an example was exhibited by Squier). For such a monoid, starting with a finite presentation, there is no hope to obtain a finite convergent presentation, by using the Knuth-Bendix procedure or other methods, with the same set of generators or another one. Conversely, if the Knuth-Bendix procedure terminates on a finite presentation, then it can be extended into a finite coherent presentation.

8.2 Homotopical completion and reduction procedures

As seen in Section 8.1.3, Squier's theorem extends a convergent presentation into a coherent one. With the Knuth-Bendix completion procedure, those are the two basic ingredients of the homotopical completion procedure we present, extended by a homotopical reduction procedure whose goal is to eliminate superfluous cells.

8.2.1 The homotopical completion procedure

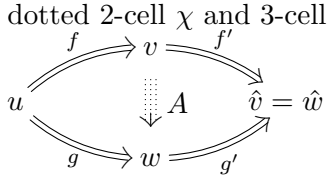
This procedure, denoted by $\mathcal{HC}$, interleaves the Knuth-Bendix completion and Squier's theorem to produce a coherent and convergent presentation from a terminating presentation: it examines the critical branchings one by one, potentially adding 2-cells to reach a convergent presentation, but also 3-cells that tend towards forming a coherent presentation.

Let P be a terminating presentation, seen as an extended presentation with no 3-cell. Thereafter, we always assume that termination is due to a fixed total termination order. For every critical branching $(f, g) : u \Rightarrow (v, w)$ of P, the procedure $\mathcal{HC}$ computes 2-cells $f' : v \Rightarrow \hat{v}$ and $g' : w \Rightarrow \hat{w}$ in P_2^*, where $\hat{v}$ and $\hat{w}$ are some normal forms for v and w, respectively. There are two possibilities:

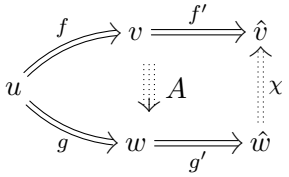
— if $\hat{v} = \hat{w}$, the dotted 3-cell A is added, as in situation (a),

— otherwise, for example if $\hat{v} < \hat{w}$, the Tietze transformation $\mathcal{T}^{+}_{g'^{-1} \circ g^{-1} \circ f \circ f'}$ is applied to add the dotted 2-cell χ and 3-cell A, as in (b).

(a)



(b)



The adjunction of new 2-cells can create new critical branchings: the procedure $\mathcal{HC}$ iterates this operation until it reaches, potentially after an infinite time, a stable extended presentation $\mathcal{HC}(P)$. From a computational point of view, an application of Squier's theorem to the result of the Knuth-Bendix completion on P would require one to compute again all the critical branchings explored during completion, while the $\mathcal{HC}$ procedure computes 3-cells

during completion. The properties of the Knuth-Bendix procedure and Squier's theorem induce the following result.

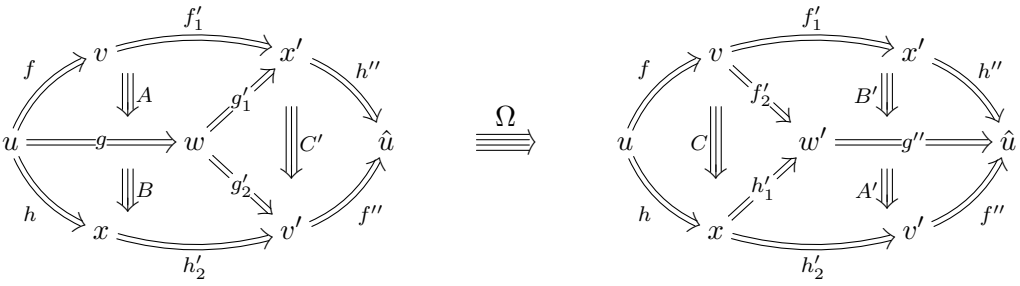
Theorem 213 ([GMM+13])**.** *Let P be a terminating presentation of a monoid $\mathbf{M}$. The extended presentation $\mathcal{HC}(P)$ is a coherent and convergent presentation of $\mathbf{M}$ and it is finite if, and only if, the presentation P is finite and the homotopical completion procedure terminates.*

Example 214. The Kapur-Narendran presentation $\mathbf{B}_3^+ = \langle s, t, a \mid \alpha : ta \Rightarrow as,\ \beta : st \Rightarrow a \rangle$ has two non-confluent critical branchings, resulting in the adjunction of the 2-cells γ and δ as in (8.2) and the 3-cells A and B as in (8.3). The 2-cells γ and δ generate two new critical branchings that are confluent: the $\mathcal{HC}$ procedure adds two extra 3-cells C and D and terminates with this finite coherent and convergent presentation of the monoid $\mathbf{B}_3^+$.

8.2.2 An optimized homotopical completion procedure

The procedure $\mathcal{HC}$ computes a coherent and convergent presentation that contains, in general, superfluous 3-cells, in the sense that they are not necessary to relate all the parallel 2-cells. To eliminate them, we apply a *homotopical reduction* mechanism in dimension 3: it computes the critical triple branchings to produce relations between 3-cells and to eliminate some of them by Tietze transformations. A *critical triple branching* (f, g, h) is a triple of distinct rewriting steps with common source, such that each one overlaps with at least one of the other two, and that is minimal for the order $\preccurlyeq$ generated by relations $(f, g, h) \preccurlyeq (ufv, ugv, uhv)$ for every such triples (f, g, h) and words u, v.

Let P be a convergent and coherent presentation. The *homotopical reduction in dimension* 3 builds, for each critical triple branching (f, g, h) of P, a 4-cell Ω with shape

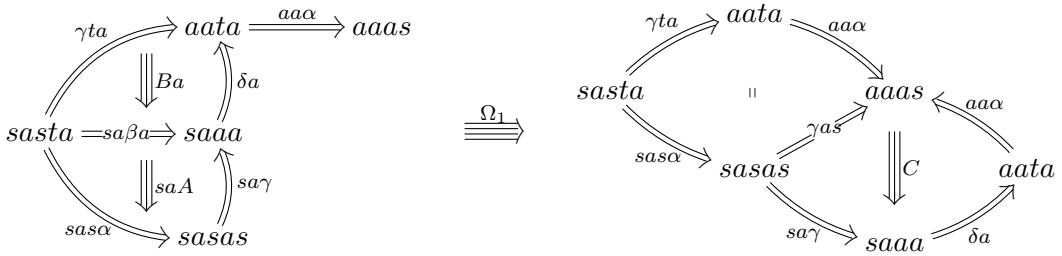


as follows. We consider the branching (f, g) and use confluence to get f'_1 and g'_1 and, then, coherence to build the 3-cell A. We proceed similarly with the branchings (g, h) and (f, h). Then, for the branching (f'_1, f'_2), we use convergence to get g'' and h'' with $\hat{u}$ as common target, and the 3-cell B' by coherence. We do the same operation with (h'_1, h'_2) to get A'. Finally, we get the 3-cell C' by coherence. The source and the target of Ω are made of generating 3-cells of P in context: they have shape uXv where X is a generating 3-cell and u and v are (identities on) words. If one of those generating 3-cells appears only once and in an empty context ($u = v = 1$), then Ω is used as a definition of X in terms of the other 3-cells: X is removed by a Tietze transformation.

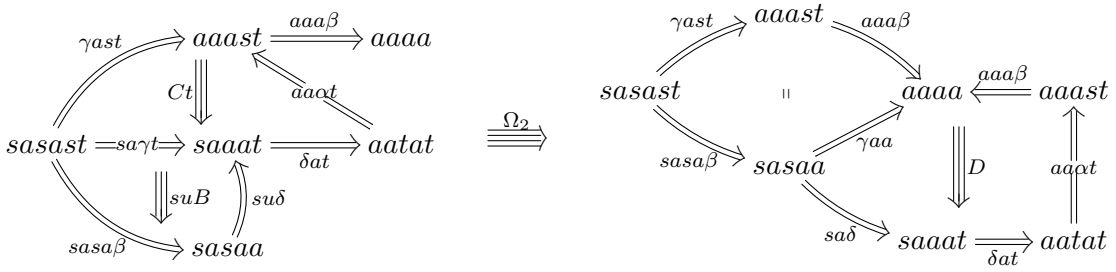
A coherent and convergent presentation on which no 3-cell can be removed by homotopical reduction in dimension 3 is called *reduced*. The *optimized homotopical completion procedure* $\overline{\mathcal{HC}}$ applies homotopical reduction in dimension 3 after $\mathcal{HC}$. Since the procedure acts by Tietze transformations only, we get:

Theorem 215 ([GMM+13])**.** *Let P be a terminating presentation of a monoid $\mathbf{M}$. The extended presentation $\overline{\mathcal{HC}}(P)$ is a reduced coherent and convergent presentation of $\mathbf{M}$, that is finite if, and only if, the presentation P is finite and the homotopical completion procedure $\mathcal{HC}$ terminates.*

Example 216. After the procedure $\mathcal{HC}$ is applied to the Kapur-Narendran presentation of the monoid $\mathbf{B}_3^+$, we have four critical triple branchings, overlapping on the words *sasta*, *sasast*, *sasasas* and *sasasaa*. On *sasta*, we get the 4-cell



This 4-cell proves that C is superfluous in the coherent presentation: it appears only once in the boundary of Ω_1, in an empty context (unlike A and B). Then, we consider the critical triple branching with source *sasast*:



For the same reasons, the 4 cell Ω_2 allows us to remove D, leaving only the 3-cells A and B to form a reduced coherent and convergent presentation of the monoid $\mathbf{B}_3^+$. The other two critical triple branchings on words *sasasas* and *sasasaa* do not generate any other relation. Indeed, since the relations are weight-homogeneous (they relate words with the same weight, where s and t have weight 1 and a has weight 2), all the words that occur in the 4-cells corresponding to *sasasas* and *sasasaa* have weight 10 and 11, respectively. Since A and B have respective weights 4 and 5, their potential occurrences in those 4-cells must be in non-empty contexts.

8.2.3 The homotopical completion-reduction procedure

After the procedure $\overline{\mathcal{HC}}$, we get a reduced coherent and convergent presentation of the considered monoid. Its underlying presentation is, in general, not minimal since homotopical completion has potentially adjoined superfluous 2-cells to get confluence. However, for each of those extra 2-cells, a 3-cell has also been added to fill the corresponding confluence diagram: a Tietze transformation can be used to remove both of them.

Given a coherent presentation P, we call *homotopical reduction in dimension* 2 the following process. For each 3-cell A of P_3, its source and target are made of reduction steps $u\alpha v$, where α is a generating 2-cell and u and v are words. If one such α appears only once and

in an empty context ($u = v = 1$), both α and A are removed by a Tietze transformation. On the special case of $\overline{\mathcal{HC}}(P)$, every superfluous 2-cell appears once and in an empty context in the boundary of its associated 3-cell. The *homotopical completion-reduction procedure* $\mathcal{HCR}$ applies homotopical reduction in dimension 2 after $\overline{\mathcal{HC}}$. Since the procedure acts by Tietze transformations only, we get:

Theorem 217 ([GMM+13])**.** *Let P be a terminating presentation of a monoid $\mathbf{M}$. The extended presentation $\mathcal{HCR}(P)$ is a coherent presentation of $\mathbf{M}$, whose underlying presentation is contained in P, and it is finite if, and only if, the homotopical completion procedure terminates.*

Example 218. After the procedure $\overline{\mathcal{HC}}$ is applied to the Kapur-Narendran presentation of the monoid $\mathbf{B}_3^+$, we have a coherent presentation with three generators s, t and a, four 2-cells α, β, γ and δ and two 3-cells A and B, corresponding to the adjunction of γ and δ respectively. They are removed by the $\mathcal{HCR}$ procedure, yielding a coherent presentation of $\mathbf{B}_3^+$ with the 2-cells α and β only, and no 3-cell. Informally, for two words on $\{s, t, a\}$ that represent the same element in the monoid $\mathbf{B}_3^+$, there is only one proof of their equality modulo α and β.

8.2.4 Completion and reduction on generators

As proved by Kapur and Narendran [KN85], and recalled at the beginning of this chapter, the introduction of superfluous generators can be necessary for completion to terminate. These generators can of course be added by hand before the completion, but we briefly indicate here a possible heuristic, based on algebraic properties observed on the examples in Section 8.3. Indeed, in those cases, it always helps completion to add generators of the *quasicenter* of each submonoid. More precisely, for a given presentation P of a monoid $\mathbf{M}$, we seek minimal elements u of P_1^* such that $uX = Xu$ holds in $\mathbf{M}$ for some subset X of P_1, maximal with this property. Such a property is possible to observe during completion: one computes the products ux and xu for u a word of bounded length and x a generator. If $uX = Xu$ for a set X of generators, one adds a new generator (u) and a relation $u \Rightarrow (u)$. Moreover, the cardinality of X seems to determine a way to extend to (u) the termination order used for completion (see 8.3.3).

Whether the generators have been added before or during homotopical completion, one can remove them at the end. Indeed, each superfluous generator (u) comes with a defining relation $\alpha : u \Rightarrow (u)$, so that a Tietze transformation removes both of them (and replaces (u) by u and α by the identity in the boundary of the other 2-cells and 3-cells). Applied to the result of the $\mathcal{HCR}$ completion on the Kapur-Narendran presentation of the monoid $\mathbf{B}_3^+$, this contracts the obtained coherent presentation (with no 3-cell) to the Artin presentation of the monoid $\mathbf{B}_3^+$, proving that this is also a coherent presentation with no 3-cell.

8.3 Applications

The results mentioned in this section were obtained with the help of a prototype implementation; an online version (unfortunately much slower than the offline one) is available at `http://www.lix.polytechnique.fr/Labo/Samuel.Mimram/rewr/`.

8.3.1 The braid monoid

The Artin presentation. The *monoid* $\mathbf{B}_n^+$ *of positive braids on* n *strands* is defined by

$$\mathbf{B}_n^+ = \langle s_1, \ldots, s_{n-1} \mid s_i s_{i+1} s_i = s_{i+1} s_i s_{i+1} \text{ for } 1 \leqslant i < n-1,\ s_i s_j = s_j s_i \text{ for } |i-j| \geqslant 2 \rangle \tag{8.4}$$

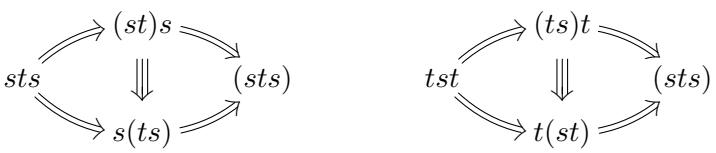
This presentation, called *Artin presentation*, is known to be minimal, so that one wants to compute a minimal coherent presentation of the monoid $\mathbf{B}_n^+$ by extending it. In [Tit81], Tits proved a result that implies that a coherent presentation is given by 3-cells whose boundaries are in one of the Artin submonoids of rank 3 of $\mathbf{B}_n^+$, i.e. the boundary of each 3-cell is made of copies of the three relations involving only three given distinct generators. As a direct consequence, the Artin presentation with no 3-cell is a coherent presentation of the monoid $\mathbf{B}_3^+$, but this result fails to say anything about $\mathbf{B}_4^+$ and does not give an explicit description of the coherence cells of $\mathbf{B}_n^+$ for $n \geqslant 5$. Unfortunately, the homotopical completion cannot be used either in practice because it does not terminate on the Artin presentation: indeed, as proved by Kapur and Narendran, any orientation of a relation $sts = tst$ generates a relation $stsst^k = ts^{k+1}ts$ for every $k \geqslant 1$ that must be contained in every convergent presentation [KN85].

The Kapur-Narendran presentation. As far as we know, the adjunction of the superfluous generator for $\mathbf{B}_3^+$, as seen in the introduction, has not been studied for $\mathbf{B}_n^+$ with $n > 3$. There are several possible generalizations, but we define the *Kapur-Narendran presentation* of $\mathbf{B}_n^+$ as the one obtained from Artin presentation by adjunction of superfluous generators corresponding to a Coxeter element for each Artin submonoid of $\mathbf{B}_n^+$, namely all the products $s_{i_1} \cdots s_{i_k}$ for every $1 \leqslant i_1 < \cdots < i_k < n$. Our experiments lead to positive results for the cases $n = 4$ and $n = 5$, see Table 8.1. We have also tested the Kapur-Narendran presentation on well-known generalizations of the braid monoids known as Artin monoids and got a finite coherent and convergent presentation for the Artin monoids of types B_2, B_3, B_4 and F_4 (the braid monoid $\mathbf{B}_n^+$ is the Artin monoid of type A_{n-1}). An open question is to determine if the Kapur-Narendran presentation yields a finite coherent and convergent presentation for any braid monoid and, more generally, for other types of Artin monoids.

The Garside presentation. The Kapur-Narendran presentation is contained in a bigger presentation called the *Garside presentation* [Gar69]. For $\mathbf{B}_3^+$, the Garside presentation is obtained from the Artin one by adjunction of superfluous generators (st), (ts) and (sts) corresponding to products st, ts and sts respectively: the element sts is the generator of the quasicenter of $\mathbf{B}_3^+$ and the elements s, t, st and ts are all its divisors. On the Garside presentation, the homotopical completion procedure produces a finite coherent and convergent presentation with five generators, twelve relations and 24 3-cells; the corresponding normal forms are known as *Deligne's normal forms* [Del72]. Deligne has proved in [Del97] that this coherent presentation of $\mathbf{B}_3^+$ can be reduced to one with six relations

$$st \Rightarrow (st) \qquad ts \Rightarrow (ts) \qquad s(ts) \Rightarrow (sts) \qquad t(st) \Rightarrow (sts) \qquad (st)s \Rightarrow (sts) \qquad (ts)t \Rightarrow (sts)$$

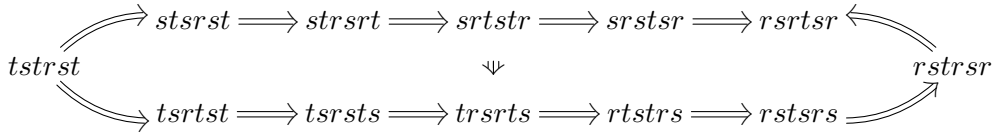
and two 3-cells:



The homotopical completion-reduction, applied to the Garside presentation, gives a new,

constructive proof of this result [GGM15]. In fact, it goes even further: the 3-cells are used to remove the relations $(st)s \Rightarrow (sts)$ and $(ts)t \Rightarrow (sts)$ and, then, the relations $st \Rightarrow (st)$, $ts \Rightarrow (ts)$ and $s(ts) \Rightarrow (sts)$ remove the generators (st), (ts) and (sts). This leaves the generators s and t, the relation $t(st) \Rightarrow (sts)$, projected onto $tst \Rightarrow sts$, and no coherence cell, yielding another proof that Artin presentation with no 3-cell is a coherent presentation of $\mathbf{B}_3^+$.

The Garside presentation exists for every monoid $\mathbf{B}_n^+$ and, more generally, for every Artin monoid: in the spherical case (such as $\mathbf{B}_n^+$), its generators are made of the generator of the quasicenter of every Artin submonoid, plus all of its divisors. On this presentation, the homotopical completion-reduction procedure also applies, extending Deligne's result to non-spherical Artin monoids. Moreover, in [GGM15], Gaussent, Guiraud and Malbos apply homotopical reduction further to get an explicit coherent presentation of every Artin monoid, thus, in particular, giving a constructive proof of Tits's result. For the particular case of $\mathbf{B}_4^+$, the homotopical completion-reduction procedure gives a (minimal) coherent presentation made of Artin presentation with exactly one coherence cell, known as the *Zamolodchikov relation*:



The Brieskorn-Saito presentation. For the monoid $\mathbf{B}_3^+$, it is defined by the adjunction to Artin presentation of a generator (sts) for sts [BS72]. This presentation is known in general for Artin monoids and obtained by adjunction of the generator (when it exists) of the quasicenter of each Artin submonoid. Those generators produce special normal forms that, up to our knowledge, are not yet linked to a convergent presentation. Contrarily to Garside's generators, Brieskorn-Saito's generators come in a finite number for every Artin monoid, motivating the search for a finite convergent presentation on those generators to give a solution to the still-open word problem for general Artin groups. Our experiments show that, on the Brieskorn-Saito presentation, the homotopical completion procedure gives a finite coherent and convergent presentation for the monoids $\mathbf{B}_3^+$, $\mathbf{B}_4^+$ and $\mathbf{B}_5^+$, but also for other Artin monoids such as the ones of type B_3 and, interestingly, of type $\tilde{A}_2$: this last example is an Artin monoid of *affine* type, for which the Garside presentation is infinite.

8.3.2 The plactic monoid

The Knuth presentation. The *plactic monoid* $\mathbf{P}_n$ *of rank* n is given by the Knuth presentation:

$$\mathbf{P}_n = \langle x_1, \dots, x_n \mid x_j x_i x_k = x_j x_k x_i \text{ for } i < j \leqslant k \text{ and } x_i x_k x_j = x_k x_i x_j \text{ for } i \leqslant j < k \rangle.$$

This monoid originates in the work of Schensted [Sch61], Knuth [Knu70], Lascoux and Schützenberger [LS81]. It has found several applications, such as in representation theory [Lit96] because of its strong connection to Young tableaux: semistandard Young tableaux correspond to elements of the plactic monoid and Schensted's insertion algorithm gives a way to compute normal forms for the Knuth presentation of the plactic monoid. In the case $n = 2$, the Knuth presentation has two generators $x_1 = a$ and $x_2 = b$ and two relations $\alpha : baa \Rightarrow aba$

and $\beta : bba \Rightarrow bab$. This terminating presentation (for the deglex order generated by $a < b$ for example) is already convergent: the homotopical completion procedure yields a homotopy basis with exactly one coherence cell $\beta a \Rrightarrow b\alpha : bbaa \Rightarrow baba$. Moreover, the convergent presentation has no critical triple branching: hence the computed coherent presentation of the monoid $\mathbf{P}_2$ is minimal.

In the case $n = 3$, with generators a, b and c, the Knuth presentation has eight relations, three pairs corresponding to the three plactic submonoids over two of the three generators, plus two relations involving all three generators: $\gamma : cab \Rightarrow acb$ and $\delta : bca \Rightarrow bac$. For the monoid $\mathbf{P}_3$, the Knuth presentation is not confluent anymore (on words $cbba$, $ccbaa$, $ccbab$) and homotopical completion adds three more relations: $\varepsilon : cbab \Rightarrow bcba$, $\varphi : cbaba \Rightarrow cacba$ and $\psi : cbcba \Rightarrow cbacb$. At the end of the process, we get a finite coherent and convergent presentation with 27 3-cells, corresponding to all the critical branchings. The presentation also has 29 triple critical branchings, and homotopical reduction uses four of them to eliminate four 3-cells. Then, the removal of the three extra relations and their corresponding coherence cells, added by completion, yields a homotopy basis with 20 coherence cells for the Knuth presentation of the monoid $\mathbf{P}_3$.

For higher values of n, the homotopical completion procedure cannot succeed on the Knuth presentation. Indeed, as in the case of braid monoids, a proof similar to the one of Kapur and Narendran for the monoid $\mathbf{B}_3^+$ shows that the infinite family of relations $cbc^kdca = cbac^kdc$, for every natural number k, must be part of any convergent presentation of the monoid $\mathbf{P}_n$.

The column presentation. The analogy with Young tableaux leads one to introduce a finite number of superfluous generators to the Knuth presentation of $\mathbf{P}_n$, representing all the possible columns in semistandard Young tableaux: one generator $(x_{i_k} \cdots x_{i_1})$ for every possible $1 < k \leqslant n$ and $1 \leqslant i_1 < \cdots < i_k \leqslant n$, together with the corresponding defining relation $x_{i_k} \cdots x_{i_1} \Rightarrow (x_{i_k} \cdots x_{i_1})$. The column generators have an important property in plactic monoids: indeed, the center (and the quasicenter) of the plactic monoid $\mathbf{P}_n$ is generated by exactly one element: $x_n \cdots x_1$. Thus the column generators for $\mathbf{P}_n$ are exactly the generators of the quasicenters of all the plactic submonoids of $\mathbf{P}_n$.

From the column presentation, homotopical completion yields a finite coherent and convergent presentation of $\mathbf{P}_n$ (as in [CGM12, Hag14]). In particular, for the monoid $\mathbf{P}_4$, we get the following construction. Starting with the Knuth presentation with four generators and 20 relations, we add the eleven column generators (ba, ca, cb, da, db, dc, cba, dba, dca, dcb, $dcba$) and the corresponding relations to get 15 generators and 31 relations. Homotopical completion results in a finite coherent and convergent presentation with 115 relations and 621 3-cells. Then, homotopical reduction in dimension 3 uses the triple critical branchings to reduce the number of 3-cells to 212. The removal of the 84 relations and 3-cells added during homotopical completion, then of the eleven superfluous generators and their defining relations, finally produces a coherent presentation made of the Knuth presentation of the monoid $\mathbf{P}_4$ and 128 3-cells.

8.3.3 The Chinese monoid

The standard presentation. The *Chinese monoid* $\mathbf{Ch}_n$ *of rank* n is defined by

$$\mathbf{Ch}_n = \langle x_1, \ldots, x_n \mid x_j x_k x_i = x_k x_i x_j = x_k x_j x_i \text{ for } i \leqslant j \leqslant k \rangle.$$

It is a variant of the plactic monoid discovered in [DK94]. For $n = 2$, the Chinese monoid coincides with the plactic monoid $\mathbf{P}_2$: its standard presentation (with the orientation $baa \Rightarrow aba$ and $bba \Rightarrow bab$) is convergent and, with the same 3-cell as $\mathbf{P}_2$, forms a coherent presentation of $\mathbf{Ch}_2$. For higher values of n, the presentation of $\mathbf{Ch}_n$ (with the orientation $x_k x_i x_j \Rightarrow x_j x_k x_i$ and $x_k x_j x_i \Rightarrow x_j x_k x_i$) is not convergent anymore, but it can be finitely completed (without change of generators) by adjunction of the relations $x_k x_j x_k x_i \Rightarrow x_k x_i x_k x_j$ for $1 \leqslant i < j < k \leqslant n$. The homotopical completion-reduction yields a coherent presentation made of the standard presentation extended with 12 3-cells for $\mathbf{Ch}_3$, 56 for $\mathbf{Ch}_4$ and 176 for $\mathbf{Ch}_5$.

The quasicentral presentation. The (quasi)center of the monoid $\mathbf{Ch}_n$ is generated by the element $x_n x_1$ [CEH$^+$01]. Thus, the generators of the quasicenters of all the Chinese submonoids of $\mathbf{Ch}_n$ are exactly the elements $x_j x_i$ for $1 \leqslant i < j \leqslant n$. Our experiments, conducted up to $n = 5$, show that the adjunction of those elements as superfluous generators still allow completion to reach a finite convergent presentation. Moreover, the obtained finite convergent presentation gives a rewriting-based procedure to compute the *column normal form* [CEH$^+$01]. For the completion, a special order has to be chosen (corresponding to the column normal form), such as a weight lexicographic order, where each x_i has weight 1 and $(x_j x_i)$ has weight 2, and with an order on generators that satisfies $(x_l x_i) > (x_k x_j)$ if $i \leqslant j \leqslant k \leqslant l$ with $i \neq j$ or $k \neq l$. This last inequality can be determined automatically from the fact that $(x_k x_i)$ commutes with $l - i$ elements and $(x_k x_j)$ with $k - j$ and, by assumption, we have $l - i > k - j$.

8.4 Future work

These homotopical completion procedures have been implemented in some proof-of-concept piece of software, and much work remains to be done in order to understand better the structures put to use and how to efficiently manipulate them. The idea of adding superfluous generators seems very promising, but we have only been able to provide heuristics to do so which have to be refined and supported by more experiments. Finally, the approach developed here handles generators, relations and homotopy generators uniformly; its likely extension to higher dimensions will be investigated in future work, in relation with methods for constructing minimal presentations of algebraic structures. A first step in this direction would be to generalize our method to the case of Lawvere theories which are to term rewriting systems what monoids are to string rewriting systems.

Also, it would be interesting to investigate the use of refined techniques for showing confluence. For instance, Yudin [Yud15] has recently started investigating possible extensions of the work presented here for non-terminating rewriting systems, using van Oostrom's decreasing diagrams [Oos94].

Coherent presentations						
Monoid	Presentation	Gen.	Rel.	Rel. comp.	Hom. gen.	Hom. gen. red.
$\mathbf{B}_3^+$	Artin	2	1	$\infty^\dagger$	$\infty^\dagger$	$0^\dagger$
	Kapur-Narendran	3	2	4	4	2
	Brieskorn-Saito	3	2	4	6	2
	Garside	5	4	12	24	8
$\mathbf{B}_4^+$	Artin	3	3	$\infty^\dagger$	$\infty^\dagger$	$1^\dagger$
	Kapur-Narendran	7	7	47	356	31
	Brieskorn-Saito	7	7	46	378	35
$\mathbf{B}_5^+$	Artin	4	6	$\infty^\dagger$	$\infty^\dagger$	$4^\dagger$
	Kapur-Narendran	15	17	692	48260	?
	Brieskorn-Saito	15	17	598	28384	?
$\mathbf{P}_2 = \mathbf{Ch}_2$	Knuth	2	2	2	1	1
	Column	3	3	3	1	1
$\mathbf{P}_3$	Knuth	3	8	11	27	23
	Column	7	12	22	42	30
$\mathbf{P}_4$	Knuth	4	20	$\infty^\dagger$	$\infty^\dagger$	$?^\dagger$
	Column	15	31	115	621	212
$\mathbf{P}_5$	Column	31	66	531	6893	?

Table 8.1 – Results of experiments indicating, for various sets of generators, the number of generators, relations (before and after completion), and homotopy generators (before and after homotopy reduction by 4-cells) of the completed rewriting system. Values marked "†" arise from theoretical computations, and "?" indicate computations too big to be performed in reasonable time with our prototype implementation.

Chapter 9

Presentations modulo of categories

This chapter is mostly based on [CM15].

The setting of higher-dimensional rewriting systems (or polygraphs) recalled in Chapter 7 is very useful to provide small descriptions of n-categories, from which many computations can be performed in order to compute invariants of the presented category. However, there is an inherent limitation to the n-categories which can be described in this way, in higher-dimensions: any category admits a presentation (see Lemma 187), but this is not true for n-categories with $n > 1$, as we now explain. Given an $(n+1)$-polygraph P, the presented n-category is $\|P\| = P_n^*/P_{n+1}$ is obtained from the n-category freely generated by the underlying n-polygraph by quotienting n-cells by the congruence generated by elements of P_{n+1}. Thus, no quotient is performed on k-cells for $0 \leqslant k < n$ and the underlying $(n-1)$-category of $\|P\|$ is free. From this point of view, this is a limitation of polygraphs: an n-category whose underlying $(n-1)$-category is not free cannot be presented by a polygraph.

This can be better understood by trying to present the monoidal category (i.e. 2-category with only one 0-cell) $\triangle \times \triangle$ with tensor product extending componentwise the one of $\triangle$, the simplicial category: the underlying monoid of objects is $\mathbb{N} \times \mathbb{N}$, which is not a free monoid. Recall from Example 199 that the monoidal category $\triangle$ admits a presentation P with one 1-generator $P_1 = \{a\}$, two 2-generators $P_2 = \{\mu : aa \Rightarrow a, \eta : 1 \Rightarrow a\}$ and three relations $P_3 = \{\alpha, \lambda, \rho\}$ respectively stating that the multiplication μ is associative and admits η as left and right unit. If we try to construct a presentation for $\triangle \times \triangle$, seen as consisting of "two copies" of the above category $\triangle$, we are led to consider a presentation containing "two copies" of the previous presentation: we consider a presentation P with $P_1 = \{a, b\}$ as 1-generators (where a and b respectively correspond to the objects $(1,0)$ and $(0,1)$), with $P_2 = \{\mu_a, \eta_a, \mu_b, \eta_b\}$ as 2-generators (with $\mu_a : a \otimes a \Rightarrow a$, $\mu_b : b \otimes b \Rightarrow b$, etc.), and with $P_3 = \{\alpha_a, \lambda_a, \rho_a, \alpha_b, \lambda_b, \rho_b\}$ as relations. If we stop here adding relations, the presented category has $\{a, b\}^*$ as underlying monoid of objects, which is not right: recalling the presentation for $\mathbb{N} \times \mathbb{N}$ seen in Example 186, we should moreover add a relation $\gamma : ba = ab$. However, such a relation between 1-generators is not allowed in the usual notion of presentation (only relations in guise of 2-cells are allowed). In order to provide a meaning to this, three constructions are available: restrict P to some canonical representatives of objects modulo the equivalence generated by γ (typically the words of the form $a^m b^n$), quotient the monoidal category $\|P\|$ presented by P by γ, or formally invert the morphism γ in $\|P\|$. We show that under reasonable assumptions on the presentation, all the three constructions coincide, thus providing one with a notion of *coherent* presentation modulo a relation such as γ. As a first step toward this situation, we study here the simpler case of presentations of categories and

introduce a notion of presentation modulo for those, leaving the case of 2-categories for future work.

This work can somehow be seen as an extension of traditional techniques of rewriting modulo an equational theory [BN99], in the case where the equational theory can itself be oriented as a convergent rewriting system, called an equational rewriting system.

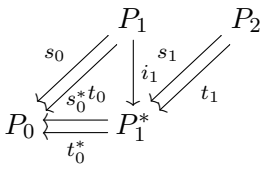
We begin by introducing the notion of presentation of a category modulo a rewriting system (Section 9.1) and study some of its properties when the rewriting system is convergent (Section 9.2) in order to show that, under suitable assumptions, the three natural constructions for the presented category coincide (Section 9.3). We finally discuss some extensions of this work (Section 9.4).

9.1 Presentations of categories modulo a rewriting system

9.1.1 Presentations of categories

Recall from Chapter 7 the definition of a presentation for a category:

Definition 219. A *presentation* $P = (P_0, s_0, t_0, P_1, s_1, t_1, P_2)$:



consists of a graph (P_0, s_0, t_0, P_1), the elements of P_0 (resp. P_1) being called *object* (resp. *morphism*) *generators*, together with a set P_2 of *relations* (or *2-generators*) and two functions $s_1, t_1 : P_2 \to P_1^*$ such that $s_0^* \circ s_1 = s_0^* \circ t_1$ and $t_0^* \circ s_1 = t_0^* \circ t_1$. The category $\|P\|$ *presented* by P is the category obtained from the category generated by the graph (P_0, s_0, t_0, P_1) by quotienting morphisms by the smallest congruence wrt composition identifying any two morphisms f and g such that there exists $\alpha \in P_2$ satisfying $s_1(\alpha) = f$ and $t_1(\alpha) = g$.

In the following, we often simply write (P_0, P_1, P_2) for a presentation as above, leaving the source and target maps implicit. We write $f : x \to y$ for an edge $f \in P_1$ with $s_0(f) = x$ and $t_0(f) = y$, and $\alpha : f \Rightarrow g$ for a relation with f as source and g as target. We sometimes write $\alpha : f \Leftrightarrow g$ to indicate that $\alpha : f \Rightarrow g$ or $\alpha : g \Rightarrow f$ is an element of P_2, and we denote by $\overset{*}{\Leftrightarrow}$ the smallest congruence such that $f \overset{*}{\Leftrightarrow} g$ whenever there exists $\alpha : f \Rightarrow g$ in P_2 ($\overset{*}{\Leftrightarrow}$ is also the smallest congruence wrt composition generated by $\Leftrightarrow$). More precisely, a presentation P generates a $(2,1)$-category (i.e. a 2-category with invertible 2-cells), the category presented by P is obtained from this 2-category by identifying 1-cells when there is a 2-cell in between [Bur93, Laf03], and we write $\alpha : f \overset{*}{\Leftrightarrow} g$ for such a 2-cell.

9.1.2 Presentations modulo

In a presentation P of a category, relations are generated by elements of P_2: the morphisms of the free category on the underlying graph will be quotiented by those in order to obtain the presented category. We now extend this notion in order to also allow for quotienting objects in the process of constructing the presented category.

Definition 220 ([CM15])**.** A *presentation modulo* $(P, \tilde{P}_1)$ consists of a presentation of category $P = (P_0, P_1, P_2)$ together with a set $\tilde{P}_1 \subseteq P_1$, whose elements are called *equational generators*.

The morphisms of $\|P\|$ generated by the equational generators are called *equational morphisms*. Intuitively, the category presented by a presentation modulo should be the "quotient category" $\|P\|/\tilde{P}_1$, as explained in the next section, where objects equivalent under $\tilde{P}_1$ (i.e. related by equational morphisms) are identified. We believe that the reason why presentations modulo of categories were not introduced before is that they are unnecessary, in the sense that we can always convert a presentation modulo into a regular presentation, see Lemma 224 below. However, the techniques developed here extend in the case of 2-categories (this will be developed in a subsequent article) and moreover, our framework already enables us to easily obtain interesting results on presented categories, see Section 9.3.3.

Definition 221. Given a presentation modulo $(P, \tilde{P}_1)$, we define the *quotient presentation* $P/\tilde{P}_1$ as the presentation (P'_0, P'_1, P'_2) where

- $P'_0 = P_0/\cong_1$ where $\cong_1$ is the smallest equivalence such that $x \cong_1 y$ whenever there exists a generator $f : x \to y$ in $\tilde{P}_1$, and we denote $[x]$ the equivalence class of $x \in P_0$,
- the elements of P'_1 are $f : [x] \to [y]$ for $f : x \to y$ in P_1,
- the elements of P'_2 are of the form $\alpha : f \to g$ for $\alpha : f \to g$ in P_2, or $\alpha_f : f \to \mathrm{id}_{[x]}$ for $f : x \to y$ in $\tilde{P}_1$.

We will sometimes need to consider presentations modulo with "arrows reversed":

Definition 222. Given a presentation modulo $(P, \tilde{P}_1)$, the presentation modulo $(P^{\mathrm{op}}, \tilde{P}_1^{\mathrm{op}})$ is given by $P^{\mathrm{op}} = (P_0, P_1^{\mathrm{op}}, P_2^{\mathrm{op}})$ where $P_1^{\mathrm{op}} = \{f^{\mathrm{op}} : y \to x \mid f : x \to y \in P_1\}$ and where $P_2^{\mathrm{op}} = \{\alpha^{\mathrm{op}} : f^{\mathrm{op}} \Rightarrow g^{\mathrm{op}} \mid \alpha : f \Rightarrow g\}$ with $f^{\mathrm{op}} = f_1^{\mathrm{op}} \circ ... \circ f_k^{\mathrm{op}}$ for $f = f_k \circ \ldots \circ f_1$ and where $\tilde{P}_1^{\mathrm{op}}$ is the subset of P_1^{op} corresponding to $\tilde{P}_1$.

9.1.3 Quotient and localization of a presentation modulo

As explained above, we want to quotient our presentations modulo by equational morphisms, in order for the equational morphisms to induce equalities in the presented category. Given a category $\mathcal{C}$ and a set Σ of morphisms, there are essentially two canonical ways to "get rid" of the morphisms of Σ in $\mathcal{C}$: we can either force them to be identities, or to be isomorphisms, giving rise to the following two notions of quotient and localization of a category. These are standard constructions in category theory and we recall them below.

Definition 223. The *quotient* of a category $\mathcal{C}$ by a set Σ of morphisms of $\mathcal{C}$ is a category $\mathcal{C}/\Sigma$ together with a *quotient functor* $Q : \mathcal{C} \to \mathcal{C}/\Sigma$ sending the elements of Σ to identities, such that for every functor $F : \mathcal{C} \to \mathcal{D}$ sending the elements of Σ to identities, there exists a unique functor $\tilde{F}$ such that $\tilde{F} \circ Q = F$.

Such a quotient category always exists for general reasons [BBP99] and is unique up to isomorphism. Given a presentation modulo $(P, \tilde{P}_1)$, the category presented by the associated (non-modulo) presentation $P/\tilde{P}_1$ described in Definition 221, corresponds to considering the category presented by the (non-modulo) presentation P and quotienting it by $\tilde{P}_1$.

Lemma 224. *Suppose given a presentation modulo* $(P, \tilde{P}_1)$*. The categories* $\|P\|/\tilde{P}_1$ *and* $\|P/\tilde{P}_1\|$ *are isomorphic.*

A second, slightly different construction, consists in turning elements of Σ into isomorphisms (instead of identities):

Definition 225. The *localization* of a category $\mathcal{C}$ by a set Σ of morphisms is the category $\mathcal{C}[\Sigma^{-1}]$ together with a *localization functor* $L : \mathcal{C} \to \mathcal{C}[\Sigma^{-1}]$ sending the elements of Σ to isomorphisms, such that for every functor $F : \mathcal{C} \to \mathcal{D}$ sending the elements of Σ to isomorphisms, there exists a unique functor $\tilde{F}$ such that $\tilde{F} \circ L = F$.

In the case where the category is presented, its localization admits the following presentation.

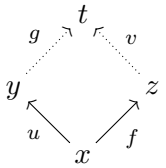
Lemma 226. *Given a presentation $P = (P_0, P_1, P_2)$ and a subset Σ of P_1, the category presented by $P' = (P_0, P_1', P_2')$ where $P_1' = P_1 \sqcup \left\{ \overline{f} : y \to x \;\middle|\; f : x \to y \in \Sigma \right\}$ and where $P_2' = P_2 \sqcup \left\{ \overline{f} \circ f \Rightarrow \mathrm{id}, f \circ \overline{f} \Rightarrow \mathrm{id} \;\middle|\; f \in \Sigma \right\}$ is a localization of the category $\|P\|$ by Σ.*

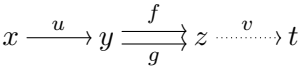
Example 227. Let us consider the category $\mathcal{C} = \; x \underset{g}{\overset{f}{\rightrightarrows}} y$ with two objects and two non-trivial morphisms. Its localization by $\Sigma = \{f, g\}$ is equivalent to the category with one object and $\mathbb{Z}$ as set of morphisms (with addition as composition), whereas its quotient by Σ is the category with one object and only identity as morphism. Notice that they are not equivalent.

The description of the localization of a category provided by the universal property is often difficult to work with. When the set Σ has nice properties, the localization admits a much more tractable description:

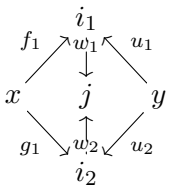
Definition 228 ([GZ67, Bor94])**.** A set Σ of morphisms of a category $\mathcal{C}$ is a *left calculus of fractions* when

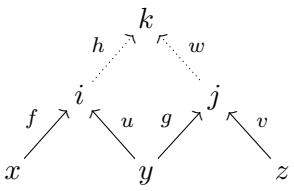
1. the set Σ is closed under composition : for f and g composable morphisms in Σ, $g \circ f$ is in σ.
2. Σ contains the identities id_x for x in P_0.
3. for every pair of coinitial morphisms $u : x \to y$ in Σ and $f : x \to z$ in $\mathcal{C}$, there exists a pair of cofinal morphisms $v : z \to t$ in Σ and $g : y \to t$ in $\mathcal{C}$ such that $v \circ f = g \circ u$.
4. for every morphism $u : x \to y$ in Σ and pair of parallel morphisms $f, g : y \to z$ such that $f \circ u = g \circ u$ there exists a morphism $v : z \to t$ in Σ such that $v \circ f = v \circ g$.





Theorem 229 ([GZ67, Bor94])**.** *When Σ is a left calculus of fractions for a category $\mathcal{C}$, the localization $\mathcal{C}[\Sigma^{-1}]$ can be described as the* category of fractions *whose objects are the objects of $\mathcal{C}$ and morphisms from x to y are equivalence classes of pairs of cofinal morphisms (f, u) with $f : x \to i \in \mathcal{C}$ and $u : y \to i \in \Sigma$ under the equivalence relation identifying two such pairs (f_1, u_1) and (f_2, u_2) when there exist two morphisms $w_1, w_2 \in \Sigma$ such that $w_1 \circ u_1 = w_2 \circ u_2$ and $w_1 \circ f_1 = w_2 \circ f_2$, as shown on the left:*





identity on an object x is the equivalence class of $(\mathrm{id}_x, \mathrm{id}_x)$ and composition of two morphisms $(f,u) : x \to y$ and $(g,v) : y \to z$ is the equivalence class of $(h \circ f, w \circ v) : x \to z$ where the morphisms h and w are provided by property 1 of Definition 228.

In such a situation, the following property often enables one to show that there is a full and faithful embedding of the category into its localization:

Proposition 230 ([Bor94])**.** *Given a left calculus of fractions Σ for a category $\mathcal{C}$, if all the morphisms of Σ are mono, then the inclusion functor $F : \mathcal{C} \to \mathcal{C}[\Sigma^{-1}]$ is faithful, where F is the identity on objects and sends a morphism $f : x \to y$ to (f, id_y).*

Given a presentation modulo, when the (abstract) rewriting system on objects given by the equational generators is convergent, normal forms for objects provide canonical representatives of objects modulo equational generators, and therefore we are actually provided with three possible and equally reasonable constructions for the category presented by a presentation modulo $(P, \tilde{P}_1)$:

1. the full subcategory on $\|P\|$ whose objects are normal forms wrt $\tilde{P}_1$,
2. the quotient category $\|P\|/\tilde{P}_1$,
3. the localization $\|P\|[\tilde{P}_1^{-1}]$.

The aim of this article is to provide reasonable assumptions on the presentation modulo ensuring that the first two categories are isomorphic, and equivalent to the third one. We introduce them gradually.

9.2 Confluence properties

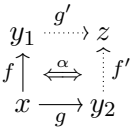
In this section, we introduce a series of local conditions that our presentations modulo that are sufficient to ensure that the constructions recalled above to coincide. These can be seen as a generalization of classical local confluence properties in our context in which rewriting rules correspond to equational generators only, and in which we keep track of 2-cells witnessing local confluence.

9.2.1 Residuation

We begin by extending, to our setting, the notion of residual, which is often associated with a confluent rewriting system in order to "keep track" of rewriting steps once others have been performed [Lév78, BKd03, DDG+15].

Assumption 1. We suppose fixed a presentation modulo $(P, \tilde{P}_1)$ such that

1. for every pair of distinct coinitial generators $f : x \to y_1$ in $\tilde{P}_1$ and $g : x \to y_2$ in P_1, there exists a pair of cofinal morphisms $g' : y_1 \to z$ in P_1^* and $f' : y_2 \to z$ in $\tilde{P}_1^*$ and a relation $\alpha : g' \circ f \Leftrightarrow f' \circ g$ in P_2,



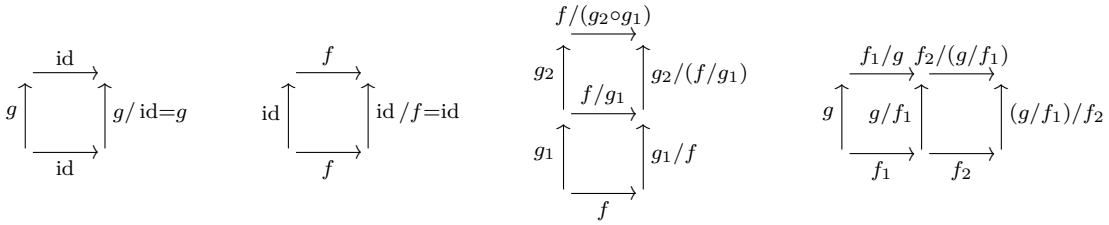
2. there is no infinite path with generators in $\tilde{P}_1$.

These assumptions ensure in particular that the (abstract) rewriting system on vertices with $\tilde{P}_1$ as set of rules is convergent. Given a vertex $x \in P_0$, we write $\hat{x}$ for the associated normal form. For every pair of distinct morphisms (f, g), as in the first assumption, we suppose fixed an arbitrary choice of a particular triple (g', α, f') associated with it, and write g/f for g', f/g for f' and $\rho_{f,g}$ for α. The morphism g/f (resp. f/g) is the *residual* of g after f (resp. f after g): intuitively, g/f corresponds to what remains of g once f has been performed. It is natural to extend this definition to paths as follows:

Definition 231. Given two coinitial paths $f : x \to y$ and $g : x \to z$ and P_1^* such that either f or g is in $\tilde{P}_1^*$, we define the *residual* g/f of g after f as above when f and g are distinct generators, and by induction with $f/f = \mathrm{id}_y$ and

$$g/\mathrm{id}_x = g \qquad \mathrm{id}_x/f = \mathrm{id}_y \qquad (g_2 \circ g_1)/f = (g_2/(f/g_1)) \circ (g_1/f) \qquad g/(f_2 \circ f_1) = (g/f_1)/f_2$$

(by convention the residual g/f is not defined when neither f nor g belongs to $\tilde{P}_1^*$). Graphically,



It can be checked that residuation is well-defined on the morphisms of the free category P_1^* in the sense that it is compatible with associativity and identities, and moreover it does not depend on the order in which rules are applied, see Lemma 235. In order for the definition to be well-founded, and thus always defined, we will make the following additional assumption.

Assumption 2. There is a weight function $\omega_1 : P_1 \to \mathbb{N}$, and we still write $\omega_1 : P_1^* \to \mathbb{N}$ for its extension as morphism of category to the category corresponding to the additive monoid $(\mathbb{N}, +)$, such that for every generator $g \in P_1$ and $f \in \tilde{P}_1$, we have $\omega_1(g/f) < \omega_1(g)$.

Remark 232. In order to simplify the presentation, we did not present the most general axiomatization for the weight function. An important point is that it induces a well-founded ordering on elements of P_1^* and satisfies properties similar to monomial orderings:

— it is compatible with composition: if $\omega_1(g) < \omega_1(g')$ then $\omega_1(h \circ g \circ f) < \omega_1(h \circ g' \circ f)$,
— identities are minimal elements: $\omega_1(\mathrm{id}) < \omega_1(f)$ for every $f \neq \mathrm{id}$; in particular, we have $\omega_1(g) < \omega_1(h \circ g \circ f)$ for $f, h \neq \mathrm{id}$.

In order to study confluence of the rewriting system provided by equational morphisms, through the use of residuals, we first introduce the following category, which allows us to consider, at the same time, both residuals g/f and f/g of two coinitial morphisms f and g.

Definition 233. The *zig-zag presentation* associated with the presentation modulo $(P, \tilde{P}_1)$ is the presentation $Z = (Z_0, Z_1, Z_2)$ with $Z_0 = P_0$, $Z_1 = P_1 \sqcup \overline{\tilde{P}_1}$ (generators in $\overline{\tilde{P}_1}$ are of the form $\overline{f} : B \to A$ for some generator $f : A \to B$ in $\tilde{P}_1$) and relations in Z_2 are of the form $g \circ \overline{f} \Rightarrow \overline{(f/g)} \circ (g/f)$ or $f \circ \overline{f} \Rightarrow \mathrm{id}_y$ for some pair of distinct coinitial generators $f : x \to y \in \tilde{P}_1$ and $g : x \to z \in P_1$.

By suitably extending the weight function of Assumption 2 to morphisms in Z_1^*, one can show that this presentation is convergent, and normal forms correspond to taking residuals:

Lemma 234 ([CM15])**.** *The rewriting system on morphisms in Z_1^* with Z_2 as rules is convergent. Given two coinitial morphisms $f : x \to y$ in $\tilde{P}_1^*$ and $g : x \to z$ in P_1^*, the normal form of $g \circ \overline{f}$ is $\overline{(f/g)} \circ (g/f)$.*

As a direct corollary of the convergence of the rewriting system, one can show that Definition 231 makes sense:

Lemma 235 ([CM15])**.** *The residuation operation does not depend on the order in which equalities of Definition 231 are applied.*

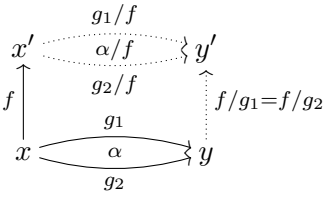
Moreover, by using Lemma 234, one can show that a "global" version of the residuation property (Assumption 1) holds:

Proposition 236 ([CM15])**.** *Given two coinitial morphisms $f : x \to y$ in $\tilde{P}_1^*$ and $g : x \to z$ in P_1^*, there exists a relation $\alpha : (g/f) \circ f \overset{*}{\Leftrightarrow} (f/g) \circ g$.*

9.2.2 The cylinder property

In the previous section, we have studied residuation, which enables one to recover a residual g/f of a morphism g after a coinitial equational morphism f. We now strengthen our hypothesis in order to ensure that if two morphisms are equal (wrt the equivalence generated by P_2^*) then their residuals after a same morphism are equal, i.e. equality is compatible with residuation.

Assumption 3. The presentation $(P, \tilde{P}_1)$ satisfies the *cylinder property*: for every triple of coinitial morphism generators $f : x \to x'$ in $\tilde{P}_1$ (resp. in P_1) and $g_1, g_2 : x \to y$ in P_1^* (resp. in $\tilde{P}_1^*$) such that there exists a relation $\alpha : g_1 \Leftrightarrow g_2$, we have $f/g_1 = f/g_2$ and there exists a 2-cell $g_1/f \overset{*}{\Leftrightarrow} g_2/f$. We write α/f for an arbitrary choice of such a 2-cell:



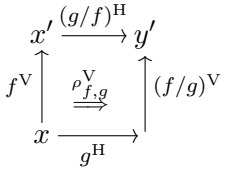
As in previous section, we would like to extend this "local" property (f and α are supposed to be generators) to a "global" one (where f and α can be composites of cells):

Proposition 237 (Global cylinder property [CM15])**.** *Given coinitial morphisms $f : x \to x'$ in $\tilde{P}_1^*$ (resp. in P_1^*) and $g_1, g_2 : x \to y$ in P_1^* (resp. in $\tilde{P}_1^*$) such that there exists a composite relation $\alpha : g_1 \overset{*}{\Leftrightarrow} g_2$, we have $f/g_1 = f/g_2$ and there exists a 2-cell $g_1/f \overset{*}{\Leftrightarrow} g_2/f$.*

The proof of previous proposition requires generalizing, in dimension 2, the termination condition (Assumption 2) and the construction of the zig-zag presentation (Definition 233).

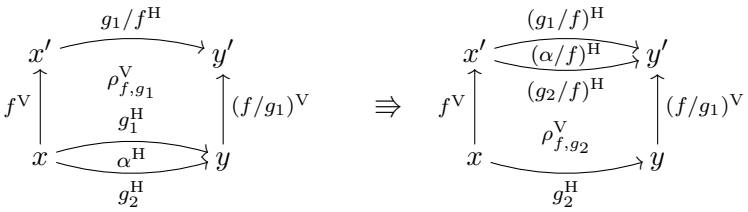
Definition 238. The *2-zig-zag presentation* associated with $(P, \tilde{P}_1)$ is $Y = (Y_0, Y_1, Y_2)$ with $Y_0 = P_0$, $Y_1 = P_1^{\mathrm{H}} \sqcup P_1^{\mathrm{V}}$ (where the morphisms of P_1^{H} are called *horizontal* of the form $f^{\mathrm{H}} : A \to B$ for some morphism $f : A \to B$ in P_1 and similarly for the morphisms in P_1^{V} which are called *vertical*), and the 2-cells in $Y_2 = Y_2^{\mathrm{H}} \sqcup Y_2^{\mathrm{V}}$ are either

— horizontal 2-cells: $Y_2^{\mathrm{H}} = P_2^{\mathrm{H}} \sqcup \overline{P_2}^{\mathrm{H}}$ (i.e. relations in P_2 taken forward or backward, and decorated by H)
— vertical 2-cells: given two generators $f : x \to y$ and $g : x \to z$ in P_1 such that f or g belongs to $\tilde{P}_1$, we have a relation $\rho^{\mathrm{V}}_{f,g} : (g/f)^{\mathrm{H}} \circ f^{\mathrm{V}} \Rightarrow (f/g)^{\mathrm{V}} \circ g^{\mathrm{H}}$ in Y_2^{V}:



The rewriting system on the 2-cells in Y_2^* is the following: for every 2-cell $\alpha : g_1 \Leftrightarrow g_2 : x \to y$ in P_2, for every coinitial 1-cell $f : x \to x'$ in P_1 such that either f or both g_1 and g_2 belong to $\tilde{P}_1^*$, there is a rewriting rule

$$((f/g_1)^{\mathrm{V}} \circ \alpha^{\mathrm{H}}) \bullet \rho^{\mathrm{V}}_{f,g_1} \quad \Rrightarrow \quad \rho^{\mathrm{V}}_{f,g_2} \bullet ((\alpha/f)^{\mathrm{H}} \circ f^{\mathrm{V}}) \tag{9.1}$$



where $\circ$ (resp. $\bullet$) denotes horizontal (resp. vertical) composition in a 2-category.

In order to ensure the termination of the rewriting system, we suppose the following.

Assumption 4. There is a weight function $\omega_2 : P_2^{\mathrm{H}} \to \mathbb{N}$ such that for every $\alpha : g_1 \Rightarrow g_2$ in Y_2^* and f in P_1 such that α/f exists we have $\omega_2(\alpha/f) < \omega_2(\alpha)$. We use the same notation $\omega_2 : \left(P_2^{\mathrm{H}} \sqcup \overline{P_2}^{\mathrm{H}}\right)^* \to \mathbb{N}$ for the function such that $\omega_2(\overline{\alpha}) = \omega_2(\alpha)$ and both horizontal and vertical compositions are sent to addition ($\mathbb{N}$ being a commutative additive monoid, this definition is compatible with the axioms of 2-categories, such as associativity or exchange law).

Corollary 239. *The rewriting system* (9.1) *is convergent.*

Proposition 237 follows easily, by a reasoning similar to that used for Proposition 236.

The cylinder property has many interesting consequences for the residuation operation, as we now investigate.

Proposition 240 ([CM15])**.** *In the category $\|P\|$, every equational morphism is epi.*

Proof. Suppose given $f : x \to y$ in $\tilde{P}_1^*$, and $g_1, g_2 : y \to z$ in P_1^* such that $g_1 \circ f \overset{*}{\Leftrightarrow} g_2 \circ f$. By Proposition 237, we have $g_1 = (g_1 \circ f)/f \overset{*}{\Leftrightarrow} (g_2 \circ f)/f = g_2$. □

Proposition 241 ([CM15])**.** *In the category $\|P\|$, every morphism g admits a pushout along a coinitial equational morphism f given by g/f.*

Proof. By Proposition 236, we have $(g/f) \circ f \overset{*}{\Leftrightarrow} (f/g) \circ g$ and $(g/f, f/g)$ can be shown to form a universal cocone by using Propositions 237 and 240. □

9.3 Comparing presented categories

9.3.1 The category of normal forms

We show here that with our hypotheses on the rewriting system, the quotient category $\|P\|/\tilde{P}_1$ can be recovered as the following subcategory of $\|P\|$, whose objects are those which are in normal form for $\tilde{P}_1$.

Definition 242. The *category of normal forms* $\|P\|\downarrow\tilde{P}_1$ is the full subcategory of $\|P\|$ whose objects are the normal forms of elements of P_0 wrt rules in $\tilde{P}_1$. We write $I : \|P\|\downarrow\tilde{P}_1 \to \|P\|$ for the inclusion functor.

Theorem 243 ([CM15])**.** *The category $\|P\| \downarrow \tilde{P}_1$ is isomorphic to the quotient category $\|P\|/\tilde{P}_1$.*

Proof. One shows that $\|P\|\downarrow\tilde{P}_1$ satisfies the universal property defining the quotient category. The global cylinder property shown in Proposition 237 is helpful here to ensure that the required functors are properly defined. □

9.3.2 Equivalence with localization

We now show that the previous two constructions (quotient and normal forms) also coincide with the third possible construction which consists in formally adding inverses for equational morphisms. First, notice that we can use the description of the localization $\|P\|[\tilde{P}_1^{-1}]$ as a category of fractions given in Theorem 229:

Lemma 244 ([CM15])**.** *The set of equational morphisms of $\|P\|$ is a left calculus of fractions.*

Proof. We have to show that the set of equational morphisms satisfies the four conditions of Definition 228: the first two (closure under composition and identities) are immediate, the third one follows from Proposition 236, and the last one is ensured by the fact that all equational morphisms are epi by Proposition 240. □

Our proof of the equivalence is based on the embedding of the presented category into the localization provided by Proposition 230. In order for the hypothesis of this proposition to hold, we first need to impose that the same properties hold for the opposite presentation as for the presentation itself:

Assumption 5. The presentation modulo $(P^{\mathrm{op}}, \tilde{P}^{\mathrm{op}})$ satisfies Assumptions 1, 2, 3 and 4.

This implies that the duals of previously shown properties hold for $\|P\|$. For instance, by the dual of Proposition 240, all equational morphisms are mono, from which follows, by Proposition 230:

Proposition 245 ([CM15])**.** *The canonical functor $\|P\| \to \|P\|[\tilde{P}_1^{-1}]$ is faithful.*

Definition 246 ([CM15])**.** A presentation modulo satisfying assumptions 1 to 5 is called *coherent*.

Theorem 247 ([CM15])**.** *Given a coherent presentation modulo $(P, \tilde{P}_1)$, the categories $\|P\|/\tilde{P}_1$ and $\|P\|[\tilde{P}_1^{-1}]$ are equivalent.*

Proof. Consider the functor $F : \|P\| \downarrow \tilde{P}_1 \to \|P\|[\tilde{P}_1^{-1}]$ defined as the composite of the inclusion functor $I : \|P\| \downarrow \tilde{P}_1 \to \|P\|$, introduced in Definition 242, with the localization functor $L : \|P\| \to \|P\|[\tilde{P}_1^{-1}]$, see Definition 225. The functor F is faithful since it is the case for both I and L by Proposition 245. It is also full. Namely, by Theorem 229, given any two objects $\hat{x}$ and $\hat{y}$ of $\|P\| \downarrow \tilde{P}_1$, a morphism from $F(\hat{x}) = \hat{x}$ to $F(\hat{y}) = \hat{y}$ in $\|P\|[\tilde{P}_1^{-1}]$ is of the form (f, u) with $f : \hat{x} \to z$ and $u : \hat{y} \to z$ equational. Since $\hat{y}$ is a normal form, we necessarily have $u = \mathrm{id}_{\hat{y}}$ and thus $(f, u) = Ff$. Finally, given an object $y \in \|P\|[\tilde{P}_1^{-1}]$, there is a morphism $u : y \to \hat{y}$ in $\tilde{P}_1^*$ to its normal form which induces an isomorphism $y \cong \hat{y}$ in $\|P\|[\tilde{P}_1^{-1}]$. The functor F is thus an equivalence of categories. □

9.3.3 An example: the dihedral category $D_4^\bullet$

As an illustration of our results, we study presentations of a family of categories arising as variants of the dihedral groups. Recall that the *dihedral group* D_n is the group of isometries of the plane preserving a regular polygon with n faces. This group is generated by a rotation r of angle $2\pi/n$ and a reflection s, and can be described as the free group over the two generators r and s quotiented by the congruence generated by the three relations $s^2 = \mathrm{id}$, $r^n = \mathrm{id}$ and $rsr = s$. We consider here a variant of this group: the category $D_n^\bullet$ of isometries of the plane preserving a regular polygon with n faces together with a distinguished vertex (the category thus has n objects). For instance, the category $D_4^\bullet$ is pictured on the left below, the distinguished vertex of the square being pictured by a black triangle:

(9.2)

This category $D_4^\bullet$ admits a presentation P with 4 objects and 8 generating morphisms, as pictured on the left above, satisfying the 12 relations:

$$r_{i+3} \circ r_{i+2} \circ r_{i+1} \circ r_i = \mathrm{id} \qquad s_{j+1} \circ s_j = \mathrm{id} \qquad r_j \circ s_{j+1} \circ r_j = s_j$$
$$s_j \circ s_{j+1} = \mathrm{id} \qquad r_{j+3} \circ s_{j+2} \circ r_{j+1} = s_{j+1}$$

for $i \in \{1, \ldots, 4\}$ and $j \in \{1, 3\}$, where the indices are to be taken modulo 4 so that they lie in $\{1, \ldots, 4\}$.

The methodology introduced earlier can be used to show that by quotienting (resp. localizing) by $\Sigma = \{r_2, r_4\}$, we obtain a category which is isomorphic (resp. equivalent) to $D_2^\bullet$: intuitively, "forgetting" about those rotations quotients the square under symmetry wrt an horizontal axis. We thus consider the presentation modulo $(P, \tilde{P}_1)$ with $\tilde{P}_1 = \Sigma$. Unfortunately, this presentation does not satisfy the assumptions required to apply our results; for instance, there is no residual of r_2 after s_2. It is thus necessary to complete the presentation in order to have the confluence properties (namely, the residuation and cylinder properties). In rewriting theory, when a rewriting system is not confluent, one usually tries to complete it (typically using a Knuth-Bendix completion algorithm) in order for confluence to hold. Similarly, we can transform our presentation using a series of Tietze transformations

(see Section 7.1.2) while preserving the same presented category, in order to obtain another presentation of the same category which satisfies the required assumptions.

We first consider the presentation P' obtained from P by adding the generator $\overline{r}_4 = r_3 \circ r_2 \circ r_1$ and its defining relation, as well as the derivable relations $r_4 \circ \overline{r}_4 = \mathrm{id}$ and $r_4 \circ \overline{r}_4 = \mathrm{id}$. We can now define r_2/s_2 as $\overline{r}_4$, if we consider $\overline{r}_4$ as an equational morphism. Fortunately, the following lemma shows that we can quotient, or localize, by $\overline{r}_4$ instead of r_4, and we therefore define $\tilde{P}'_1 = \{r_2, \overline{r}_4\}$:

Lemma 248. *Let P be a presentation of category such that there exist f and g in P_1 and two relations $f \circ g \Leftrightarrow \mathrm{id}$ and $g \circ f \Leftrightarrow \mathrm{id}$ in P_2. Let Σ be a subset of P_1 not containing f nor g. Then the quotients (resp. localizations) of $\|P\|$ by $\Sigma \sqcup \{f\}$, $\Sigma \sqcup \{f, g\}$, and $\Sigma \sqcup \{g\}$ are isomorphic.*

In this way, we have transformed the presentation $(P, \tilde{P}_1)$ into a presentation $(P', \tilde{P}'_1)$ for which we can now define the residual r_2/s_2. Similarly, in order for all the required residuals to be defined, we modify P' using Tietze transformations by adding generators $\overline{r}_1 = r_4 \circ r_3 \circ r_2$ and $\overline{r}_3 = r_2 \circ r_1 \circ r_4$ and modifying the set of relations. Finally, we end up with a presentation P'' which has 11 morphism generators $r_i, s_i, \overline{r}_k$, as shown on the right of (9.2), and 16 relations:

$$\begin{array}{lllll} s_{j+1} \circ s_j = \mathrm{id} & r_1 \circ s_2 \circ r_1 = s_1 & r_k \circ \overline{r}_k = \mathrm{id} & r_2 \circ r_1 = \overline{r}_3 \circ \overline{r}_4 & s_3 \circ r_2 = \overline{r}_4 \circ s_2 \\ s_j \circ s_{j+1} = \mathrm{id} & \overline{r}_3 \circ s_3 \circ \overline{r}_3 = s_4 & \overline{r}_k \circ r_k = \mathrm{id} & r_3 \circ r_2 = \overline{r}_4 \circ \overline{r}_1 & r_2 \circ s_1 = s_4 \circ \overline{r}_4 \end{array}$$

for $i \in \{1, \dots, 4\}$, $j \in \{1, 3\}$ and $k \in \{1, 3, 4\}$, which is considered modulo $\tilde{P}''_1 = \{r_2, \overline{r}_4\}$. This presentation modulo is coherent. It satisfies convergence assumption 1, and residuals are defined by

$$r_2/s_2 = r_2/\overline{r}_1 = \overline{r}_4 \quad \overline{r}_4/s_1 = \overline{r}_4/r_1 = r_2 \quad s_1/\overline{r}_4 = s_4 \quad r_1/\overline{r}_4 = \overline{r}_3 \quad s_2/r_2 = s_3 \quad \overline{r}_1/r_2 = r_3$$

For termination assumption 2, we define ω_1 as equal to 1 on s_1, s_2, $\overline{r}_1$ and r_1 and 0 on other morphism generators. The cylinder assumption 3 follows from considering 5 diagrams. For termination assumption 4 we define ω_2 as 1 on relation generators such that the only morphism generators occurring in the source or the target are r_1, $\overline{r}_1$, s_1 or s_2, and as 0 otherwise. It can be checked similarly that $(P''^{\mathrm{op}}, (\tilde{P}''_1)^{\mathrm{op}})$ satisfies the assumptions. Therefore $\|P''\| \downarrow \{r_2, \overline{r}_4\}$ is isomorphic to $\|P''\| / \{r_2, \overline{r}_4\}$ by Theorem 243, and equivalent to $\|P''\|[\{r_2, \overline{r}_4\}^{-1}]$ by Theorem 247, the left-to-right part of the equivalence being an embedding by Proposition 245. An explicit (non-modulo) presentation for the quotient can be obtained by Lemma 224, and this presentation is Tietze equivalent to the canonical presentation of $D_2^\bullet$. We finally obtain the following result:

Theorem 249 ([CM15])**.** *The category $D_2^\bullet$ is isomorphic to the quotient $D_4^\bullet / \{r_2, r_4\}$, embeds fully and faithfully into the category $D_4^\bullet$, and is equivalent to the localization $D_4^\bullet[\{r_2, r_4\}^{-1}]$.*

Remark 250. In this case, since r_2 and r_4 are already invertible in $\|P\|$, we moreover have $D_4^\bullet[\{r_2, r_4\}^{-1}] \cong D_4^\bullet$.

This illustrates the fact that, even though restricted for now to categories, the tools developed in this article enable one to obtain interesting results about presented categories.

9.4 Future work

As illustrated in the example above, the properties we have shown allow us to obtain interesting results. We would like to briefly mention some currently studied extensions to this work.

Other termination properties. The assumption that the rewriting systems we are considering are terminating is sometimes too strong and we would like to investigate some variants of this condition. In particular, Dehornoy et al. have given conditions on a presented monoid under which a monoid embeds into its enveloping groupoid [DDG$^+$15]: this is a particular case of the situation we are considering, since taking the enveloping groupoid amounts to localize wrt every generator. However, a monoid being a category with only one object, there is no hope that the rewriting system formed by all the generators is terminating since they all trivially loop! In fact, Dehornoy uses a different condition (we do not say weaker because it is not clear that termination implies this condition), and we would like to investigate a likely generalization to the case of localization.

Extension to presentations of 2-categories. The case of presentations of categories is however somehow limited since, by Lemma 224, for every presentation modulo there is a presentation (without modulo) presenting the same category. As explained in the introduction of this chapter, this is not the case for presentations of 2-categories, which is thus more interesting. We have good hope that our proof techniques extend to this case. We would like to briefly mention some of the adjustments necessary to cover this case. Firstly, since the exchange law in a 2-category ensures that two disjoint rewrites commute, it is enough to impose the existence of suitable residuals for critical pairs only (this is, in our context, a variant of Newman's lemma), and similarly the cylinder property only has to be imposed for triples of coinitial rewriting rules forming a critical triple. Secondly, since in practice not all operations (residuation for instance in our example) are compatible with the exchange law, one actually has to explicitly handle this law and work in the setting of sesquicategories. Thirdly, the precise notion of equivalence between 2-categories is subtle. For instance, the canonical "inclusion" functor $\|P\| \downarrow \tilde{P}_2 \hookrightarrow \|P\|$, exhibiting the restriction to 1-cells in normal form as a "sub-2-category" of $\|P\|$, is in fact a lax 2-functor: the 0-composition of two 1-cells in normal form is not necessarily a normal form, but always normalizes to one.

Chapter 10

Toward an implementation of polygraphs

We have seen in previous chapters that the setting of polygraphs is very rich and enables one to perform many computations on the presentations of categories, or even coherent presentations or resolutions. It would of course be desirable to have a program which could help handling the inherent combinatorial complexity of those structures. The typical operations that such a program could perform would be: implement Tietze transformations (the use of the program would guarantee that we have not changed the presented category), compute (partial) resolutions from convergent presentations, orient and complete presentations using the Knuth-Bendix algorithm, compute the homology of a resolution, etc. As mentioned in Chapter 8, a prototype of such a tool for coherent presentations of monoids has been implemented but this remains low-dimensional and, to the best of our knowledge, no tool for handling rewriting in higher-categorical structures exists. The only related tool we are aware of is *Globular* [BCK+15] which allows the formalization of morphisms in semistrict higher categories, but provides no automation or rewriting tools as for now.

The main difficulty here is to provide a concrete and reasonably efficient data structure for representing the n-cells in the n-category P^* freely generated by an n-polygraph P. For instance, a naive approach could be to consider the formal composites of generators in P_n, i.e. roughly define P_n^* as the smallest set of terms such that

— P_n^* contains every n-generator of P_n,

— if $f, g \in P_n^*$ are two i-composable terms then "$g \circ_i f$" is also a term of P_n^*.

However, one would then have to explicitly work modulo the axioms of n-categories (such as the exchange laws), which is very difficult because there is apparently no reasonable notion of canonical representative form for terms modulo the axioms. Our approach consists in generalizing an inductive data structure whose elements are (up to isomorphism) the globular sets in order to obtain one which is able to represent polygraphs: the elements of this data structure should be (up to isomorphisms) the n-cells of the category generated by an n-polygraph. Notice that the notion of equivalence here (isomorphism) is much simpler than the previous one (the laws of n-categories). The case of polygraphs is however much more difficult than the case of globular sets, in particular because n-polygraphs are not presheaf categories for $n \geqslant 3$ [MZ01, Che12], so that the generalization is not immediate. The first ideas on this approach started being developed during the author's PhD thesis [Mim08], see also [Mim14], and were later on inspired by Batanin's approach using monoidal globular categories [Bat98b] and Burroni's logographs [Bur12]. The results mentioned in this chapter

are preliminary and still need to be developed and checked in order to give rise to a proper implementation. However, we would like to present our current ideas on how to approach the problem and propose a data structure which we think could be used to represent polygraphs. The examples are given in the OCaml language.

Before we begin, we would like to emphasize an important point about our goal. We want to find out a data structure which is able to faithfully represent the mathematical objects we are interested in, mainly polygraphs here, but not necessarily fully. This means that we do not require every element of the data structure to correspond to a mathematical object: those for which it is the case can be characterized as satisfying certain invariants. The preservation of these invariants is usually ensured by preventing the user from directly manipulating the data structures: a library offers functions to manipulate those, and if the user only uses these functions, the invariants will be maintained.

Since the structure of polygraph is intricate, we first present our main ideas for an implementation on simpler cases: the construction of the free category on a graph (Section 10.1) and the free ω-category on a globular set (Section 10.2). Finally, we explain how these constructions could be generalized to polygraphs (Section 10.3).

10.1 The free category on a graph in OCaml

Before even considering globular sets, we would like to illustrate some ideas on the particular case of the category of graphs: given a representation of a graph, we would like to represent the morphisms of the category it freely generates, i.e. its paths, in a way which will generalize later on.

10.1.1 Graphs

A graph consists of two sets (of vertices and edges). Because our machines are finite, we will only consider finite graphs, so that the elements of the sets can be represented by any data type which is at least countable (or can be thought so), typically an integer (of type `int`) which is called an *identifier*. A first implementation of a data structure to represent graphs could be the following one:

```
type vertex = int
type edge = int
type graph = {
    vertices : vertex list;
    edges : (edge * (vertex * vertex)) list;
  }
```

A graph is thus represented as a record consisting of a set (concretely, a list) of vertex identifiers and a set of edge identifiers together with their source and target. The fact that elements of the sets are coded as integers makes it hard to maintain coherence. For instance we have to ensure that identifiers are unique, so that if we want to compute the coproduct of a graph with itself we have to arbitrarily rename identifiers to avoid clashes:

$$8 \xrightarrow{5} 2 \quad \sqcup \quad 8 \xrightarrow{5} 2 \qquad = \qquad 4 \xrightarrow{7} 3 \qquad 5 \xrightarrow{3} 2$$

If those identifiers are referred to in other parts of the program, they have to be renamed accordingly. Another, more "cosmetic", point is that there is no deep reason for using these

identifiers: we would like a graph to contain no data, and to correspond to a pure structure of pointers in memory. Finally, as illustrated by the coproduct example above, we need to generate fresh identifiers: it is of course possible to use a global counter, but this is not really modular.

All these defects can be corrected if we remark that we have a canonical source of unique identifiers: memory locations. Namely, when a new data structure is allocated in memory, it is of course placed at a disjoint location. Contrarily to e.g. C, there is no way to know the memory location in OCaml (because data can be moved around by the garbage collector); however, we can check whether two data structures are at the same memory location, and this is actually all we will need. We have at our disposal two types of comparison: *structural equality* `=` which tests if data structures contain the same values, and *physical equality* `==` which tests if two data structures are stored at the same memory location. Of course, physical equality implies structural one, but the converse is not true:

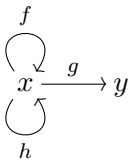
```
# let a = (1,2);;              # a == a;;          # a = a;;
val a : int * int = (1, 2)     - : bool = true     - : bool = true
# let b = (1,2);;              # a == b;;          # a = b;;
val b : int * int = (1, 2)     - : bool = false    - : bool = true
```

From this point of view, values of type `unit ref` should really be considered as *points*: they consist of memory cells which can only contain the value `()`, which is the only element of the terminal type `unit`, but they can of course have different memory locations. In a set-theoretic interpretation, the memory acts as a *Grothendieck universe*, containing all the things we will ever be able to construct or manipulate (we suppose here that the memory is infinite or at least that we never run out of memory). This suggests modifying the definition of graphs as follows:

```
type vertex = unit ref
type edge = vertex * vertex
```

```
type graph = {
    vertices : vertex list;
    edges : edge list;
  }
```

with the convention that we should compare vertices (resp. edges) using physical equality only. For instance, the graph on the left is constructed as on the right:




```
let x = ref ()
let y = ref ()
let f = (x,x)
let g = (x,y)
```

```
let h = (x,x)
let gr = { vertices = [x;y];
           edges = [f;g;h] }
```

(10.1)

and the comparison of vertices (resp. edges) gives the expected results:

```
# x == y;;              # f == h;;
- : bool = false        - : bool = false
# x == x;;              # f == f;;
- : bool = true         - : bool = true
```

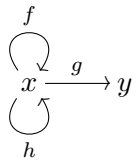
Notice that not every value of type `graph` encodes a valid graph: an actual graph should be such that for every element `(x,y)` of the list of edges, the values `x` and `y` have to be members of the list of vertices. This is a first example of invariant that has to be maintained by a library providing functions to manipulate such data structures. Apart from solving the technical problems raised above, we will see that this approach generalizes to globular sets and polygraphs.

10.1.2 The free category on a graph

It is well-known that the forgetful functor $U : \mathbf{Cat} \to \mathbf{Graph}$, sending a category to its underlying graph admits a left adjoint $-^* : \mathbf{Graph} \to \mathbf{Cat}$ associating, with a graph G, the category G^* with the vertices of G as objects and paths from x to y in G as morphisms from x to y. We would like to implement the construction of G^*, i.e. to have a data structure to represent the morphisms in G^* and perform the usual operations on those, typically composing them. Of course, the paths can be represented as lists of composable generators in G, and composition can simply be implemented as concatenation of lists. Those lists can be considered as canonical representatives of formal composites of edges, modulo the laws of category (associativity and neutral elements): the normal forms for those consist in bracketing compositions on the right and removing identities. In higher dimensions, we do not know any such canonical representative, so we would like to investigate instead another possible representation, using a labeled variant of the structure of graph.

10.1.3 Labeled graphs

Suppose given a graph G called *signature*. A *labeled graph* is an object in the slice category $\mathbf{Graph}/G$, i.e. a graph H together with a morphism of graphs $\ell : H \to G$. For instance, considering G the graph of (10.1) drawn again on the left below, the graph H in the middle (with integers as vertices and edges) can be labeled by $\ell(0) = \ell(1) = \ell(2) = x$, $\ell(3) = y$, $\ell(4) = \ell(5) = f$, $\ell(6) = g$:



$$0 \xrightarrow{4} 1 \xrightarrow{5} 2 \xrightarrow{6} 3 \qquad\qquad x_0 \xrightarrow{f_4} x_1 \xrightarrow{f_5} x_2 \xrightarrow{g_6} y_3 \tag{10.2}$$

In the following, we write x_i for a vertex i with x as label, and similarly for edges. In this case, the vertex x_i is called an *instance* of x, and i the *identifier* of x_i. With this convention, the graph H above, together with its labeling, can be drawn as on the right. The type of labeled graphs is easily defined as a variant of the type of graphs:

```
type lvertex = vertex ref
type ledge = lvertex * edge * lvertex
```

```
type lgraph = {
    lvertices : lvertex list;
    ledges : ledge list;
  }
```

Above, a labeled edge is defined as an instance of a vertex in the base graph, coded as a memory cell containing the labeling vertex. Again, the identifier is implicitly coded by the memory location of the cell. For instance the graph H above can be defined as

```
let x0 = ref x
let x1 = ref x
let x2 = ref x
let y3 = ref y
```

```
let f4 = (x0,f,x1)
let f5 = (x1,f,x2)
let g6 = (x2,g,y3)
```

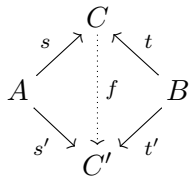
```
let h = {
    lvertices = [x0;x1;x2;y3];
    ledges = [f4;f5;g6]
  }
```

Again, not every value of type `lgraph` encodes a valid graph. Apart from issues similar to the case of graphs, the labeling function, being supposed to be a morphism of graph, has to commute with source and target. In our encoding, this means that every edge `(xi,f,xj)` has to satisfy the conditions `!xi == fst f` and `!xj == snd f`.

10.1.4 Representing the free category

It can now be noticed that labeled graphs provide a representation for morphisms in the free category generated by G. For instance, the graph (10.2) provides a representation for the morphism $g \circ f \circ f$, which is canonical up to the choice of identifiers. In order to make this assertion precise, we should also take in consideration the source and target of morphisms (respectively x_0 and y_3 in the example), and consider the category $\mathrm{Cospan}(\mathbf{Graph}/G)$:

Definition 251. Given a category $\mathcal{C}$ with enough colimits, the bicategory $\mathrm{Cospan}(\mathcal{C})$ has the objects of $\mathcal{C}$ as 0-cells, a 1-cell $A \to B$ is a pair $(s : C \to A, t : C \to B)$ of morphisms of $\mathcal{C}$, called a *cospan* and written $(s, C, t) : A \to B$, and 2-cell between (s, C, t) and (s', C', t') is an isomorphism $f : C \to C'$ making the following diagram commute:



Horizontal composition is given by pushout and vertical composition is induced by composition in $\mathcal{C}$.

Notice that, contrary to more usual habits, we consider only isomorphisms as 2-cells. Given a bicategory $\mathcal{B}$, such as $\mathrm{Cospan}(\mathcal{C})$, there is an induced *quotient category* $\overline{\mathcal{B}}$ obtained by identifying any two 1-cells between which there is a 2-cell, with composition induced by the horizontal composition of the bicategory. The category G^* can then be described as a full subcategory of $\overline{\mathrm{Cospan}(\mathbf{Graph}/G)}$ as follows. In $\mathrm{Cospan}(\mathbf{Graph}/G)$, a cospan is said to be *graded* when its source and target are graphs without edges; a graded cospan $(s, C, t) : A \to B$ in $\mathrm{Cospan}(\mathbf{Graph}/G)$ is said to be *linear* when it is a finite sequence of edges glued along vertices such as in (10.2), i.e. formally

— its source and target graph A and B are reduced to one vertex,
— C is finite,
— a vertex in C is the source and the target of exactly one edge, except the image under s (resp. t) of the vertex of A (resp. B) which is the source (resp. target) of exactly one edge and the target (resp. source) of none.

Equivalently, the last condition can be replaced by the requirement that C should be acyclic. Finally, we suppose fixed a set with one element $\{\star\}$. An object A is *canonical* when it is reduced to one vertex whose identifier is $\star$, i.e. it is of the form $A = x_\star$ for some vertex x of G.

Proposition 252. *The category G^* is isomorphic to the full subcategory of $\overline{\mathrm{Cospan}(\mathbf{Graph}/G)}$ whose objects are canonical ones and morphisms are (equivalence classes of) linear cospans.*

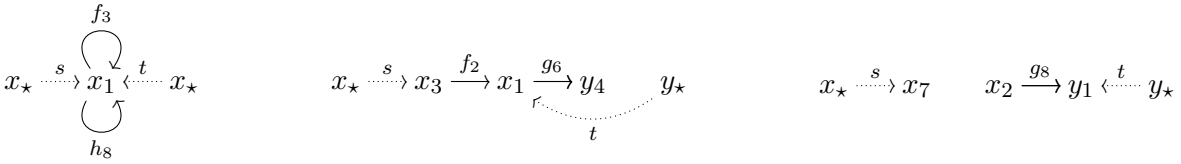
For instance, a linear cospan between canonical objects is shown on the left below:

$$x_\star \overset{s}{\dashrightarrow} x_3 \xrightarrow{f_2} x_1 \xrightarrow{g_6} y_4 \overset{t}{\dashleftarrow} y_\star \qquad x_\star \overset{s}{\dashrightarrow} x_{12} \xrightarrow{f_{56}} x_1 \xrightarrow{g_{43}} y_{98} \overset{t}{\dashleftarrow} y_\star \qquad x_\star \overset{s}{\dashrightarrow} x_1 \overset{t}{\dashleftarrow} x_\star$$

Notice that the identifiers do not really matter here (this is akin to the fact that terms are usually considered modulo α-conversion in λ-calculus): for instance, the span on the left is isomorphic to the span on the middle, and they are thus considered as equal in the

quotient category. This equivalence class corresponds to the morphism $g \circ f$ in G^* via the correspondence of Proposition 252, and similarly the equivalence class of the span on the right above corresponds to the morphism id_x of G^*. Composition corresponds to "gluing graphs along endpoints"; for instance, the fact that $f \circ f$ composed with g is $g \circ f \circ f$, corresponds to the following pushout:

In the correspondence established by Proposition 252, only the linear graphs represent free morphisms. The reason for this should be intuitively clear from the following three examples of non-linear spans:



10.1.5 Generating the free category

The fact that, inside the category of cospans of labeled graphs, those corresponding to free morphisms are the linear ones is not really necessary for our purpose. In practice, all we will need is the fact that those spans can be generated by composing suitable morphisms corresponding to the edges of the signature graph G. Moreover, in higher dimensions, there is no known condition generalizing the linearity condition, but the generation principle will still hold.

With any edge $f : x \to y$ of G as on the left, one can canonically associate a cospan of labeled graphs $[\![f]\!]$ as on the right:

$$x \xrightarrow{f} y \qquad \rightsquigarrow \qquad x_\star \overset{s}{\dashrightarrow} x_0 \xrightarrow{f_0} y_1 \overset{t}{\dashleftarrow} y_\star \tag{10.3}$$

and those generate the category G^*, i.e. by composing them we obtain only linear spans and all of them can be obtained in this way:

Proposition 253. *The category G^* is isomorphic to the full subcategory of $\overline{\mathrm{Cospan}(\mathbf{Graph}/G)}$ whose objects are canonical ones and morphisms are generated by morphisms of the form $[\![f]\!]$ for some edge f of G.*

10.1.6 Renaming objects

In the previous construction, the 2-cells allow for renaming identifiers in the morphisms, but not in the objects. For instance, the following two spans

$$x_0 \overset{s}{\dashrightarrow} x_1 \xrightarrow{f_0} y_2 \overset{t}{\dashleftarrow} y_0 \qquad\qquad x_5 \overset{s}{\dashrightarrow} x_1 \xrightarrow{f_0} y_2 \overset{t}{\dashleftarrow} y_2$$

intuitively represent the same morphism, but there is currently no way of identifying them because 2-cells relate only 1-cells with the same source and the same target: with respect

to this, the situation here is quite similar to the one described in Chapter 9. This explains why in the previous construction we had to choose an arbitrary identifier name $\star$ for objects (if we had allowed any identifier, we would have obtained a category which is equivalent, but not isomorphic, to the category G^*). It would be much more satisfactory to change the notion of "2-cell" in order to allow for renaming objects too. Another related point is that Cospan(**Graph**/G) is naturally structured as a bicategory, and we can expect in higher dimensions that we will have to consider weak higher categories, which are notoriously subtle and difficult to manipulate, so we had rather avoid them if we could.

This suggests considering a "category" with labeled graphs as 0-cells, cospans of labeled graphs as 1-cells and as 2-cells $f : (s, C, t) \Rightarrow (s', C', t')$, between 1-cells $(s, C, t) : A \to B$ and $(s', C', t') : A' \to B'$, triples (f_0, f_1, f_2) of isomorphisms making the following diagram commute

$$\begin{array}{ccccc} A & \xrightarrow{s} & C & \xleftarrow{t} & B \\ {\scriptstyle f_0}\downarrow & & \downarrow{\scriptstyle f_1} & & \downarrow{\scriptstyle f_2} \\ A' & \xrightarrow[s']{} & C' & \xleftarrow[t']{} & B' \end{array}$$

This data does not even form a bicategory, because we are considering 2-cells between 1-cells which do not have the same source or target. We will see in the next section, that the right structure for it is the notion of monoidal globular category introduced by Batanin.

10.2 The free category on a globular set

In this section, we generalize the construction of the free category on a graph to the free ∞-category on a globular set. As in the previous section, the representation we obtain is not necessarily the simplest or most efficient one, although it is reasonable (Batanin's trees [Bat98b, Str98] are arguably simpler for this task), but will extend to the case of polygraphs.

10.2.1 Globular sets

Globular sets generalize graphs in the sense that they allow edges between edges, edges between edges between edges, and so on.

Definition 254. The *globular category* $\bigcirc$ has integers as objects and morphisms are generated by $s_n, t_n : n \to n+1$ quotiented by the relations

$$s_{n+1} \circ s_n = t_{n+1} \circ s_n \qquad\qquad s_{n+1} \circ t_n = t_{n+1} \circ t_n$$

A presheaf G in $\hat{\bigcirc}$ is called a *globular set*, and an element of Gn is called an *n-cell* or an *n-generator*. Given $n \in \mathbb{N}$, we write $\bigcirc_n$ for the full subcategory whose objects are integers k with $k \leqslant n$; presheaves over this category are called *n-globular sets*. More generally, given a category $\mathcal{C}$, a functor $\bigcirc^{\mathrm{op}} \to \mathcal{C}$ is called a *globular object* in $\mathcal{C}$.

Notice that, because of the inclusion functor $\bigcirc_n \to \bigcirc_{n+1}$, every $(n+1)$-globular set has an underlying n-globular set. An $(n+1)$-globular set G thus consists of an n-globular set together with a set G_{n+1} of $(n+1)$-cells and suitable morphisms $s_n, t_n : G_{n+1} \to G_n$. This suggests defining an *n-generator* (i.e. an element of G_n) as a value of the type on the left:

```
type generator = {
    dim : int;
    label : generator option;
    source : generator;
    target : generator;
  }
```

```
let rec dummy = {
    dim = -1;
    label = None;
    source = dummy;
    target = dummy;
  }
```

An n-generator thus consists of a dimension (the integer n) and a source and target $(n-1)$-generator. For now, the field `label` can be ignored since it will always be filled with `None`. Since, with this type, every generator has to have a source and a target, for 0-generators we will use the convention that their source and target are always the "dummy (-1)-generator" shown on the right above. Invariants for the values of this type should be maintained by our library, such as the globular identities for a generator `g`:

```
g.source.source == g.target.source        g.source.target == g.target.target
```

Finally, an n-globular set can simply be implemented as a set of generators together with an underlying $(n-1)$-globular set (we do not need the source and target functions since every generator is equipped with its source and target):

```
type gset = {
    dim : int;
    generators : generator list;
    prev : gset;
  }
```

Again, we use a "dummy (-1)-globular set" as underlying globular set of a 0-globular set. Here also, invariants should be maintained by our library such as, for a globular set `gs` distinct from the dummy one:

```
gs.prev.dim + 1 = gs.dim
List.for_all (fun g -> g.dim = gs.dim) gs.generators
List.for_all (fun g -> List.memq g.source gs.prev.generators) gs.generators
```

In examples below, we will sometimes use assertions in order to ensure that this is the case.

Morphisms between n-globular sets can easily be defined in the same recursive way: a morphism $f : G \to H$ consists of a suitable function $f_n : G_n \to H_n$ between n-generators, and a morphism between the underlying $(n-1)$-globular sets of G and H. This suggests implementing morphisms as values of the following type:

```
type morphism = {
    dim : int;
    source : gset;
    target : gset;
    map : (generator,generator) Mapq.t;
    prev : morphism;
  }
```

The values of type `('a,'b) Mapq.t` encode set-theoretic finite functions from `'a` to `'b`: the module `Mapq` is similar to the module `Map` of the standard OCaml library, except that values are compared with physical equality instead of structural one (here, as explained in Section 10.1.1, physical equality is the only meaningful one between generators). The signification for the members of the records of type `morphism` should otherwise be pretty clear.

10.2.2 Basic functions on globular sets

Most of the expected category-theoretic functions on globular sets can be implemented with this data structure. In order to illustrate this, we will sketch here how coproduct, quotient and pushouts of globular sets can be implemented. As a first example, the forgetful functor which, with an $(n+1)$-globular set, associates the underlying n-globular set is easily defined on objects by

```
let prev gset = gset.prev
```

(and similarly on morphisms). This functor admits a left adjoint which allows one to see an n-globular set as an $(n+1)$- one by adjoining to it an empty set of $(n+1)$-cells. This functor can be implemented on objects by

```
let degenerate gset = {
    dim = gset.dim + 1;
    generators = [];
    prev = gset;
  }
```

In the following, we will use some simple functions such as the sequential composition of two morphisms. We will not detail their implementation:

```
seq : morphism -> morphism -> morphism
```

the application of a morphism to a generator

```
app : morphism -> generator -> generator
```

the inclusion of a globular set into a given bigger one

```
inclusion : gset -> gset -> morphism
```

the identity morphism

```
id : gset -> morphism
```

which is easily deduced from the previous function by `let id gs = inclusion gs gs`, and so on. Moreover, for clarity, the code we have presented up to now was simplified a bit, mostly in terms of notations. From now on, we will provide verbatim copies of the code as it is actually implemented. In practice, we do not access directly the members of records but use helper functions: in this way, the code will be easier to adapt if the data structure changes (which of course happened many times when elaborating the library, either because something was wrong or to improve the efficiency). For instance, we use the following function to retrieve the dimension of a globular set

```
let dim gset = gset.dim
```

and a new globular set is created using

```
let create ~generators ~prev () =
  let dim = dim prev + 1 in
  assert (List.for_all (fun g -> G.dim g = dim) generators);
  { dim; generators; prev }
```

notice that this also allows us to ensure that dimensions of generators did not get mixed up. The functions for manipulating generators (such as `G.dim` above) are in the module `G` and those for manipulating morphisms are in the module `M`.

Most functions are of course implemented in a recursive way. For instance, the non-disjoint union of two globular sets is computed by

```
let rec union s1 s2 =
  assert (dim s1 = dim s2);
  let dim = dim s1 in
  if dim < 0 then dummy else
    let prev = union (prev s1) (prev s2) in
    let generators = Listq.union (generators s1) (generators s2) in
    create ~generators ~prev ()
```

Notice that because we use physical equality in order to compare generators, it is easy to implement a function

```
copy : gset -> morphism
```

which, given a globular set G, creates a new globular set G' which is isomorphic to G but uses disjoint identifiers from G (i.e. $G_n \cap G'_n = \emptyset$ for every dimension n), and returns the isomorphism $G \cong G'$:

```
let rec copy s =
  let dim = dim s in
  if dim < 0 then M.dummy else
    let f' = copy (prev s) in
    let g_copy g =
      let source, target =
        if G.dim g = 0 then G.dummy, G.dummy
        else M.app f' (G.source g), M.app f' (G.target g)
      in
      G.create ~name:(G.name g) ~source ~target ()
    in
    let map = Mapq.of_list (List.map (fun g -> g, g_copy g) (generators s)) in
    let generators = List.map (Mapq.app map) (generators s) in
    let target = create ~generators ~prev:(M.target f') () in
    M.create ~map ~prev:f' ~source:s ~target ()
```

The categorical coproduct of two globular sets

```
coprod : gset -> gset -> morphism * morphism
```

can then be implemented as the non-disjoint union of two fresh copies of the globular sets. The function returns the two arrows of the cocone:

```
let coprod s1 s2 =
  let i1, i2 = copy s1, copy s2 in
  let s1', s2' = M.target i1, M.target i2 in
  let s = union s1' s2' in
  let i1', i2' = M.inclusion s1' s, M.inclusion s2' s in
  M.seq i1 i1', M.seq i2 i2'
```

Quotients of globular sets can be implemented in the same way. First a module for manipulating equivalence relations can easily be implemented, typically by representing an equivalence relation $R \subseteq X \times X$ on a set X as a set of pairs $(x, \hat{x})$ consisting of an element x and an arbitrarily chosen canonical representative $\hat{x}$ of the equivalence class of x. A *globular equivalence relation* on a globular set G, consists of a family $(R_n \subseteq G_n \times G_n)_{n\in\mathbb{N}}$ of equivalence relations which are compatible with source and target maps: for $x, y \in G_{n+1}$, $x\ R_{n+1}\ y$ implies $Gs_n(x)\, R_n\, Gs_n(y)$ and $Gt_n(x)\, R_n\, Gt_n(y)$. The quotient of a globular set by a globular equivalence relation can be implemented using the canonical representatives provided for equivalence classes of cells by the equivalence relation module. This can be used to compute the coequalizer of two morphisms `f1` and `f2` by

```
let coeq f1 f2 =
  assert (M.dim f1 = M.dim f2);
  assert (eq (M.source f1) (M.source f2));
  assert (eq (M.target f1) (M.target f2));
  let rec equiv f1 f2 =
    if M.dim f1 < 0 then Q.dummy else
      let r = equiv (M.prev f1) (M.prev f2) in
      let r = Q.degenerate ~set:(M.target f1) r in
      let r = List.fold_left (fun r g -> Q.add r (M.app f1 g) (M.app f2 g)) r
              (generators (M.source f1)) in
      List.fold_left (fun r g -> Q.add r g g) r (generators (M.target f1))
  in
  let e = equiv f1 f2 in
  let s = M.target f1 in
  Q.set e
```

where `Q` is the module implementing operations on globular equivalence relations. Finally, pushouts can be implemented using their usual expression in terms of coproduct and coequalizer:

```
let pushout f1 f2 =
  assert (M.dim f1 = M.dim f2);
  assert (eq (M.source f1) (M.source f2));
  let i1, i2 = coprod (M.target f1) (M.target f2) in
  let g = coeq (M.seq f1 i1) (M.seq f2 i2) in
  M.seq i1 g, M.seq i2 g
```

With slightly more work, universal maps (from a coproduct, a coequalizer or a pushout) can also be computed.

10.2.3 Labeled globular sets

There is an obvious functor n-**Cat** $\to \hat{\bigcirc}_n$, sending an n-category to the underlying n-globular set, and this functor admits a left adjoint sending a globular set G to the free n-category G^* it generates [Str98, Lei04]. As explained in the introduction, in order to describe the cells of G^*, we could have considered formal composites of generators, i.e. values in the type

```
type cell =
  | Generator of generator
  | Composition of cell * int * cell
  | Identity of cell
```

but these should have been considered modulo the theory of n-categories, which is very difficult to handle. Instead, we generalize the approach described in Section 10.1 for graphs.

We suppose fixed a signature n-globular set G, and our goal is to describe the free n-category G^* it generates. The idea is that an n-cell should be represented by a labeled globular set, i.e. an object in the slice category $\hat{\bigcirc}/G$ (those are easily implemented: we simply have to use the `label` field in the structure `generator`). Moreover, one needs to take into account the source and targets: each such globular set has a source and a target, which are $(n-1)$-globular sets, which themselves have source and target, and so on. This suggests considering n-cospans in the category $\hat{\bigcirc}/G$:

Definition 255 ([Bat98b])**.** We write $\mathbf{Oct}_n$ for the category corresponding to the poset whose elements are $([n] \times \{-,+\}) \sqcup \{n\}$, and such that n is a maximum element and $(i,\epsilon) < (j,\epsilon')$ if and only if $i < j$. An *n-span* in a category $\mathcal{C}$ is a functor $\mathbf{Oct}_n^{\mathrm{op}} \to \mathcal{C}$ and an *n-cospan* is an n-span in $\mathcal{C}^{\mathrm{op}}$.

In particular, a 0-cospan is an object of $\mathcal{C}$, a 1-cospan is a cospan in $\mathcal{C}$ (in the sense of Definition 251), and we have also shown below the diagrams corresponding to 2- and 3-cospans in $\mathcal{C}$ (all the squares commute):

$$
\begin{array}{ccccc}
A & & \begin{array}{ccc} & A & \\ \nearrow & & \nwarrow \\ A_0^- & & A_0^+ \end{array} & & \begin{array}{ccc} & A & \\ \nearrow & & \nwarrow \\ A_1^- & & A_1^+ \\ \uparrow & & \uparrow \\ A_0^- & & A_0^+ \end{array} \qquad \begin{array}{ccc} & A & \\ \nearrow & & \nwarrow \\ A_2^- & & A_2^+ \\ \uparrow & & \uparrow \\ A_1^- & & A_1^+ \\ \uparrow & & \uparrow \\ A_0^- & & A_0^+ \end{array}
\end{array}
\tag{10.4}
$$

Example 256. Consider the 2-globular set G with $G_0 = \{x, y\}$, $G_1 = \{f : x \to y, g : y \to y\}$ and $G_2 = \{\alpha : f \Rightarrow f\}$, which can be drawn as on the left

$$
x \xrightarrow{f} y \circlearrowright g \quad (\alpha) \qquad\qquad x_0 \overset{\overset{f_0}{\Downarrow \alpha_0}}{\underset{\underset{f_2}{\Downarrow \alpha_1}}{\xrightarrow{f_1}}} y_1 \xrightarrow{g_4} y_2 \xrightarrow{g_5} y_3
$$

The 2-cell $\mathrm{id}_g \circ_0 \mathrm{id}_g \circ_0 (\alpha \circ_1 \alpha)$ can be represented by the labeled globular set H pictured on the right above: its source H_1^- and target H_1^+ respectively correspond to the globular sets

$$
H_1^- = x_0 \xrightarrow{f_0} y_1 \xrightarrow{g_4} y_2 \xrightarrow{g_5} y_3 \qquad\qquad H_1^- = x_0 \xrightarrow{f_2} y_1 \xrightarrow{g_4} y_2 \xrightarrow{g_5} y_3
$$

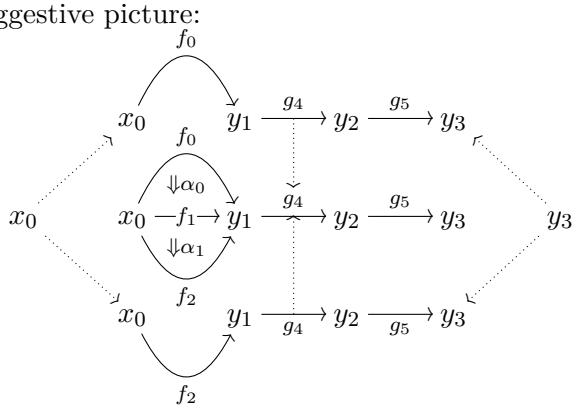
and their source H_0^- and H_0^+ respectively correspond to the globular sets

$$
H_0^- = x_0 \qquad\qquad H_0^+ = y_3
$$

The data corresponding to the 2-cell is thus, up to isomorphism, the 2-cospan on the left,

where the morphisms are the obvious inclusions, which can also be pictured as on the right:

If, on the picture on the right, we replace the objects by the corresponding globular sets, we obtain the following suggestive picture:



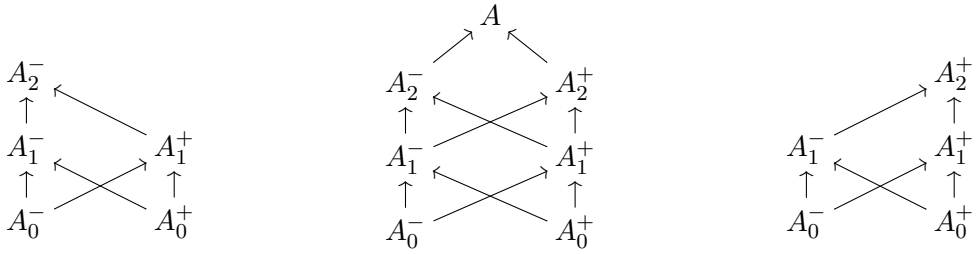
In practice, we can suppose that the morphisms of the n-cospans we consider are inclusions, so that we do not have to specify them, and a cell can be coded as

```
type cell = {
    dim : int;
    set : gset;
    source : cell;
    target : cell;
  }
```

We have the following alternative description of the base category for n-spans:

Lemma 257. *Given an integer n, the category $\mathbf{Oct}_n$ is isomorphic to the slice category $\bigcirc/n$.*

All the categories $\mathbf{Oct}_n$ are thus the images of the functor $O : \bigcirc \to \mathbf{Cat}$ defined on objects by $On = \bigcirc/n$ and in the expected way on morphisms. By post-composing with the functor $\hat{-} : \mathbf{Cat} \to \mathbf{Cat}^{\mathrm{op}}$ (we ignore size issues here), which, with a category $\mathcal{C}$, associates the category $\hat{\mathcal{C}}$ of covariant presheaves over it, we obtain a functor $\bigcirc \to \mathbf{Cat}^{\mathrm{op}}$, or equivalently a functor $\bigcirc^{\mathrm{op}} \to \mathbf{Cat}$, sending n to $\widehat{\bigcirc/n}$ which groups all the n-cospans as a globular object in $\mathbf{Cat}$. More prosaically, this says that one can define reasonable source and target maps for n-cospans. For instance, the 3-cospan on the middle below, already seen in (10.4), has the 2-cospans in the left and right as respective source and target:



Moreover, there are ways to compose n-cospans by pushout, generalizing the composition already seen in Section 10.1.4, giving rise to a monoidal globular category [Bat98b]: in the same way an ω-category is a globular object in **Set** equipped with suitable compositions, a monoidal globular category is a globular object in **Cat** equipped with suitable compositions. Given a globular object C in a category with pullbacks, $n \in \mathbb{N}$ and $k \in [n]$, we write $C(n) \times_k C(n)$ for the pullback

$$\begin{array}{ccc} C(n) \times_k C(n) & \dashrightarrow & C(n) \\ \vdots & & \downarrow{\scriptstyle C(s_k^n)} \\ C(n) & \xrightarrow[C(t_k^n)]{} & C(k) \end{array}$$

where $s_k^n : k \to n$ is defined as $s_{n-1} \circ \ldots \circ s_{k+1} \circ s_k$ and similarly for $t_k^n : k \to n$.

Definition 258 ([Bat98b])**.** A *monoidal globular category* $\mathcal{C}$, or MGC, consists of a globular object $\mathcal{C} : \bigcirc^{\mathrm{op}} \to \mathbf{Cat}$ together with

— *composition functors* $\circ_k^n : \mathcal{C}(n) \times_k \mathcal{C}(n) \to \mathcal{C}(n)$ for $n \in \mathbb{N}$ and $k \in [n]$,

— *identity functors* $\mathrm{Id}^n : \mathcal{C}(n) \to \mathcal{C}(n+1)$,

— suitable coherence natural isomorphisms,

satisfying suitable coherence conditions.

As a slight variant of the previous situation, one can construct a globular object $\mathcal{C}_G : \bigcirc^{\mathrm{op}} \to \mathbf{Cat}$ which, with n, associates the category of graded n-cospans of globular sets labeled in G and their isomorphisms. The structure of MGC allows for a simple account of renaming via the morphisms of the category $\mathcal{C}_G(n)$ of n-cells. By generalizing the construction described in (10.3), one can associate with any n-cell x of G an object $[\![x]\!]$ of $\mathcal{C}_G(n)$. For instance, to a 2-cell $\alpha : f \Rightarrow g : x \to y$ we could associate the following 2-cospan of globular sets labeled in G:

$$[\![\alpha]\!] \quad = \quad \begin{array}{ccccccc} & & & f_0 & & & \\ & & x_0 & \longrightarrow & y_2 & & \\ & \nearrow & & \vdots & & \searrow & \\ x_0 & & x_0 & \Downarrow \alpha_0 & y_2 & & y_2 \\ & \searrow & & \vdots & & \nearrow & \\ & & x_0 & \longrightarrow & y_2 & & \\ & & & g_1 & & & \end{array}$$

More generally, the n-cospan associated with an n-cell is induced by a globular set with one n-generator and two k-generators for each $k \in [n]$, along with its faces, which is easy to compute effectively. We write

```
of_generator : gset -> generator -> cell
```

for the corresponding function.

We conjecture that Proposition 252 can then be extended as follows. Given an MGC $\mathcal{C}$, we write $\overline{\mathcal{C}}$ for the category obtained by taking as n-cells equivalence classes of objects of $\mathcal{C}(n)$ under the relation identifying two objects related by a morphism.

Conjecture 259. *Given a globular set G, the free ω-category G^* it generates is isomorphic to the full subcategory of $\overline{\mathcal{C}_G}$ whose n-cells are generated by those of the form $[\![x]\!]$ for some n-generator $x \in G(n)$.*

Of course, with a slight variant of this construction, we should be able to construct the free n-category G^* on an n-globular set.

Given two cells which are isomorphic, we can inductively construct the isomorphism between the underlying globular sets (there is at most one isomorphism between them), and call the resulting function `identify`. Composition of cells, which is given abstractly by pushouts, can be implemented as follows:

```
let seq d c1 c2 =
  assert (dim c1 = dim c2 && 0 <= d && d < dim c1);
  let k = dim c1 - d in
  let t1, s2 = iterate k target c1, iterate k source c2 in
  let f = identify t1 s2 in
  let f1, f2 = S.pushout (target_morphism ~k c1)
               (M.seq (iterate k M.degenerate f) (source_morphism ~k c2)) in
  let s = M.target f1 in
  (* Recusive composition of the sources and targets. *)
  let rec seq c1 f1 c2 f2 = ... in
  let source, target =
    (if k = 1 then map (M.prev f1) (source c1)
     else seq (source c1) (M.prev f1) (source c2) (M.prev f2)),
    (if k = 1 then map (M.prev f2) (target c2)
     else seq (source c1) (M.prev f1) (source c2) (M.prev f2))
  in
  create ~set:s ~source ~target ()
```

It is satisfying to see that this code is almost a direct translation of the usual laws for composition in ω-categories.

10.3 Implementing polygraphs

We claim that the construction of the free n-category on an n-polygraph should be "roughly the same". We do will not detail much the implementation or the theory, but rather stress important points. However, the constructions are now much more intricate, because we need the definition of the free n-category on an n-polygraph in order to define an $(n+1)$-polygraph. This is why we believe that it is important to entirely understand the simpler case of globular sets first. As an illustration of this, here is our implementation of the data structures for generators, polygraphs, morphisms and cells: notice that they are all mutually recursive. Namely, the source of a generator is a cell which is a span of labeled polygraphs (and the labeling morphism associates a generator of the signature with a generator). However, the dimensions decrease suitably, which makes the structures well-founded.

```
type generator = {
    g_dim : int;
    g_source : cell; (** source (n-1)-cell *)
    g_target : cell; (** target (n-1)-cell *)
  }
and polygraph = {
    p_dim : int;
    p_generators : generator list; (** n-generators with source and target labeled
                                       in the underlying (n-1)-polygraph *)
    p_prev : polygraph; (** underlying (n-1)-dimensional polygraph *)
  }
and morphism = {
```

```
    m_map : (generator,generator) Mapq.t; (** function between top-dimensional generators *)
    m_prev : morphism; (** morphism between lower-dimensional cells *)
    m_source : polygraph; (** source of the map *)
    m_target : polygraph; (** target of the map *)
  }
and cell = {
    c_label : morphism; (** labeling morphism for the polygraph of the cell *)
    c_source : cell; (** source (n-1)-cell *)
    c_source_morphism : morphism; (** inclusion of the polygraph of the source cell *)
    c_target : cell; (** target (n-1)-cell *)
    c_target_morphism : morphism; (** inclusion of the polygraph of the target cell *)
  }
```

With this data structure, we were able to implement most basic manipulations such as coproducts and pushouts of polygraphs, the cell $[\![x]\!]$ corresponding to a generator x, etc. For instance, a 2-polygraph P with one 0-generator $*$, one 1-generator $a : * \to *$ and one 2-generator $\mu : a \circ_0 a \to a$ can be described as follows, along with the 2-cell $\mu \circ_1 (\mu \circ_0 \mu)$ of P^*, named here `mumumu`:

```
let uid = uidebug in
let p0 = P.degenerate P.dummy in
let star = G.create ~name:"*" ~source:C.dummy ~target:C.dummy () in
let p0 = P.add p0 star in
let star_c = C.of_generator p0 star in
let p1 = P.degenerate p0 in
let one = G.create ~name:"a" ~source:star_c ~target:star_c () in
let p1 = P.add p1 one in
let one_c = C.of_generator p1 one in
let p2 = P.degenerate p1 in
let two_c = C.seq 0 one_c one_c in
let mu = G.create ~name:"μ" ~source:(C.seq 0 one_c one_c) ~target:one_c () in
let p2 = P.add p2 mu in
let mu_c = C.of_generator p2 mu in
let mumumu = C.seq 1 (C.seq 0 mu_c mu_c) mu_c in
Printf.printf "mumumu:\n%s\n%!" (C.to_string uid mumumu)
```

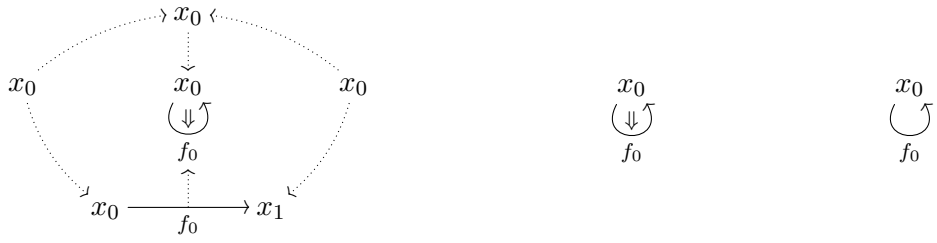
The output of the program then describes the cospan of labeled polygraphs corresponding to the constructed 2-cell:

```
mumumu:
2-cell:
    dim 0:
      *39(*0) *46(*0) *40(*0) *38(*0) *43(*0)
    dim 1:
      a37(a0) : *1(*46) → *1(*39)
      a39(a0) : *1(*40) → *1(*46)
      a41(a0) : *1(*40) → *1(*39)
      a33(a0) : *1(*43) → *1(*40)
      a35(a0) : *1(*38) → *1(*43)
      a42(a0) : *1(*38) → *1(*40)
      a32(a0) : *1(*38) → *1(*39)
    dim 2:
      μ6(μ0) : *9(*40) -a4(a39)→ *7(*46) -a5(a37)→ *8(*39)
               => *3(*40) -a1(a41)→ *2(*39)
      μ7(μ0) : *9(*38) -a4(a35)→ *7(*43) -a5(a33)→ *8(*40)
```

```
              => *3(*38) -a1(a42)→ *2(*40)
    μ8(μ0) : *9(*38) -a4(a42)→ *7(*40) -a5(a41)→ *8(*39)
              => *3(*38) -a1(a32)→ *2(*39)
1-source:
  *42(*0) -a36(a0)→ *44(*0) -a34(a0)→ *41(*0) -a40(a0)→ *47(*0) -a38(a0)→ *45(*0)
1-source morphism:
  *41↦*40 *42↦*38 *44↦*43 *45↦*39 *41↦*40 *47↦*46
  a34↦a33 a36↦a35 a38↦a37 a40↦a39
1-target:
  *3(*0) -a1(a0)→ *2(*0)
1-target morphism:
  *3↦*38 *2↦*39
  a1↦a32
```

A notation such as `*1(*46)` means a cell which has instance number `46` (for readability we generate such small numbers from memory locations) and is labeled by the cell with instance number `1`. The name `*` is a string which helps keeping track of what this generator refers to, but is not used except for printing purposes. Hopefully, the other notations are clear. Below, we list some important choices and limitations of the current implementation.

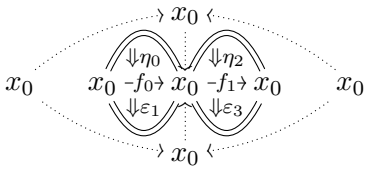
Faces are not included. Notice that, contrarily to globular sets, we cannot suppose that the cospan morphisms are simply inclusions so that we have to keep track of them (in `c_source_morphism` and `c_target_morphism`). As an illustration of this, consider a 2-category generated by a polygraph with a 0-generator x, a 1-generator $f : x \to x$, two 2-generators $\eta : \mathrm{id}_x \Rightarrow f$ and $\varepsilon : f \Rightarrow \mathrm{id}_x$. The 2-cell η is represented by the span of polygraphs $[\![\eta]\!]$ pictured on the left



where the target $x_0 \overset{f_0}{\to} x_1$ is not a sub-polygraph of the top-dimensional polygraph (in the middle). Moreover, it could not be the sub-polygraph pictured on the right, which is not linear in the sense of Section 10.1.4.

Isomorphism of cells. The composition of two cells f and g is performed by pushout, by identifying the target of f with the source of g. Since we are working up to renaming of cells, this means that we need an isomorphism from the target of f to the source of g. In the case of globular sets, this was not problematic because there was at most one, which was easy to compute. In the case of polygraphs, there can be multiple distinct isomorphisms (but the result should not depend on the choice of the isomorphism) and those are more difficult to compute (the currently implemented algorithm needs backtracking). For instance, the cell

corresponding to $(\varepsilon \circ_1 \eta) \circ_0 (\varepsilon \circ_1 \eta)$ is

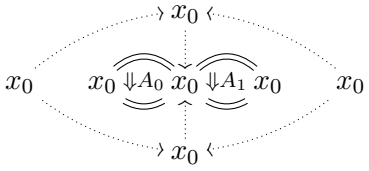


(the three "x_0" in the middle are the same 0-generator, drawn three times for the clarity of the figure). This cell admits two automorphisms: the trivial one (the identity) and the one performing a "mirror symmetry around a vertical axis in the middle", i.e. sending η_0 to η_2 and conversely, ε_1 to ε_3 and conversely, f_0 to f_1 and conversely, x_0 to x_0.

Quotienting by isomorphisms. Because of the presence of non-trivial automorphisms, considering cells up to isomorphism amounts to quotient them more than simply renaming of cells. This took us some time to figure out (it actually provides a counter-example to a conjecture in [Bur12, Section 4.2]), and is thus not currently taken into account in the implementation. Let us illustrate the problem on a simple example. Consider a polygraph P with

— one 0-generator x,
— no 1-generator,
— one 2-generator $A : \mathrm{id}_x \to \mathrm{id}_x$,
— two 3-generators $f : A \to A$ and $g : A \to A$.

Notice that the cell $A \circ_0 A : \mathrm{id}_x \Rightarrow \mathrm{id}_x : x \to x$, which corresponds to the cospan of labeled polygraphs



(10.5)

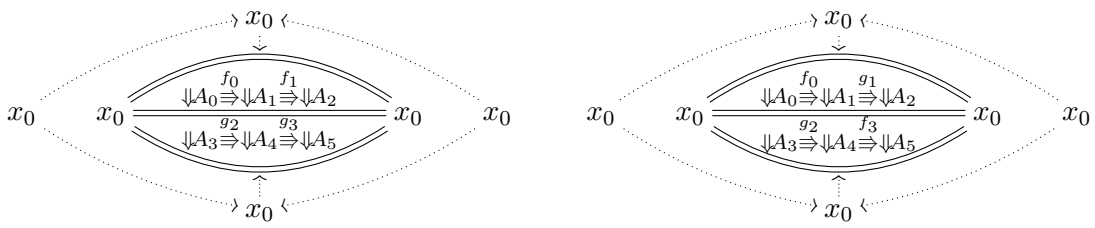
has a non-trivial isomorphism for similar reasons to before. Now, in the 3-category generated by this polygraph, we have $g \circ_0 f = f \circ_0 g$ by a typical Eckmann-Hilton type argument:

$$\begin{aligned} g \circ_0 f &= (g \circ_1 \mathrm{id}_{\mathrm{id}_x}) \circ_0 (\mathrm{id}_{\mathrm{id}_x} \circ_1 f) = (g \circ_0 \mathrm{id}_{\mathrm{id}_x}) \circ_1 (\mathrm{id}_{\mathrm{id}_x} \circ_0 f) \\ &= g \circ_1 f = (\mathrm{id}_{\mathrm{id}_x} \circ_0 g) \circ_1 (f \circ_0 \mathrm{id}_{\mathrm{id}_x}) = (\mathrm{id}_{\mathrm{id}_x} \circ_1 f) \circ_0 (g \circ_1 \mathrm{id}_{\mathrm{id}_x}) \\ &= f \circ_0 g \end{aligned}$$

and thus

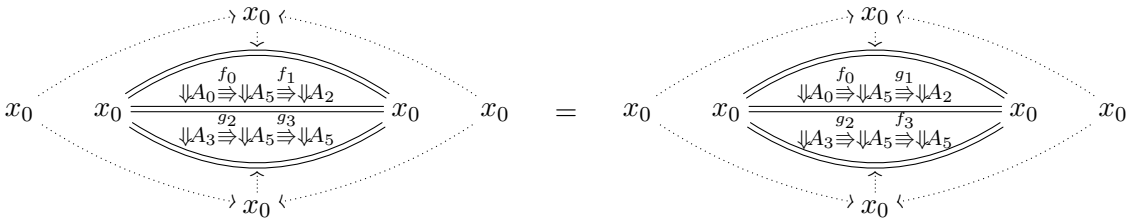
$$(g \circ_0 f) \circ_2 (g \circ_0 f) \quad = \quad (f \circ_0 g) \circ_2 (g \circ_0 f)$$

However, if we compute by pushout the two composites corresponding to the two sides of the above equality, we obtain the following cells:



(10.6)

For clarity, we have pictured only the principal polygraph (and not the whole span) and omitted drawing the 0-cell (there is only one in both cases). These are not isomorphic. The problem here comes from the fact that we cannot simply compose cospans by pushouts and quotient in the end, we have to quotient at each level. Here, the morphisms are obtained by composing on a face which is (isomorphic to) the 2-cell (10.5), which contains a non-trivial isomorphism by which we should quotient. With the notations of (10.6), the isomorphism exchanges A_1 and A_4 in both cases, and therefore those two 2-cells should be identified, i.e. the result of both compositions should in fact be



Both theoretical and practical properties of this representation of polygraphs remain to be studied in details.

Logographs. We would like to mention that a very similar construction has been investigated by Burroni when defining *logographs* [Bur12]; the precise links between the two constructions are left as future work. Logographs are polygraphic higher-dimensional generalizations of automata, and are defined using cospans of polygraphs, similarly to the representation that we use for cells. In fact, this notion should deserve the name of *higher-dimensional automaton*, the HDA recalled in Section 2.2.2 being the particular case where cells are of cubical shape.

Index

Bibliography

[AAD+93] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit. Atomic snapshots of shared memory. *J. ACM*, 40(4):873–890, September 1993.

[AM11] D. Ara and F. Métayer. The Brown-Golasiński model structure on strict ∞-groupoids revisited. *Homology, Homotopy and Applications*, 13(1):121–142, 2011.

[AMLH14] C. Angiuli, E. Morehouse, D. R. Licata, and R. Harper. Homotopical patch theory. In *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming*, pages 243–256. ACM, 2014.

[And93] J. H. Anderson. Composite registers. In *Conference on Principles of Distributed Computing*, pages 15–30. ACM, New York, 1993.

[AR94] J. Adámek and J. Rosicky. *Locally presentable and accessible categories*, volume 189. Cambridge University Press, 1994.

[Bat98a] M. Batanin. Computads for finitary monads on globular sets. *Contemporary Mathematics*, 230:37–58, 1998.

[Bat98b] M. Batanin. Monoidal Globular Categories As a Natural Environment for the Theory of Weak n-Categories. *Advances in Mathematics*, 136(1):39–103, 1998.

[BBC+97] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-C. Filliâtre, E. Gimenez, H. Herbelin, G. Huet, C. Munoz, C. Murthy, et al. The Coq proof assistant reference manual: Version 6.1, 1997.

[BBM11] D. Baelde, R. Beauxis, and S. Mimram. Liquidsoap: A High-Level Programming Language for Multimedia Streaming. In I. Černá, T. Gyimóthy, J. Hromkovič, K. Jefferey, R. Králović, M. Vukolić, and S. Wolf, editors, *SOFSEM 2011: Theory and Practice of Computer Science*, volume 6543 of *Lecture Notes in Computer Science*, pages 99–110. Springer Berlin Heidelberg, 2011.

[BBP99] M. A. Bednarczyk, A. M. Borzyszkowski, and W. Pawlowski. Generalized Congruences – Epimorphisms in Cat. *Theory and Applications of Categories*, 5(11):266–280, 1999.

[BCC+11] R. Bonichon, G. Canet, L. Correnson, É. Goubault, E. Haucourt, M. Hirschowitz, S. Labbé, and S. Mimram. Rigorous Evidence of Freedom from Concurrency Faults in Industrial Control Software. In *SAFECOMP*, pages 85–98, 2011.

[BCK+15] K. Bar, K. Casey, A. Kissinger, J. Vicary, and C. Wylie. Globular, 2015. `http://globular.science`.

[BCM12] O. Bouissou, A. Chapoutot, and S. Mimram. HySon: Set-based simulation of hybrid systems. In *Rapid System Prototyping (RSP), 2012 23rd IEEE International Symposium on*, pages 79–85, 2012.

[BCM14] O. Bouissou, A. Chapoutot, and S. Mimram. Set-based Simulation for Design and Verification of Simulink Models. In *Embedded Real Time Software and Systems (ERTS)*, 2014.

[BCSW83] R. Betti, A. Carboni, R. Street, and R. Walters. Variation through enrichment. *Journal of Pure and Applied Algebra*, 29(2):109–127, 1983.

[Bed88] M. A. Bednarczyk. *Categories of asynchronous systems.* PhD thesis, University of Sussex, 1988.

[BG93] E. Borowsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proc. of the 25th STOC*. ACM Press, 1993.

[BH09] M. R. Bridson and A. Haefliger. *Metric spaces of non-positive curvature*, volume 319. Springer, 2009.

[BH10] T. Balabonski and E. Haucourt. A geometric approach to the problem of unique decomposition of processes. In *Concurrency Theory 21th International Conference*, volume 6269 of *Lecture Notes in Computer Science*, pages 132–146. Springer, 2010.

[BKd03] M. Bezem, J. W. Klop, and R. de Vrijer. *Term rewriting systems.* Cambridge University Press, 2003.

[BLM15] F. Becker, R. Lepigre, and P.-É. Meunier. Pijul, 2015. `https://pijul.org/`.

[BM08] D. Baelde and S. Mimram. De la webradio lambda à la λ-webradio. In *JFLA (Journées Francophones des Langages Applicatifs)*, pages 47–62, Étretat, France, 2008. INRIA.

[BM11] R. Beauxis and S. Mimram. A Non-Standard Semantics for Kahn Networks in Continuous Time. In M. Bezem, editor, *Computer Science Logic (CSL'11) - 25th International Workshop/20th Annual Conference of the EACSL*, volume 12 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35–50, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[BMZ88] O. Biran, S. Moran, and S. Zaks. A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, pages 263–275. ACM, 1988.

[BN99] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1999.

[Bor94] F. Borceux. *Handbook of Categorical Algebra 1. Basic Category Theory.* Encyclopedia of Mathematics and its Applications. Cambridge Univ. Press, 1994.

[BR15] F. Benavides and S. Rajsbaum. The read/write protocol complex is collapsible. *arXiv preprint arXiv:1512.05427*, 2015.

[Bro87] K. S. Brown. Finiteness Properties of Groups. *Journal of Pure and Applied Algebra*, 44(1):45–75, 1987.

[BS72] E. Brieskorn and K. Saito. Artin-Gruppen und Coxeter-Gruppen. *Invent. Math.*, 17:245–271, 1972.

[Bur93] A. Burroni. Higher-dimensional word problems with applications to equational logic. *Theoretical computer science*, 115(1):43–62, 1993.

[Bur12] A. Burroni. Automates et grammaires polygraphiques. *Diagrammes*, 67:9–32, 2012.

[BW06] P. Bubenik and K. Worytkiewicz. A Model Category for Local PO-Spaces. *Homology, Homotopy and Applications*, 8(1):263–292, 2006.

[CEH+01] J. Cassaigne, M. Espie, F. Hivert, D. F. Krob, and J.-C. Novelli. The Chinese monoid. *Internat. J. Algebra Comput.*, 11(3):301–334, 2001.

[CGM12] A. Cain, R. Gray, and A. Malheiro. Finite Gröbner–Shirshov bases for plactic algebras and biautomatic structures for plactic monoids. Preprint, 16 pages, 2012.

[Cha90] S. Chaudhuri. Agreement is harder than consensus: set consensus problems in totally asynchronous systems. In *Proc. of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 311–334. ACM Press, August 1990.

[Che12] E. Cheng. A direct proof that the category of 3-computads is not cartesian closed. *arXiv preprint arXiv:1209.0414*, 2012.

[Cla09] E. M. Clarke. My 27-year quest to overcome the state explosion problem. In *Logic In Computer Science, 2009. LICS '09. 24th Annual IEEE Symposium on*, pages 3–3, August 2009.

[CM15] F. Clerc and S. Mimram. Presenting a Category Modulo a Rewriting System. In M. Fernández, editor, *26th International Conference on Rewriting Techniques and Applications (RTA 2015)*, volume 36 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 89–105, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Coh92] D. E. Cohen. A monoid which is right FP_∞ but not left FP_1. *Bulletin of the London Mathematical Society*, 24(4):340–342, 1992.

[Cra95] S. E. Crans. Pasting schemes for the monoidal biclosed structure on ω-cat. *Utrecht University, April*, 1995.

[Day70] B. Day. On closed categories of functors. In *Reports of the Midwest Category Seminar IV*, pages 1–38. Springer, 1970.

[DDG+15] P. Dehornoy, F. Digne, E. Godelle, D. Krammer, and J. Michel. *Foundations of Garside theory*, volume 22 of *EMS tracts in mathematics*. European Mathematical Society, 2015.

[Del72] P. Deligne. Les immeubles des groupes de tresses généralisés. *Invent. Math.*, 17:273–302, 1972.

[Del97] P. Deligne. Action du groupe des tresses sur une catégorie. *Invent. Math.*, 128(1):159–175, 1997.

[DGG15] J. Dubut, É. Goubault, and J. Goubault-Larrecq. Natural homology. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 171–183, 2015.

[Dij68a] E. W. Dijkstra. Cooperating sequential processes. In *Programming Languages: NATO Advanced Study Institute*, pages 43–112. Academic Press, 1968.

[Dij68b] E. W. Dijkstra. The Structure of the THE Operating System. *Communications of the ACM*, 11(15):341–436, 1968.

[DK94] G. Duchamp and D. Krob. Plactic-growth-like monoids. In *Words, languages and combinatorics, II (Kyoto, 1992)*, pages 124–142. World Sci. Publ., River Edge, NJ, 1994.

[DM97] V. Diekert and Y. Métivier. Partial commutation and traces. In *Handbook of formal languages*, pages 457–533. Springer, 1997.

[Dow09] A. B. Downey. *The Little Book of SEMAPHORES (2nd Edition): The Ins and Outs of Concurrency Control and Common Mistakes*. Createspace Independent Pub, 2009.

[DR95] V. Diekert and G. Rozenberg. *The Book of Traces*. World Scientific, Singapore, 1995.

[Dro90] M. Droste. Concurrency, automata and domains. In *Automata, Languages and Programming*, pages 195–208. Springer, 1990.

[DS93] M. Droste and R. Shortt. Petri nets and automata with concurrency relations—an adjunction. In *Sem. of Prog. Lang. and Model Theory*, pages 69–87. Gordon and Breach Science Publishers, Inc., 1993.

[Eil41] S. Eilenberg. Ordered Topological Spaces. *American Journal of Mathematics*, 63:39–45, 1941.

[Eng91] J. Engelfriet. Branching processes of petri nets. *Acta Informatica*, 28(6):575–591, 1991.

[Fah04] U. Fahrenberg. Directed homology. *Electr. Notes Theor. Comput. Sci.*, 100:111–125, 2004.

[Fah05] U. Fahrenberg. *Higher-Dimensional Automata from a Topological Viewpoint*. PhD thesis, Aalborg University, 2005.

[Faj03] L. Fajstrup. Dicovering spaces. *Homology, Homotopy and Applications*, 5(2):1–17, 2003.

[Faj05] L. Fajstrup. Dipaths and dihomotopies in a cubical complex. *Advances in Applied Mathematics*, 35(2):188–206, 2005.

[Faj11] L. Fajstrup. Erratum to 'Dicovering spaces'. *Homology, Homotopy and Applications*, 13(1):403–406, 2011.

[Faj14] L. Fajstrup. Trace spaces of directed tori with rectangular holes. *Mathematical Structures in Computer Science*, 24(02):e240202, 2014.

[FGH+12] L. Fajstrup, É. Goubault, E. Haucourt, S. Mimram, and M. Raussen. Trace Spaces: An Efficient New Technique for State-Space Reduction. In *ESOP*, pages 274–294, 2012.

[FGH+16] L. Fajstrup, É. Goubault, E. Haucourt, S. Mimram, and M. Raussen. *Directed Algebraic Topology and Concurrency*. Springer International Publishing, 2016.

[FGHR04] L. Fajstrup, É. Goubault, E. Haucourt, and M. Raussen. Components of the Fundamental Category. *Applied Categorical Structures*, 12(1):81–108, 2004.

[FGR98] L. Fajstrup, É. Goubault, and M. Raussen. Detecting Deadlocks in Concurrent Systems. *CONCUR'98 Concurrency Theory*, pages 332–347, 1998.

[Fis70] P. C. Fishburn. Intransitive indifference with unequal indifference intervals. *Journal of Mathematical Psychology*, 7(1):144–149, 1970.

[FL13] U. Fahrenberg and A. Legay. History-preserving bisimilarity for higher-dimensional automata via open maps. *Electr. Notes Theor. Comput. Sci.*, 298:165–178, 2013.

[FL14] U. Fahrenberg and A. Legay. Homotopy bisimilarity for higher-dimensional automata. arXiv preprint arXiv:1409.5865, 2014.

[FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[FR08] L. Fajstrup and J. Rosický. A convenient category for directed homotopy. *Theory and Applications of Categories*, 21(1):7–20, 2008.

[FRG06] L. Fajstrup, M. Raussen, and É. Goubault. Algebraic topology and concurrency. *Theoretical Computer Science*, 357(1):241–278, 2006.

[Gar69] F. Garside. The braid group and other groups. *Quart. J. Math. Oxford Ser.*, 20:235–254, 1969.

[Gau03] P. Gaucher. A model category for the homotopy theory of concurrency. *Homology, Homotopy and Applications*, 5(1):549–599, 2003.

[Gau05] P. Gaucher. Homological properties of non-deterministic branchings of mergings in higher dimensional automata. *Homology, Homotopy and Applications*, 7(1):51–76, 2005.

[Gau06] P. Gaucher. Inverting weak dihomotopy equivalence using homotopy continuous flow. *Theory and Applications of Categories*, 16(3):59–83, 2006.

[Gau08] P. Gaucher. Towards a homotopy theory of process algebra. *Homology, Homotopy and Applications*, 10(1):353–388, 2008.

[Gau10] P. Gaucher. Directed algebraic topology and higher dimensional transition systems. *New-York Journal of Mathematics*, to appear, 2010.

[Gau11] P. Gaucher. Towards a homotopy theory of higher dimensional transition systems. *Theory and Applications of Categories*, 25(12):295–341, 2011.

[Gau14a] P. Gaucher. Erratum to "towards a homotopy theory of higher dimensional transition systems". *Theory and Applications of Categories*, 29(2):17–20, 2014.

[Gau14b] P. Gaucher. Homotopy theory of labelled symmetric precubical sets. *New York Journal of Mathematics*, 20:93–131, 2014.

[GdLH+90] É. Ghys, P. de La Harpe, et al. *Sur les groupes hyperboliques d'apres Mikhael Gromov*, volume 83. Birkhäuser Basel, 1990.

[GG03] P. Gaucher and É. Goubault. Topological deformation of higher dimensional automata. *Homology, Homotopy and Applications*, 5(2):39–82, 2003.

[GGM15] S. Gaussent, Y. Guiraud, and P. Malbos. Coherent presentations of Artin monoids. *Compositio Mathematica*, 151(05):957–998, 2015.

[GH05] É. Goubault and E. Haucourt. A practical application of geometric semantics to static analysis of concurrent programs. In *Proceedings of CONCUR'05*, volume 3653 of *Lecture Notes in Computer Science*, pages 503–517. Springer, 2005.

[GH07] É. Goubault and E. Haucourt. Components of the Fundamental Category II. *Applied Categorical Structures*, 15(4):387–414, 2007.

[GHK09] É. Goubault, E. Haucourt, and S. Krishnan. Covering Space Theory for the Directed Topology. *Theory and Applications of Categories*, 22(9):252–268, 2009.

[GHK10] É. Goubault, E. Haucourt, and S. Krishnan. Future path-components in directed topology. In *MFPS*, volume 265 of *ENTCS*, pages 325–335, September 2010.

[GHM13] É. Goubault, T. Heindel, and S. Mimram. A Geometric View of Partial Order Reduction. *Proceedings of Mathematical Foundations of Programming Semantics, Electr. Notes Theor. Comput. Sci.*, 298:179–195, 2013.

[GJ92] É. Goubault and T. P. Jensen. Homology of higher-dimensional automata. In *Proc. of CONCUR'92*, Stonybrook, New York, August 1992. Springer-Verlag.

[GJ09] P. G. Goerss and J. F. Jardine. *Simplicial homotopy theory*, volume 174. Springer, 2009.

[GL13] J. Goubault-Larrecq. *Non-Hausdorff Topology and Domain Theory: Selected Topics in Point-Set Topology*, volume 22. Cambridge University Press, 2013.

[GM03] M. Grandis and L. Mauri. Cubical sets and their site. *Theory Appl. Categ*, 11(8):185–211, 2003.

[GM09] Y. Guiraud and P. Malbos. Higher-dimensional categories with finite derivation type. *Theory Appl. Categ.*, 22(18):420–478, 2009.

[GM12a] É. Goubault and S. Mimram. Formal relationships between geometrical and classical models for concurrency. *Electronic Notes in Theoretical Computer Science*, 283:77–109, 2012.

[GM12b] Y. Guiraud and P. Malbos. Coherence in monoidal track categories. *Math. Structures Comput. Sci.*, 22(6):931–969, 2012.

[GM12c] Y. Guiraud and P. Malbos. Higher-dimensional normalisation strategies for acyclicity. *Advances in Mathematics*, 231(3):2294–2351, 2012.

[GM13] Y. Guiraud and P. Malbos. Identities among relations for higher-dimensional rewriting systems. *Semin. Congr.*, 26:145–161, 2013.

[GM14] Y. Guiraud and P. Malbos. Polygraphs of finite derivation type. *arXiv preprint arXiv:1402.2587*, 2014.

[GMM+13] Y. Guiraud, P. Malbos, S. Mimram, et al. A homotopical completion procedure with applications to coherence of monoids. In *RTA-24th International Conference on Rewriting Techniques and Applications-2013*, volume 21, pages 223–238, 2013.

[GMT14] É. Goubault, S. Mimram, and C. Tasson. Iterated chromatic subdivisions are collapsible. *Applied Categorical Structures*, pages 1–42, 2014.

[GMT15] É. Goubault, S. Mimram, and C. Tasson. From geometric semantics to asynchronous computability. In *Proc. of DISC 2015*, volume 9363 of *LNCS*. Springer, 2015.

[GMT16] É. Goubault, S. Mimram, and C. Tasson. Geometric and Combinatorial Views on Asynchronous Computability. Submitted, 2016.

[God96] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.

[Gou95] É. Goubault. *The Geometry of Concurrency*. PhD thesis, Ecole Normale Supérieure, 1995.

[Gou96a] É. Goubault. Durations for truly-concurrent actions. In *Proceedings of ESOP'96*, number 1058, pages 173–187. Springer-Verlag, 1996.

[Gou96b] É. Goubault. The dynamics of wait-free distributed computations. Technical report, Research Report LIENS-96-26, December 1996.

[Gou96c] É. Goubault. A semantic view on distributed computability and complexity. In *Proceedings of the 3rd Theory and Formal Methods Section Workshop*. Imperial College Press, 1996.

[Gou97] É. Goubault. Optimal implementation of wait-free binary relations. In *Proceedings of the 22nd CAAP*. Springer Verlag, 1997.

[Gou01] É. Goubault. Labelled cubical sets and asynchronous transistion systems: an adjunction. In *Presented at CMCIM'02*, volume 2, page 2002, 2001.

[Gou03] É. Goubault. Some Geometric Perspectives in Concurrency Theory. *Homology, Homotopy and Applications*, 2003.

[GR02] É. Goubault and M. Raussen. Dihomotopy as a Tool in State Space Analysis. In *Proceedings of LATIN'02*, pages 16–37, 2002.

[Gra03] M. Grandis. Directed Homotopy Theory, I. The Fundamental Category. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 44(3):281–316, 2003.

[Gra04] M. Grandis. Inequilogical spaces, directed homology and noncommutative geometry. *Homology, Homotopy and Applications*, 6:413–437, 2004.

[Gra05] M. Grandis. The shape of a category up to directed homotopy. *Theory Appl. Categ*, 15(4):95–146, 2005.

[Gra09] M. Grandis. *Directed Algebraic Topology : Models of Non-Reversible Worlds*, volume 13 of *New Mathematical Monographs*. Cambridge University Press, 2009.

[Gro87] M. Gromov. *Hyperbolic groups*. Springer, 1987.

[Gro13] T. O. Group. Base Specifications, POSIX.1-2008, 2013. `http://pubs.opengroup.org/onlinepubs/9699919799/`.

[Gui04] Y. Guiraud. *Présentations d'opérades et systèmes de réécriture*. PhD thesis, Université Montpellier 2, 2004.

[Gui06] Y. Guiraud. Termination orders for 3-dimensional rewriting. *Journal of Pure and Applied Algebra*, 207(2):341–371, 2006.

[Gun94] J. Gunawardena. Homotopy and concurrency. In *Bulletin of the EATCS*, number 54, pages 184–193, October 1994.

[GZ67] P. Gabriel and M. Zisman. *Calculus of fractions and homotopy theory*. Springer, 1967.

[H+16] E. Haucourt et al. The ALCOOL Analyzer, 2016. See `http://www.lix.polytechnique.fr/alcool/`.

[Hag14] N. Hage. Finite convergent presentation for the plactic monoid for type C. *arXiv preprint arXiv:1412.0539*, 2014.

[Hat02] A. Hatcher. *Algebraic topology*. Cambridge Univ. Press, 2002.

[Hau05a] E. Haucourt. A framework for component categories. *ENTCS*, 2005.

[Hau05b] E. Haucourt. *Topologie Algébrique Dirigé et Concurrence*. PhD thesis, Université Paris 7 Denis Diderot / CEA LIST, October 2005.

[Hau06] E. Haucourt. Categories of Components and Loop-free Categories. *Theory and Applications of Categories*, 16(27):736–770, 2006.

[Hau12] E. Haucourt. Streams, d-spaces and their fundamental categories. *Electronic Notes in Theoretical Computer Science*, 283:111–151, 2012.

[HKR14] M. Herlihy, D. Kozlov, and S. Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Elsevier, 2014.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[Hol04] G. J. Holzmann. *The SPIN model checker: Primer and reference manual*, volume 1003. Addison-Wesley Reading, 2004.

[Hou12] R. Houston. *On editing text*, May 2012. `http://bosker.wordpress.com/2012/05/10/on-editing-text`.

[Hou14] R. Houston. *Revisiting "On editing text"*, June 2014. `https://bosker.wordpress.com/2014/06/19/revisiting-on-editing-text`.

[HS93] M. Herlihy and N. Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 111–120. ACM, 1993.

[HS96] T. T. Hildebrandt and V. Sassone. *Comparing transition systems with independence and asynchronous transition systems*. Springer, 1996.

[HS99] M. Herlihy and N. Shavit. The topological structure of asynchronous computability. *Journal of the ACM (JACM)*, 46(6):858–923, 1999.

[Hue80] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *Journal of the ACM (JACM)*, 27(4):797–821, 1980.

[HW08] J. Hayman and G. Winskel. The unfolding of general petri nets. In *FSTTCS*, volume 8, pages 223–234, 2008.

[Jac09] J. Jacobson. A formalization of darcs patch theory using inverse semigroups. Technical report, CAM report 09-83, UCLA, 2009.

[Kah13] T. Kahl. The homology graph of a higher dimensional automaton, 2013.

[KB70] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967)*, pages 263–297. Pergamon, Oxford, 1970.

[Kel82] M. Kelly. *Basic concepts of enriched category theory*, volume 64. Cambridge Univ. Press, 1982.

[KN85] D. Kapur and P. Narendran. A finite Thue system with decidable word problem and without equivalent finite canonical system. *Theoret. Comput. Sci.*, 35(2-3):337–344, 1985.

[Knu70] D. E. Knuth. Permutations, matrices, and generalized young tableaux. *Pacific J. Math.*, 34(3):709–727, 1970.

[Koz08] D. Kozlov. *Combinatorial algebraic topology*, volume 21. Springer, 2008.

[Koz12] D. Kozlov. Chromatic subdivision of a simplicial complex. *Homology, Homotopy and Applications*, 14(2):197–209, 2012.

[Koz15] D. N. Kozlov. Topology of the view complex. *Homology, Homotopy and Applications*, 17(1):307–319, 2015.

[Kri09] S. Krishnan. A Convenient Category of Locally Preordered Spaces. *Applied Categorical Structures*, 17(5):445–466, 2009.

[Kri13] S. Krishnan. Cubical approximation for directed topology I. *Applied Categorical Structures*, pages 1–38, 2013.

[LAA87] M. C. Loui and H. H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4, 1987.

[Laf95] Y. Lafont. A new finiteness condition for monoids presented by complete rewriting systems (after Craig C. Squier). *J. Pure Appl. Algebra*, 98(3):229–244, 1995.

[Laf03] Y. Lafont. Towards an algebraic theory of boolean circuits. *Journal of Pure and Applied Algebra*, 184(2):257–310, 2003.

[Lam79] L. Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Transactions on Computers*, 28(9):690–691, 1979.

[Law73] F. W. Lawvere. Metric spaces, generalized logic, and closed categories. *Rendiconti del seminario matématico e fisico di Milano*, 43(1):135–166, 1973.

[Lei04] T. Leinster. *Higher Operads, Higher Categories*. Number 298 in London Mathmatics Society Lecture Note Series. Cambridge University Press, 2004.

[Les84] P. Lescanne. Term rewriting systems and algebra. In *7th International Conference on Automated Deduction*, pages 166–174. Springer, 1984.

[Lév78] J.-J. Lévy. *Réductions correctes et optimales dans le lambda-calcul*. PhD thesis, Université Paris VII, 1978.

[Lit96] P. Littelmann. A plactic algebra for semisimple Lie algebras. *Adv. Math.*, 124(2):312–331, 1996.

[LM09] Y. Lafont and F. Métayer. Polygraphic resolutions and homology of monoids. *Journal of Pure and Applied Algebra*, 213(6):947–968, 2009.

[LMW10] Y. Lafont, F. Métayer, and K. Worytkiewicz. A folk model structure on omega-cat. *Advances in Mathematics*, 224(3):1183–1231, 2010.

[LP81] W. Lipski and C. H. Papadimitriou. A fast algorithm for testing for safety and detecting deadlocks in locked transaction. *ALGORITHMS: Journal of Algorithms*, 1981.

[LS81] A. Lascoux and M.-P. Schützenberger. Le monoïde plaxique. In *Noncommutative structures in algebra and geometric combinatorics (Naples, 1978)*, volume 109 of *Quad. "Ricerca Sci."*, pages 129–156. CNR, Rome, 1981.

[Lyn96] N. A. Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996.

[Mac98] S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 1998.

[Maz88] A. W. Mazurkiewicz. Basic notions of trace theory. *Lecture Notes In Computer Science; Vol. 354*, pages 285–363, 1988.

[Mel04] P.-A. Melliès. Asynchronous games 2: the true concurrency of innocence. In *Proceedings of the 15th CONCUR*, number 3170 in LNCS, pages 448–465. Springer Verlag, 2004.

[Mét83] Y. Métivier. About the rewriting systems produced by the knuth-bendix completion algorithm. *Information Processing Letters*, 16(1):31–34, 1983.

[Mét03] F. Métayer. Resolutions by polygraphs. *Theory and Applications of Categories*, 11(7):148–184, 2003.

[MG13] S. Mimram and C. D. Giusto. A Categorical Theory of Patches. *Electronic Notes in Theoretical Computer Science*, 298(0):283–307, 2013. Proceedings of the Twenty-ninth Conference on the Mathematical Foundations of Programming Semantics, MFPS XXIX.

[MG16] S. Mimram and É. Goubault. Directed Homotopy in Non-Positively Curved Spaces. Journal article in preparation., 2016.

[Mil89] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., 1989.

[Mim08] S. Mimram. *Sémantique des jeux asynchrones et réécriture 2-dimensionnelle*. PhD thesis, Université Paris Diderot – Paris VII, 2008.

[Mim10a] S. Mimram. Computing Critical Pairs in 2-Dimensional Rewriting Systems. In C. Lynch, editor, *RTA*, volume 6 of *LIPIcs*, pages 227–242, 2010.

[Mim10b] S. Mimram. Focusing in Asynchronous Games. In *CIE'10*, 2010. Accepted for publication.

[Mim11] S. Mimram. The structure of first-order causality. *Mathematical Structures in Computer Science*, 21(01):65–110, 2011.

[Mim14] S. Mimram. Towards 3-Dimensional Rewriting Theory. *Logical Methods in Computer Science*, 10(2):1–47, 2014.

[Mim15] S. Mimram. Presenting finite posets. In A. Middeldorp and F. v. Raamsdonk, editors, *Proceedings 8th International Workshop on Computing with Terms and Graphs, Vienna, Austria, July 13, 2014*, volume 183 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–17. Open Publishing Association, 2015.

[Min11] A. Miné. Static analysis of run-time errors in embedded critical parallel C programs. In *Proc. of the 20th European Symposium on Programming (ESOP'11)*, volume 6602 of *Lecture Notes in Computer Science (LNCS)*, pages 398–418. Springer, 2011.

[MM92] S. Mac Lane and I. Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer, 1992.

[MM07] P.-A. Melliès and S. Mimram. Asynchronous Games: Innocence without Alternation. In *Proceedings of the 18th International Conference on Concurrency Theory CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 395–411. Springer, Heidelberg, 2007.

[MMS97] J. Meseguer, U. Montanari, and V. Sassone. On the semantics of place/transition petri nets. *Mathematical Structures in Computer Science*, 7(04):359–397, 1997.

[Mor05] R. Morin. Concurrent automata vs. asynchronous systems. In *Mathematical Foundations of Computer Science*, pages 686–698. Springer, 2005.

[Muk92] M. Mukund. Petri nets and step transition systems. *International Journal of Foundations of Computer Science*, 3(4):443–478, 1992.

[Mul94] P. Mulry. Lifting theorems for Kleisli categories. In *Mathematical Foundations of Programming Semantics*, pages 304–319. Springer, 1994.

[MZ01] M. Makkai and M. Zawadowski. 3-computads do not form a presheaf category. *Personal letter to Michael Batanin*, 2001.

[Nac65] L. Nachbin. *Topology and Order*, volume 4 of *Mathematical Studies*. Van Nostrand (Princeton), 1965.

[New42] M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

[NPW79] M. Nielsen, G. D. Plotkin, and G. Winskel. Petri nets, event structures and domains. In *Semantics of Concurrent Computation*, pages 266–284, 1979.

[NPW81] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13(1):85–108, 1981.

[NSW94] M. Nielsen, V. Sassone, and G. Winskel. *Relationships between models of concurrency*. Springer, 1994.

[Oos94] V. V. Oostrom. Confluence by decreasing diagrams. *Theoretical computer science*, 126(2):259–280, 1994.

[Pap83] C. H. Papadimitriou. Concurrency control by locking. *SIAM Journal on Computing*, 12(2):215–226, 1983.

[Par73] R. Paré. Connected components and colimits. *Journal of Pure and Applied Algebra*, 3(1):21–42, 1973.

[Ped89] J. Pedersen. Morphocompletion for one-relation monoids. In *RTA*, volume 355 of *LNCS*, pages 574–578. Springer, 1989.

[Pel93] D. Peled. All from one, one for all: on model checking using representatives. In *Computer Aided Verification*, pages 409–423. Springer, 1993.

[Pet66] C. Petri. *Communication with automata*. PhD thesis, 1966.

[Pow91] J. Power. An n-categorical pasting theorem. In *Category theory*, pages 326–358. Springer, 1991.

[Pra91] V. R. Pratt. Modeling Concurrency with Geometry. *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 311–322, 1991.

[R+13] D. Roundy et al. *The Darcs Theory*, 2013. `http://darcs.net/Theory`.

[Rau10] M. Raussen. Simplicial models of trace spaces. *Alg. & Geom. Top.*, 10:1683–1714, 2010.

[Rau12a] M. Raussen. Execution spaces for simple higher dimensional automata. *Appl. Algebra Engrg. Comm. Comput.*, 23:59–84, 2012.

[Rau12b] M. Raussen. Simplicial models for trace spaces II: General Higher-Dimensional Automata. *Algebr. Geom. Topol.*, 12(3):1745–1765, 2012.

[RNRG96] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser. 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.

[Sch61] C. Schensted. Longest increasing and decreasing subsequences. *Canad. J. Math.*, 13:179–191, 1961.

[Shi85] M. W. Shields. Concurrent Machines. *The Computer Journal*, 28(5):449–465, 1985.

[Sim94] C. C. Sims. *Computation with finitely presented groups*, volume 48. Cambridge University Press, 1994.

[SNW94] V. Sassone, M. Nielsen, and G. Winskel. Relationships between models of concurrency. In *Proceedings of the Rex'93 school and symposium*, 1994.

[SNW96] V. Sassone, M. Nielsen, and G. Winskel. Models for concurrency: Towards a classification. *Theoretical Computer Science*, 170(1):297–348, 1996.

[SOK94] C. C. Squier, F. Otto, and Y. Kobayashi. A finiteness condition for rewriting systems. *Theoret. Comput. Sci.*, 131(2):271–294, 1994.

[Sor47] R. Sorgenfrey. On the topological product of paracompact spaces. *Bulletin of the American Mathematical Society*, 53(6):631–632, 1947.

[Squ87] C. C. Squier. Word problems and a homological finiteness condition for monoids. *J. Pure Appl. Algebra*, 49(1-2):201–217, 1987.

[Sta89] A. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64:221–269, 1989.

[Sta90] E. W. Stark. Connections between a concrete and an abstract model of concurrent systems. In *Mathematical Foundations of programming semantics*, pages 53–79. Springer, 1990.

[Str76] R. Street. Limits indexed by category-valued 2-functors. *Journal of Pure and Applied Algebra*, 8(2):149–181, 1976.

[Str98] R. Street. The role of michael batanin's monoidal globular categories. *Contemporary Mathematics*, 230:99–116, 1998.

[SW73] R. Street and R. Walters. The comprehensive factorization of a functor. *Bull. Amer. Math. Soc*, 79(2):936–941, 1973.

[SZ93] M. Saks and F. Zaharoglou. Wait-Free k-set Agreement is impossible: The Topology of Public Knowledge. In *Proc. of the 25th STOC*. ACM Press, 1993.

[Thu14] A. Thue. *Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln.* Skrifter utg. av Videnskapsselsk. i Kristiania. 1. Matem.-naturv. Klasse. 1914.

[Tie08] H. Tietze. Über die topologischen Invarianten mehrdimensionaler Mannigfaltigkeiten. *Monatsh. Math. Phys.*, 19(1):1–118, 1908.

[Tit81] J. Tits. A local approach to buildings. In *The geometric vein*, pages 519–547. Springer, 1981.

[Val89] A. Valmari. Stubborn sets for reduced state space generation. In *Applications and Theory of Petri Nets*, pages 491–515, 1989.

[vG91] R. van Glabbeek. bisimulations for higher dimensional automata. E-mail, available at `http://theory.stanford.edu/~rvg/hda`, June 1991.

[vG99] R. van Glabbeek. Petri nets, configuration structures and higher dimensional automata. *Lecture notes in computer science*, pages 21–27, 1999.

[vG06] R. van Glabbeek. On the expressiveness of higher dimensional automata. *Theoretical computer science*, 356(3):265–290, 2006.

[Whi39] J. H. C. Whitehead. Simplicial spaces, nuclei and m-groups. *Proceedings of the London mathematical society*, 2(1):243–327, 1939.

[Whi50] J. Whitehead. Simple homotopy types. *American Journal of Mathematics*, pages 1–57, 1950.

[Win86] G. Winskel. Event structures. In *Advances in Petri Nets*, pages 325–392, 1986.

[Win93] G. Winskel. *The formal semantics of programming languages: an introduction*. MIT press, 1993.

[WN95] G. Winskel and M. Nielsen. Models for concurrency. In *Handbook of logic in computer science (vol. 4)*, pages 1–148. Oxford University Press, 1995.

[Wor10] K. Worytkiewicz. Sheaves of ordered spaces and interval theories. *Journal of Homotopy and Related Structures*, 5(1):37–61, 2010.

[Yon54] N. Yoneda. On the homology theory of modules. *J. Fac. Sci. Univ. Tokyo. Sect. I*, 7:193–227, 1954.

[YPK79] M. Yannakakis, C. H. Papadimitriou, and H. T. Kung. Locking Policies: Safety and Freedom from Deadlock. In *20th Annual Symposium on Foundations of Computer Science*, pages 286–297, San Juan, Puerto Rico, 29–31 October 1979. IEEE.

[Yud15] I. Yudin. Decreasing diagrams and coherent presentations. *arXiv preprint arXiv:1512.00799*, 2015.

[Zie87] W. Zielonka. Notes on finite asynchronous automata. *Informatique théorique et applications*, 21(2):99–135, 1987.

[Zie15] K. Ziemiański. On execution spaces of PV-programs. 2015.