Basic Research in Computer Science

BRICS NS-00-3 Cousot et al. (eds.): GETCO '00 Preliminary Proceedings

Preliminary Proceedings of the Workshop on

Geometry and Topology in Concurrency Theory

GETCO '00

State College, USA, August 21, 2000

Patrick Cousot
Eric Goubault
Jeremy Gunawardena
Maurice Herlihy
Martin Raussen
Vladimiro Sassone
(editors)

BRICS Notes Series **NS-00-3**

ISSN 0909-3206 **August 2000**

See back inner page for a list of recent BRICS Notes Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `NS/00/3/`

*GE*ometry and *T*opology in *CO*ncurrency theory

Preliminary Proceedings of GETCO'2000
Satellite Workshop of Concur'2000
PennState, USA, August 21, 2000

Patrick Cousot
Eric Goubault
Jeremy Gunawardena
Maurice Herlihy
Martin Raussen
Vladimiro Sassone

Preliminary print under the patronage of
BRICS: Basic Research in Computer Science
Centre of the National Danish Research Foundation

*GE*ometry and *T*opology in *CO*ncurrency theory

Foreword

The main mathematical disciplines that have been used in theoretical computer science are discrete mathematics (especially, graph theory and ordered structures), logics (mostly proof theory for all kinds of logics, classical, intuitionistic, modal etc.) and category theory (cartesian closed categories, topoi etc.). General Topology has also been used for instance in denotational semantics, with relations to ordered structures in particular.

Recently, ideas and notions from mainstream "geometric" topology and algebraic topology have entered the scene in Concurrency Theory and Distributed Systems Theory (some of them based on older ideas). They have been applied in particular to problems dealing with coordination of multi-processor and distributed systems. Among those are techniques borrowed from algebraic and geometric topology: Simplicial techniques have led to new theoretical bounds for coordination problems. Higher dimensional automata have been modelled as cubical complexes with a partial order reflecting the time flows, and their homotopy properties allow to reason about a system's global behaviour.

This workshop aims at bringing together researchers from both the mathematical (geometry, topology, algebraic topology etc.) and computer scientific side (concurrency theorists, semanticians, researchers in distributed systems etc.) with an active interest in these or related developments.

It follows the first workshop on the subject "Geometric and Topological Methods in Concurrency Theory" which has been held in Aalborg, Denmark, in June 1999.

The Workshop has been financially supported by Hewlett Packard's Basic Research Institute in the Mathematical Sciences (Bristol, England) and the Basic Research Institute in Computer Science (Aarhus, Denmark), and I thank these institutions for this, and more specifically Jeremy Gunawardena and Uffe Engberg. I also wish to thank the referees, the authors and the programme committee members for their very precise and timely job. Many thanks are also due to Michael Mislove who kindly supported the workshop by letting us submit the papers through the Electronic Notes in Theoretical Computer Science. Last but not least, I wish to thank the Concur organizers, Catuscia Palamidessi and Dale Miller, and the Workshop coordinator, Uwe Nestmann, for making this possible.

Eric Goubault, the 28th of July 2000.

Programme

	Monday 21 August
8:45 - 09:00	Opening
9:00 - 10:00	*Tutorial*: Topology and Directed Topology by Martin Raussen
10:00 - 10:15	Coffee break
10:15 - 11:15	*Tutorial*: Geometry of Fault-Tolerant Distributed Systems by Maurice Herlihy
11:15 - 11:30	Coffee break
11:30 - 12:00	Concurrent Processes with Loops from a Geometric Viewpoint by Lisbeth Fajstrup and Stefan Sokolowski
12:00 - 12:30	*First Informal Discussion*
12:30 - 14:00	Lunch
14:00 - 14:30	A Study on Semi-Sheaves Associated to Transition Systems by Ana Isabel de Azevedo Spinola and Edward Hermann Haeusler
14:30 - 15:00	Synchronous Message-Passing and Topology by Maurice Herlihy, Mark Tuttle and Sergio Rajsbaum
15:00 - 15:15	Coffee break
15:15 - 15:45	From Concurrency to Algebraic Topology by Philippe Gaucher
15:45 - 16:15	Occurrence Counting Analysis for the π-calculus by Jerome Feret
16:15 - 16:30	Coffee break
16:30 - 17:10	*Conclusion*: Results, Perspectives and Open Problems by Eric Goubault
17:10 - 17:30	*Second Informal Discussion*

Contents

(Di)topology with applications to concurrency. A tutorial

Martin Raussen, Department of Mathematical Sciences
Aalborg University, Fredrik Bajersvej 7E
DK-9220 Aalborg Øst, Denmark
e-mail:raussen@math.auc.dk

July 27, 2000

1 Introduction

1.1 Topology in concurrency?

From a general perspective, concurrency theory is using many mathematical tools. Predominant are the use of graph theory (often labeled directed graphs) and of logics. Topology has also played a role. Many people talk about the topology of networks meaning nothing else than the graph determined by the connections in the netwok. General (or set-theoretic) topology has been applied in, e.g., fixed point theory, and systematically in connection with lattice theory in domain theory (work of D. Scott and al.; see [7] for a classical reference).

We shall proceed in a different direction: We want to give evidence for that also classical *algebraic topology* (with roots in mainly *geometric* problems) has a capacity of modelling concurrent processes and interesting phenomena attached to them – after a "twist".

1.2 Example: Progress graphs

The first "algebraic topological" seems to be that of a progress graph and has appeared in operating systems theory, in particular for describing the problem of "deadly embrace"[1] in "multiprogramming systems". Progress

[1] as E. W. Dijkstra originally put it in citeD68, now more usually called deadlock.

graphs are introduced in [1], but attributed there to E. W. Dijkstra. In fact they also appeared slightly earlier (for editorial reasons it seems) in [12].

The basic idea is to give a description of what can happen when several processes are modifying shared ressources. Given a shared resource a, we see it as its associated semaphore that rules its behaviour with respect to processes. For instance, if a is an ordinary shared variable, it is customary to use its semaphore to ensure that only one process at a time can write on it (this is mutual exclusion). Then, given n deterministic sequential processes $Q_1 \ldots, Q_n$, abstracted as a sequence of locks and unlocks on shared objects, $Q_i = R_1 a_i^1 . R_2 a_i^2 \ldots R_n^i a_i^{n_i}$ (R_k being P or V)[2], there is a natural way to understand the possible behaviours of their concurrent execution, by associating to each process a coordinate line in $\mathbf{R}^n$. The state of the system corresponds to a point in $\mathbf{R}^n$, whose ith coordinate describes the state (or "local time") of the ith processor.

1.2.1 Example

Consider a system with finitely many processes running altogether. We assume that each process starts at (local time) 0 and finishes at (local time) 1; the P and V actions correspond to sequences of real numbers between 0 and 1, which reflect the order of the P's and V's. The initial state is $(0, ..., 0)$ and the final state is $(1, ..., 1)$. An example consisting of the two processes $T_1 = P_a.P_b.V_b.V_a$ and $T_2 = P_b.P_a.V_a.V_b$ gives rise to the two dimensional progress graph of Fig. 1.2.1.

The shaded area represents states which are not allowed in any execution path, since they correspond to mutual exclusion. Such states constitute the forbidden area. An execution path in an n-dimensional progess graph in the unit square in $\mathbf{R}^n$ is a path from the initial state $(0, ..., 0)$ to the final state $(1, ..., 1)$ avoiding the forbidden area and increasing in each coordinate – time cannot run backwards. We call these paths directed paths or *dipaths*. This entails that paths reaching the states in the dashed square underneath the forbidden region, marked "unsafe" are deemed to deadlock, i.e. they cannot possibly reach the allowed terminal state which is $(1, 1)$ in dimension 2. Similarly, by reversing the direction of time, the states in the square above the forbidden region, marked "unreachable", cannot be reached from the initial state, which is $(0, 0)$ here. Also notice that all terminating paths above the forbidden region are "equivalent" in some sense, given that they are all characterized by the fact that T_2 gets a and b before T_1 (as far as resources are concerned, we call this a schedule). Similarly, all paths below

[2] Using E.W. Dijkstra's notation P and V [2] for respectively acquiring and releasing a lock on a semaphore.

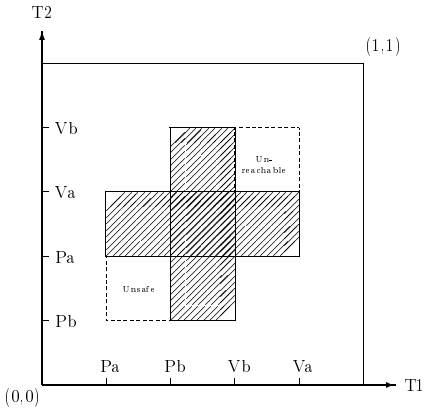


Figure 1: Example of a progress graph

the forbidden region are characterized by the fact that T_1 gets a and b before T_2 does.

1.3 Directed homotopy

In this picture, one can already recognize many ingredients that are at the center of the main problem of algebraic topology, namely the classification of shapes modulo "elastic deformation". As a matter of fact, the actual coordinates that are chosen for representing the times at which Ps and Vs occur are unimportant, and these can be "stretched" (preserving the order on the axes) in any manner, so the properties (deadlocks, schedules etc.) are invariant under some notion of deformation. A deformation of paths is called a *homotopy* in topology. Since directions (partial orders) are essential, we have to insist on that those are preserved under deformations. We call such an order preserving deformation of paths a directed homotopy or dihomotopy. Already for 2-dimensional progress graphs, this yields a different concept: Consider for instance the two homeomorphic shapes (deformable into each other by an elastic deformation) with two holes in Fig. 2 and Fig. 3 In Fig. 2, there are four essentially different dipaths up to dihomotopy (i.e. four schedules corresponding to all possibilities of accesses of resources a and b) whereas in Fig. 3, there are only three dipaths up to dihomotopy.

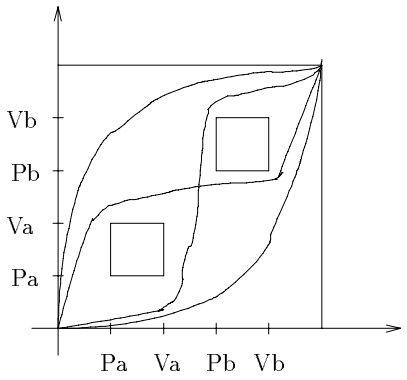


Figure 2: The progress graph corresponding to $P_a.V_a.P_b.V_b|P_a.V_a.P_b.V_b$

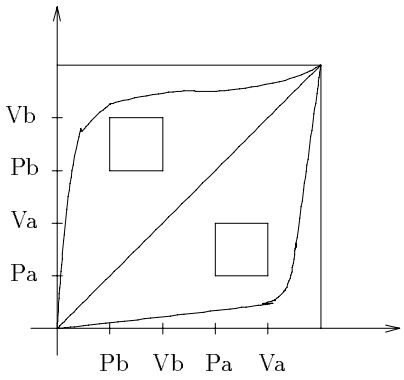


Figure 3: The progress graph corresponding to $P_b.V_b.P_a.V_a|P_a.V_a.P_b.V_b$

2 A short Tutorial in Topology

In this chapter, we touch upon central notions, methods and results from algebraic topology that have been applied or modified with applications in concurrency in mind – or where there should be a potential to do so in future work. Of course, these pages cannot replace a book; proofs are mainly omitted. There are lots of nice books at all levels on Algebraic Topology on the market; a nice one [9] is available on the internet.

2.1 Topological Spaces

Topological spaces are generalizations of metric spaces. They model *"nearness"* in more abstract situations. An axiomatic formulation makes use of *open subsets*. In a metric space X with distance function d, a subset $U \subseteq X$ is open if, for every $x \in X$ there is a positive real nmber $\varepsilon > 0$, such that $U_\varepsilon(x) = \{y \in X |\ d(x,y) < \varepsilon\} \subseteq U$.

Definition 2.1 *1. A topological space is a pair $(X,\mathcal{U})$ with $\mathcal{U} \subseteq 2^X$ a system of (open) subsets such that*

(a) $X, \emptyset \in \mathcal{U}$;

(b) Any union of open sets is open.

(c) Any finite *intersection of open sets is open.*

2. A subset $A \subseteq X$ is closed *if and only if its complement $X \setminus A$ is open.*

3. *Two points $x, y \in X$ can be* separated *if there are open sets $U_x, U_y \in \mathcal{U}$ such that $x \in U_x$, $y \in U_y$ and $U_x \cap U_y = \emptyset$.*

4. *A topological space such that any pair of points $x \neq y \in X$ can be separated is called* Hausdorff.

Example 2.2 *1. A metric space is Hausdorff.*

2. *A strange topology on $X = \mathbf{R}^2$ is given by :$U \subseteq X$ is open if and only if for every $(x, y) \in U$ there is an $\varepsilon > 0$ such that $]x \Leftrightarrow \varepsilon, x + \varepsilon[\times \mathbf{R} \subseteq U$. This is a topological space in which two points on the same verical line* cannot *be separated.*

3. *Many computer scientists are familiar with the* Scott topology, *which is not Hausdorff in general.*

Maps between topological spaces that preserve nearness are called continuous. They are generalizations of the continuous maps between metric spaces mapping points "sufficiently close" to each other into points close to each other. A neat formulation is:

Definition 2.3 *A map $f : X \to Y$ between two topological spaces X and Y is* continuous *if and only if $f^{-1}(U) \subseteq X$ is open for every open set $U \subseteq Y$.*

Example 2.4 *Let $X = \mathbf{R}^2$ be endowed with the topology from Ex. 2.2 and $Y = \mathbf{R}^2$ endowed with the (standard) topology inherited from the standard metric. The identity map $id : X \to Y$ is* not *continusous, whereas the identity map $id : Y \to X$ is continuous.*

Definition 2.5 *1. A map $f : X \to Y$ between two topological spaces X and Y is called a* homeomorphism *if it is a bijection and if both f and its inverse $f^{-1} : Y \to X$ are continuous.*

2. *Two topological spaces are called* homeomorphic *if and only if there exists a homeomorphism $f : X \to Y$.*

Example 2.6 *1. The open interval $]0, 1[$ is homeomorphic to the real half-line $]0, \infty[$ (both with standard topology inherited from the standard metric). A homeomorphism is given by the map $f :]0, 1[\to]0, \infty[$, $f(x) = \frac{x}{1-x}$ with $f^{-1}(y) = \frac{y}{1+y}$. In particular, a bounded and a non-bounded space can be homeomorphic.*

2. *A 2-dimensional sphere (boundary of a 3-dimensional ball) is homeomorphic to an ellipsoid, but not to a torus (doughnut).*

3. *The two topologies on* $\mathbf{R}^2$ *from Ex. 2.2.2 give rise to non-homeomorphic spaces. It is easy to see that a space homeomrphic to a Hausdorff space has to be Hausdorff again.*

Homeomorphy is an equivalence relation. From the topological point of view, one should not discriminate between two homeomorphic topological spaces from each other.

2.2 Paths

Let $I = [0,1]$ denote the unit interval with standard metric and topology, and let X denote a topological space. Any continuous map $\alpha : I \to X$ is called a *path* in X.

How can one compose paths? In general this is not possible. But if the endpoint $\alpha_1(1)$ of α_1 agrees with the start point $\alpha_2(0)$ of α_2, their *concatenation* $\alpha_1 * \alpha_2 : I \to X$ is defined by $(\alpha_1 * \alpha_2)(s) = \begin{cases} \alpha_1(2s), & t \leq \frac{1}{2} \\ \alpha_2(2s \Leftrightarrow 1), & t \geq \frac{1}{2}. \end{cases}$ (Both paths are pursued with "double speed"). Concatenation defines a (non-commutative, non-associative) monoidal structure on the path space $\mathcal{P}(X)$ of all paths on X. (OBS: Not all elements of $\mathcal{P}(X)$ can be composed with each other).

Definition 2.7 *A topological space* X *is called* path-connected *if and only if, for every pair of elements* $x_0, x_1 \in X$*, there exists a path* $\alpha : I \to X$ *with* $\alpha(0) = x_0$ *and* $\alpha(1) = x_1$*.*

2.3 Homotopy

What is a path in the space of maps between two topological spaces X and Y? Let again I denote the unit interval.

Definition 2.8 1. *A* homotopy *is a family* $H_t : X \to Y$*,* $t \in I$ *of maps, such that the associated map* $H : X \times I \to Y$ *is continuous.*

2. *Two continuous maps* $f, g : X \to Y$ *are homotopic if and only if there is a homotopy* $H : X \times I \to Y$ *such that* $H(x,0) = f(x)$ *and* $H(x,1) = g(x)$ *for all* $x \in X$*.*

Example 2.9 1. *Let* $S^1 = \{(x,y) | x^2 + y^2 = 1\} \subset \mathbf{R}^2$ *denote the unit circle. The map* $H : S^1 \times I \to \mathbf{R}^2$*,* $H((x,y),t) = (tx, ty)$ *is a homotopy between the constant map and the inclusion of the unit circle into* $\mathbf{R}^2$*.*

2. *There is no homotopy between the inclusion* $i : S^1 \to \mathbf{R}^2 \setminus \{(0,0)\}$ *in the pointed plane and any constant map* $c : S^1 \to \mathbf{R}^2 \setminus \{(0,0)\}$.

Homeomorphy is still a quite fine relation between topological spaces. It is in general quite difficult to find algebraic counterparts to help with a classification of certain spaces up to homeomorphism. The following relation is coarser and often easier to handle algebraically:

Definition 2.10 *1. A continuous map* $f : X \to Y$ *is called a* homotopy equivalence *if there are a continuous map* $g : Y \to X$ *and two homotopies between* $g \circ f : X \to X$ *and* id_X*, resp.* $f \circ g : Y \to Y$ *and* id_Y.

2. *Two spaces* X *and* Y *are called* homotopy equivalent *if and only if there is a homotopy equivalence* $f : X \to Y$.

Example 2.11 *1. The spaces* $X = S^1$ *and* $Y = \mathbf{R}^2 \setminus \{(0,0)\}$ *are homotopy equivalent (though of different dimension) via the inclusion map* $i : X \to Y$ *and the "contraction"* $c : Y \to X$ *with* $c(x,y) = (\frac{x}{x^2+y^2}, \frac{y}{x^2+y^2})$. *In fact,* $c \circ i = id_X$*; the map* $H : Y \times I \to Y$, $H((x,y),t) = (1 \Leftrightarrow t)(x,y) + t(\frac{x}{x^2+y^2}, \frac{y}{x^2+y^2})$ *defines a homotopy between* $id_Y (t = 0)$ *and* $i \circ c (t = 1)$.

2. *The spaces* $Z = \mathbf{R}^2$ *and* Y *(from above) are* not *homotopy equivalent, as will be shown in Sect. 2.5*

2.4 The fundamental group

2.4.1 Definitions

We shall now introduce the first algebraic construction associating to a topological space X a *group*. We shall make use of (some of the) paths considered in Sect. 2.2 "up to" a specific type of homotopy. More specifically: Let X denote a topological space, and let $x_0 \in X$ denote an (arbitrarily chosen) *basepoint*.

Definition 2.12 *1. A path* $\alpha : I \to X$ *is called a loop with basepoint* x_0 *if* $\alpha(0) = \alpha(1) = x_0$. *The set of loops with basepoint* x_0 *is denoted* $\mathcal{P}_1(X; x_0)$.

2. *Concatenation defines a binary operation* $C : \mathcal{P}_1(X; x_0) \times \mathcal{P}_1(X; x_0) \to \mathcal{P}_1(X; x_0)$.

3. *A* homotopy *of loops at* x_0 *is a family of loops* $H_t : I \to X$ *at* x_0 *such that the associated map* $H : I \times I \to X$, $H(x,t) = H_t(x)$ *is continuous.*

4. *Two loops α and β at x_0 are* homotopic *if there exists a homotopy H_t of loops with $H_0 = \alpha$ and $H_1 = \beta$. In that case, we write: $\alpha \simeq \beta$.*

It is essential that every path in the homotopy is a loop, i.e., that $H_t(0) = H_t(1)$ for all $t \in I$. Moreover, loops with the same basepoint can always be concatenated.

Example 2.13 *1. Let $X = \mathbf{R}^n$ and $x_0 \in \mathbf{R}^n$ any base point. Any two loops α, β at x_0 are homotopic via the linear homotopy $H_t = (1 \Leftrightarrow t)\alpha + t\beta$. The same result holds for a* convex *subset of $\mathbf{R}^n$, and even for a subset X that is* star-shaped *with respect to $x_0 \in X$, i.e., containing the line segment between x_0 and every $y \in X$.*

2. The same argument does not work for $Y_n = \mathbf{R}^n \setminus \{\mathbf{0}\}$. It turns out that two loops in Y_n are always homotopic for $n > 2$, but not always for $n = 2$.

3. A reparametrization *of a path (loop) α in X is a composition $\beta = \alpha \circ \varphi$ where φ is a continuous map with $\varphi(0) = 0$ and $\varphi(1) = 1$. Essentially, a reparametrization of α is a loop with the same base point running along the same trace as α, but possibly at another "speed".*

A loop α in X and every reparametrization $\beta = \alpha \circ \varphi$ are homotopic; a homotopy is given by $H_t(s) = \alpha((1 \Leftrightarrow t)\varphi(s) + ts)$.

Proposition 2.14 *1. The homotopy relation on paths with fixed base-point defines an equivalence relation. The set of equivalence classes is denoted $\pi_1(X; x_0)$.*

2. Concatenation factors over the homotopy relation and thus defines a binary operation. $C : \pi_1(X; x_0) \times \pi_1(X; x_0) \to \pi_1(X; x_0)$. We write $[\alpha] * [\beta]$ for $C([\alpha], [\beta])$.

3. $\pi_1(X; x_0)$ with the operation $*$ is a group.

Proof. (Sketch)

1. *Reflexivity*: Homotopy constant in t. *Symmetry*: $\tilde{H}(t) = H(1 \Leftrightarrow t)$. *Transitivity*: Concatenation of two homotopies H^1 and H^2 "in the parameter t": $H_t = \begin{cases} H^1(2t) & t \leq \frac{1}{2} \\ H^2(2t \Leftrightarrow 1) & t \geq \frac{1}{2}. \end{cases}$

2. Concatenation of two homotopies H^1 and H^2 "in the parameter s": $H_t = H^1_t * H^2_t$.

3. *Associativity:* $\alpha_1 * (\alpha_2 * \alpha_3)$ is a reparametrization of $(\alpha_1 * \alpha_2) * \alpha_3$. Use Ex. 2.13.3. Concatenation of any loop α at x_0 with the constant loop c (with $c(s) = x_0$ for all $s \in I$) yields a reparametrization of α; hence $[c]$ is a *two-sided identity* in $\pi_1(X; x_0)$. The inverse path to a path in X is defined by $\bar{\alpha}(s) = \alpha(1 \Leftrightarrow s)$. The path $\alpha^t(s) = \alpha(ts)$ runs from $\alpha(0)$ to $\alpha(t)$. For every $t \in I$, the concatenation $\alpha^t * \overline{\alpha^t}$ is a loop at x_0. Altogether, these maps define a homotopy of loops between $\alpha * \bar{\alpha}$ and $c = \alpha^0 * \overline{\alpha^0}$. Replacing α with $\bar{\alpha}$ yields a homotopy between $\bar{\alpha} * \alpha$ and c, i.e., $[\bar{\alpha}]$ is inverse to $[\alpha]$ in $\pi_1(X; x_0)$.

□

Example 2.15 *1. $\pi_1(\mathbf{R}^n, x_0)$ is the (one-element) trivial group.*

2. *The fundamental group of a circle is isomorphic to the integers. (To a loop on the circle, you may associate its winding number counting the total number of – directed – turns around the circle.)*
   The fundamental group of a higher-idmensional sphere is trivial.

3. *The fundamental group of a space is in general* not *commutative. The simplest example of a space with non-commutative fundamental group consisting of two circles with one common point. It turns out that the fundamental group of this space (with the common point as base point) is a* free group on two generators, *cf. Ex. 2.23.2.*

The definition of the fundamental group depends on the base point. But it is easy to see, that fundamental groups corresponding to two points x_0, x_1 in the space X are isomorphic, if there exists a path β from x_0 to x_1. A concrete isomorphism is given by $[\alpha] \to [\beta * \alpha * \beta^{-1}]$.

Remark 2.16 *The geometric shapes under consideration are usually uncountable, and so is the set of loops through a given point. The homotopy relation has two important effects: it reduces the cardinality to something typically discrete (finite or at most countable) and it imposes an algebraic (group) structure.*

2.4.2 Induced homomorphisms.

A continuous map $f : X \to Y$ induces a map $f_\# : \pi_1(X, x_0) \to \pi_1(Y, f(x_0))$ between the associated fundamental groups. The definition is easy: Associate to a loop α in X the loop $f \circ \alpha$ in Y; this map factors over the homotopy relation. Moreover, $f_\#$ is a *group homomorphism.*

Example 2.17 *Let $f : S^1 \to S^1$ denote the circle self-map, that "doubles angles", i.e., $f(\exp(it)) = \exp(2it)$. The winding number of the loop $f \circ \alpha$ is twice the winding number of the lopp α on S^1. Hence, $f_\# : \mathbf{Z} \cong \pi_1(S^1, 1) \to \pi_1(S^1, 1) \cong \mathbf{Z}$ corresponds to multiplication with 2.*

The following two properties of induced homomorphism are easy to derive, but essential:

1. let $f_1, f_2 : X \to Y$ denote *homotopic* maps from X to Y. Then, the induced maps $f \Leftrightarrow j\# : \pi_1(X; x_0) \to \pi_1(Y; f(x_0))$ coincide.

   Corollary 2.18 *Homotopy equivalent spaces have isomorphic fundamental groups.*

2. Let $g : Y \to Z$ denote another continuous map inducing the homomorphism $g_\# : \pi_1(Y; f(x_0)) \to \pi_1(Z; g(f(x_0)))$. The composite map $g \circ f : X \to Z$ induces the homomorphism $(g \circ f)_\# : \pi_1(X; x_0) \to \pi_1(Z; g(f(x_0)))$.

   Lemma 2.19 *The homomorphisms $(g \circ f)_\# = g_\# \circ f_\# : \pi_1(X; x_0) \to \pi_1(Z; g(f(x_0)))$ coincide.*

Generally speaking, we have the first example of a functor ("translator") that allows to associate to continuous geometric objects and their relations (topological spaces and continuous maps) discret algebraic counterparts. The aim is to allow geometric conclusions based on properties of these algebaic images.

2.5 Functoriality: an example

The following is to serve as an example how the translation mechanims from topology to algebra can serve to yield non-trivial topological reuslts. Let $B^n := \{\mathbf{x} \in \mathbf{R}^n | ||x|| \leq 1\}$ denote an n-dimensional ball, and $S^{n-1} = \partial B^n = \{\mathbf{x} \in \mathbf{R}^n | ||x|| = 1\}$ an $n \Leftrightarrow$1-dimensional sphere.

Theorem 2.20 ***(Brouwer's fixed point theorem)*** *Every continuous self-map $f : B^n \to B^n$ has a fixed point $x_0 \in B^n$ $(f(x_0) = 0)$.*

A proof for this theorem is elementary for $n = 1$. In that case, the continuous map $g : [0, 1] \to \mathbf{R}$, $g(x) = f(x) \Leftrightarrow x$ has the number 0 amongst its values since $g(\Leftrightarrow 1) \geq 0$ and $g(1) \leq 0$. For $n > 1$, it is a consequence of the following

Lemma 2.21 *There is no continuous map $F : B^n \to S^{n-1}$ extending the identity on S^{n-1}.*

Proof. The proof given here applies only to $n = 2$. For a proof in higher dimensions, one needs higher homotopy or homology groups cf. e.g. [9]):

Let $i : S^{n-1} \to B^n$ denote the continuous inclusion map. A map F as in the lemma would satisfy: $F \circ i = id$, the identity map on S^{n-1}. On the fundamental groups level (choose $x_0 \in S^{n-1}$), this amounts to

$$id_\# : \pi_1(S^{n-1}; x_0) = F_\# \circ i_\# : \pi_1(S^{n-1}; x_0) \to \pi_1(B^n; x_0) \to \pi_1(S^{n-1}; x_0).$$

Since B^n is convex (homotopy equivalent to a one-point space), we have $\pi_1(B^n; x_0) = 0$, and thus $id_\#$ has to be the zero-map. On the other hand, $id_\#$ is the identity map on $\pi_1(S^{n-1}; x_0)$. This is a contradiction for $n = 2$, where $\pi_1(S^1; x_0) \cong \mathbf{Z}$.

□

Proof. of Brouwer's fixed point theorem. Assume there is a continuous map $f : B^n \to B^n$ *without fixed point.* Then, one can construct a continuous map $F : B^n \to S^{n-1}$ by associating to x the intersection of the half-line starting at $f(x)$ through x with S^{n-1} (can be described by a formula using the solution of a quadratic equation and is thus continuous). Obviously, F restricts to the identity map on s^{n-1}. The existence of F contradicts Lemma 2.21.

□

The general idea is, that the (highly structured) discrete structure corresponding to a continuous structue is often easier to overlook than the original. Most often, the methods gives rise to *impossibility* results. In some cases, *existence* of objects or maps can be unveiled algebraically; this requires a proof that the vanishing of an algebraic obstruction is not only necessary, but indeed *sufficient* for the construction.

2.6 Compositions: The van Kampen theorem

The calculation of fundamental groups and of induced homomorphisms is difficult in general. One of the methods is a calculation "by recurrence", i.e., determining the fundamental group of a space by considering fundamental groups of subspaces and of relations between those. We look at the simplest case only:

Let $A_1, A_2 \subset X$ denote subsets each containing the base point x_0. Let $i_j : A_j \to X$, $i_{12} : A_1 \cap A_2 \to A_1$ and $i_{21} : A_1 \cap A_2 \to A_2$ denote the inclusion

maps. They satisfy: $i_1 \circ i_{12} = i_2 \circ i_{21} : A_1 \cap A_2 \to X$, and the obvious relations can be seen from the diagrams

$$\begin{array}{ccc} A_1 \cap A_2 & \longrightarrow & A_1 \\ \downarrow & & \downarrow \\ A_2 & \longrightarrow & X \end{array} \qquad \begin{array}{ccc} \pi_1(A_1 \cap A_2; x_0) & \longrightarrow & \pi_1(A_1; x_0) \\ \downarrow & & \downarrow \\ \pi_1(A_2; x_0) & \longrightarrow & \pi_1(X; x_0) \end{array}$$

From the fundamental groups of the pieces A_j, one can construct the free group $\pi_1(A_1; x_0) * \pi_1(A_2; x_0)$ generated by the two fundamental groups. It consists of all words in the two "alphabets". It contains the *normal subgroup* N generated by all words of type $i_{12}(\alpha) i_{21}(\alpha^{-1})$ with $\alpha \in \pi_1(A_1 \cap A_2; x_0)$.

Theorem 2.22 ***(van Kampen theorem)*** *Let $A_1, A_2 \subset X$ denote* path-connected *(cf. Def. 2.7)* open *subsets with* path-connected intersection $A_1 \cap A_2$. *The fundamental group $\pi_1(X; x_0)$ is then isomorphic to the quotient group of $\pi_1(A_1; x_0) * \pi_1(A_2; x_0)$ by the normal group N described above.*

A more categorical way to phrase van Kampen's theorem is as follows: The push-out diagram of spaces on the left-hand side of the diagram above is translated into a push-out diagram of groups on the right-hand side of that diagram.

Example 2.23 *1. It is essential that the intersection is path-connected, as well. The van Kampen theorem does thus* not *apply to the calculation of the fundamental group of the circle from the (trivial) fundamental groups of two half-circles (well, a bit more than a half to ensure openness of the pieces). The intersection consists of two "intervals" that cannot be connected by a path.*
On the other hand, the theorem shows that the fundamental group of an n-sphere S^n is trivial for $n > 1$: An n-sphere can be described as the union of two half-spheres, that are homeomorphic to n-dimensional balls with trivial fundamental groups. Their intersection is homotopy equivalent to an $(n \Leftrightarrow 1)$-dimensional sphere, which is path-connected for $n > 1$.

2. The fundamental group of the "one point union" of two subspaces (in which the base point has a neighborhood that is contractible (i.e., homotopy equivalent to a 1-point space) is the free product of the fundamental group of the subspaces.

2.7 Further topics

Higher homotopy groups Definition. Abelian groups. Difficult to determine. Results on spheres.

Particular topological spaces Simplicial complexes. CW-complexes. Approximation.

Simplicial homology Definition. Induced maps.

Singular homology Definition. Induced maps. Naturality. Homotopy invariance.

Mayer-Vietoris Homology of unions and intersections. Long exact sequence.

Functoriality Brouwer. Euclidean spaces up to homeomorphism.

3 A tutorial in ditopology

3.1 Introduction

Ditopology is not yet a well-established discipline. It presents our attempt to apply *methodology* from classical topology to the study of concurrency. The main difference compared to classical topology is, that we have to work with spaces and maps with an extra structure given by a *(local) partial order*. In the applications, the partial order reflects the time flow for the processors involved in the concurrent system under consideration.

Hence, we have to rephrase parts of the classical curriculum in topology in a category of partially ordered spaces and maps between them. The term *ditopology* (directed topology) was coined for this situation. It turns out, that this rephrasing is not just a dull exercise, and that the partial orders force you to invent notions that seem necessary for progressing with the applications – sometimes with help from neighbouring disciplines like, e.g., relativity theory.

Algebraic topology has been highly successful in deriving results about geometric structures that are *robust under deformations*. The key ingredient is very often an algebraization of the geometric structures to be considered and the use of *functoriality*, cf. Sect. 2.5. The introduction to [13] is a very readable account of our dream how this methodology might be applied in concurrency theory; moreover, it gives an elementary example (non-existence of a simulation), in which this dream actually works out.

We have to admit from the very beginning, that ditopology is not at all as advanced as classical topology is. The foundational definitions are still under debate, only few general results or calculations are achieved so far. Nevertheless, the few tools and results have shown to be useful in several applications; to mention

- An algorithm detecting deadlocks and associated safe/unsafe (and reachable/unreachable) regions for concurrent systems generalising the progress graphs studied in the introduction [3, 4, 6];
- Some results about the scheduling of actions [5];
- A topological underpinning of the result "2-phase locking is safe" used as a data engineering approach to ensure serialisability of procotcols for distributed databases [8, 5].

3.2 (Local) po-spaces

We start with elementary definitions and properties of po-spaces, cf. e.g. [7]:

Definition 3.1 *1. A* partial order $\leq$ *on a set* U *is a reflexive, transitive and antisymmetric relation. We write* $x < y$ *for* ($x \leq y$ *and* $x \neq y$).

2. A partial order $\leq$ *on a topological space* X *is called* closed *if* $\leq$ *is a closed subset (cf. Def. 2.1.2) of* $X \times X$ *in the product topology. If* $\leq$ *is closed, we call* $(X, \leq)$ *a* po-space.

In fact some studies have been made for other reasons in the mathematical litterature about such po-spaces, and in particular compact ones, see for instance [10].

Remark 3.2 *Let* $(X, \leq)$ *denote a po-space.*

1. *For every* $x \in X$, *the sets* $\downarrow x :== \{y \in X | y \leq x\}$ *and* $\uparrow x = \{y \in X | y \geq x\}$ *are closed.*
2. *For every pair of points* $y_1, y_2 \in X$, *the set* $[y_1, y_2] = \{x \in X \; y_1 \leq x \leq y_2\} = \downarrow y_2 \cap \uparrow y_1$ *is closed.*
3. *A po-space is Hausdorff[7].*

Example 3.3 *The progress graph , of a concurrent system modelling mutual exclusion from Sect. 1.2 can be considered as a po-space as follows:* $\mathbf{R}^n$ *is equipped with the partial order*

$$(x_1, \ldots, x_n) \leq (y_1, \ldots, y_n) \Leftrightarrow \; \forall 1 \leq i \leq n : \; x_i \leq y_i.$$

The progress graph , $\subset \mathbf{R}^n$ *inherits the partial order as a subspace.*

A loop cannot be given a consistent partial order: anti-symmetry will always be violated. But locally, "within the loop", there is still an order between the steps. We have thus to generalise our framework to include situations where a partial order only can be established *locally*:

Definition 3.4 *Let X be a topological space. A collection $\mathcal{U}(X)$ of pairs $(U, \leq_U)$ with partially ordered open subsets U covering X is a* local partial order *on X if for every $x \in X$ there is a nonempty open neighbourhood $W(x) \subset X$ such that the restrictions of $\leq_U$ to $W(x)$ coincide for all $U \in \mathcal{U}(X)$ with $x \in U$, i.e.,*

$$y \leq_{U_1} z \iff y \leq_{U_2} z \qquad \begin{array}{l} \text{for all } U_1, U_2 \in \mathcal{U}(X) \text{ such that } x \in U_i \\ \text{and for all } y, z \in W(x) \cap U_1 \cap U_2 \end{array}$$

A neighbourhood $W(x)$ with a well-determined partial order as above is called a po-neighbourhood *of x.*

Example 3.5 *The circle $S^1 = \{e^{i\theta} \in \mathbf{C}\}$ has a local partial order: the open subsets*

$$U_1 = \{e^{i\theta} \in S^1 | \, 0 < \theta < \frac{3\pi}{2}\} \quad \text{and} \quad U_2 = \{e^{i\theta} \in S^1 | \, \pi < \theta < \frac{5\pi}{2}\}$$

are (partially) ordered by the order on the θ's. Notice that the relation on S^1 generated by these local partial orders by taking the transitive closure is of no use: it is the trivial relation: $x \leq y$ for any pair of elements x, $y \in S^1$.

Remark 3.6

1. *In the applications, only processes without loops can be modelled by a partially ordered space. Processes allowing loops have to be modelled by locally partially ordered spaces.*

2. *It is necessary to define when two coverings by partially ordered subspaces define the* same *local partial order, cf. [5].*

3.3 Dimaps and Dipaths

Looking back at our example on progress graphs, we observe that executions correspond to paths (defined on a closed interval I with the usual order) in the partially ordered space *preserving that partial order.* A generalisation of this concept is as follows:

Definition 3.7 *Let $(X, \mathcal{U})$ and $(Y, \mathcal{V})$ be locally partially ordered spaces. A continuous map $f : X \to Y$ is called a* dimap *(directed map) if for any $x \in X$ there are po-neighborhoods $W(x)$ and $W(f(x))$ such that*

$$x_1 \leq_{W(x)} x_2 \Rightarrow f(x_1) \leq_{W(f(x))} f(x_2) \text{ whenever } x_1, x_2 \in f^{-1}(W(f(x))) \cap W(x)$$

It is not hard to see, that this definition does not depend on the choice of representative $\mathcal{U}$ of the equivalence class of local po-structures (cf. Rem. 3.6.2). In the case of po-spaces (not just local ones), dimaps are the same as monotone continuous maps. It is straightforward to see that local po-spaces and dimaps form a category.

A *dipath* is a dimap defined on either the unit interval I (relevant for paths in compact po-spaces; this is the approach used e.g. in [5, 11]), or, the half-line $\mathbf{R}_{\geq 0} := \{t \in \mathbf{R} | t \geq 0\}$ (relevant for paths in local po- spaces) – both with the usual order as the partial order relation $\leq$ on the domain. An execution where one process loops infinitely often corresponds to the exponential map $\varphi : \mathbf{R}_{\geq 0} \to S^1,\ \varphi(t) = exp(2\pi it)$ considered as a dipath; two processors looping infinitely many times can be modelled by a dipath into the 2-torus of type $\psi : \mathbf{R}_{\geq 0} \to T = S^1 \times S^1,\ \psi(t) = (\varphi(mt), \varphi(nt)),\ m, n > 0$.

Definition 3.8 *Let X denote a local po-space.*

1. *A* dipath *in X is a dimap $\alpha : \mathbf{R}_{\geq 0} \to X$.*

2. *We call α* finite *it there is a real number $T > 0$ such that αs restriction to $[T, \infty[$ is constant.*

3. *We call a dipath β in X an* extension *of a dipath α in X if there is a real number $T > 0$ and a surjective dimap $\varphi : [0, T[\to \mathbf{R}_{\geq 0}$ such that the diagram*

$$\begin{array}{ccc} [0,T[& \xrightarrow{\varphi} & \mathbf{R}_{\geq 0} \\ \downarrow{\scriptstyle \subset} & & \downarrow{\scriptstyle \alpha} \\ [0,\infty[& \xrightarrow{\beta} & X \end{array}$$

   commutes and such that βs restriction to $[T, \infty[$ is non-constant.

4. *A dipath $\alpha : \mathbf{R}_{\geq 0} \to X$ is called* inextendible *if it does not admit any extension $\beta : \mathbf{R}_{\geq 0} \to X$.*

5. *A new* local *partial order $\prec$ on X is defined as follows: $x \prec y \Leftrightarrow$ there is a finite dipath from x to y.*

6. *For $X_0, X_1 \subset X$, we define the dipath spaces*

$$\vec{P}_1(X; X_0, X_1) = \{\alpha : \mathbf{R}_{\geq 0} \to X \text{ finite } | \quad \alpha(0) \in X_0, \alpha(T) \in X_1 \text{ for large } T\} \text{ and}$$
$$\vec{P}_1(X; X_0, \infty) = \{\alpha : \mathbf{R}_{\geq 0} \to X \quad \text{inextendible } | \ \alpha(0) \in X_0\}.$$

Example 3.9 *1. In a finite mutual exclusion model (with PV semantics) the state space is $X = I^n \setminus int(F)$, the complement of the interior of the forbidden region F in a cube. $X_0 = \{\mathbf{0}\}$ consists of the initial point; X_1 will typically either contain only the final point $\mathbf{1}$ or be a (finite) set of (deadlock – cf, Def. 3.11) points. For such a compact po-space, it is a bit artificial to consider dipaths defined on $\mathbf{R}_{\geq 0}$; dipaths defined on a closed interval I give rise to an equivalent notion.*

2. Let $X = S^1$ denote a circle, or more general, $X = (S^1)^n$ denote an n-torus (with the product local partial order) modelling concurrent loops. In that case, the interesting dipaths are the non-finite ones. If a forbidden region is removed from X, finite dipaths ending in a deadlock arise naturally, as well.

We would like to have a clean definition for dimaps that send inextendible dipaths to inextendible dipaths in order to imitate the set-up of homotopy alluded to in Sect. 2. This is work in progress.

3.4 Dihomotopy

State spaces for concurrent systems tend to have an enormous (but finite) size. The main idea with the ditopology approach is to replace the finite state space by a continuous higher-dimensional (infinite) one, and then to impose relevant equivalent relations on the associated space of dipaths (and, as a result, on the state space itself), yielding classification patterns that apply to the original state space. As an effect, the number of *essentially different* states can usually be reduced drastically.

The relevant equivalence relation is given by a special type of homotopy:

Definition 3.10 *Let X denote a local po-space with subspaces $X_0, X_1 \subset X$. A continuous family $H_t : \mathbf{R}_{\geq 0} \to X$ of dipaths (giving rise to a homotopy $H : \mathbf{R}_{\geq 0} \times I \to X$) is called*

1. *a* dihomotopy *from X_0 to X_1, if every map $H_t \in \vec{P}_1(X; X_0, X_1)$ is a finite dipath from X_0 to X_1.*

2. *an inextendible* dihomotopy *from X_0 if every map $H_t \in \vec{P}_1(X; X_0, \infty)$ is an inextendible dipath from X_0. .*

These notions give rise to equivalence relations on the path spaces. Their quotient sets are denoted by $\vec{\pi}_1(X; X_0, X_1)$, resp. $\vec{\pi}_1(X; X_0, \infty)$.

Why is there any relation between dihomotopy and concurrency? In fact, dihomotopy of dipaths corresponds to the *commutativity* of local actions. Consider the following basic example: There are (essentially, i.e., up to reparametrization) two dipaths in the *boundary of a rectangle* from the "bottom" edge to the "top" edge. As dipaths in the

boundary: they are not dihomotopic (even not homotopic with end points fixed);

filled-in rectangle: they are dihomotopic (connect them linearly).

At least in dimension two, it is quite convincing, that execution paths in the mutual exclusion models discussed in the introduction yield equivalent results if they are dihomotopic (and that you can invent situations where they yield inequivalent results, if not). A theoretical classification of dipaths up to dihomotopy in 2-dimensionial mutual exclusion models and an algorithm determining the (finite) set $\vec{P}_1(X; \mathbf{0}, \mathbf{1})$ is described in [11].

The dihomotopy notion is certainly even more interesting and more promising – but also more involved in higher dimensions. Let us again consider the basic example, dipaths from the bottom to the top on the *boundary of a 3-dimensional cube*. This boundary models a piece of shared memory, that two, but not three processes can access and manipulate in a commutative way. In this simple case, it is elementary to see that any execution is equivalent to a serial one, and that all serial ones are equivalent – corresponding to the fact, that all dipaths in the model are dihomotopic to each other.

The following example of a space consisting of a cube from which 3 forbidden "bars" are removed, is a bit more involved and probably already quite difficult to analyse combinatorially: It describes 3 concurrent processes that access 3 shared objects, two of which can only handle one of them at any given time while the "middle" one can handle access of two of them in parallel. In this case, there exist five essentially different schedules corresponding to dihomotopy classes of dipaths. Two of those dipaths are in fact homotopic (with end points fixed), but *not di*homotopic. An example in which the schedules corresponding to those may lead to different results of a concurrent calculation is given in [5].

Which algebraic structures should one consider on top of the dihomotopy set $\vec{\pi}_1$. This question is not quite settled yet. The most promising so far is that of a partial order in non-published work of S. Sokołowski.

3.5 Deadlocks, unsafe and unreachable regions

We survey a fully-developped fast algorithm detecting *deadlocks, unsafe* and *unreachable regions* for mutual exclusion models. Though it does not use the

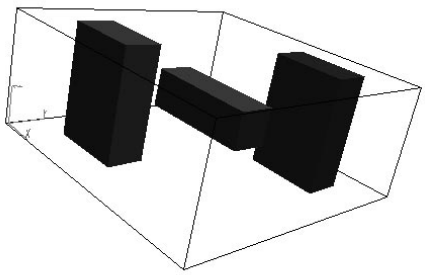


Figure 4: Room with 3 barriers

general framework for local po-spaces nor the notion of dihomotopy, it was conceived in the same geometrical spirit. Details can be found in [3, 4].

3.5.1 Definitions

In the applications, a *deadlock* is a state in which the system under consideration is blocked, i.e., there is no execution leaving that particular state. The associated *unsafe region* is the set of states that are bound to be blocked in that deadlock somewhere in the future. If executions are modelled by dipaths in (local) po-spaces, both notions (and their relatives) have counterparts with nice and clear definitions:

Definition 3.11 *1. An element $x \in X$ with $\uparrow x = \{x\}$ is called a deadlock. The set of all deadlocks in X is denoted by $\mathcal{D}(X)$. (Sometimes, a particular final state is exempted from $\mathcal{D}(X)$).*

2. The unsafe region $Uns(X; X_1) = U(X; X, X_1)$ *consists of all $x \in X$ that* cannot be connected *to any point in X_1 by a dipath, i.e.,*

$$\begin{aligned} Uns(X; X_1) &= \{x \in X | \vec{P}_1(X; x, X_1) = \emptyset\} = X \setminus (\downarrow X_1) \\ &= \{x \in X | (\uparrow x) \cap X_1 = \emptyset\}. \end{aligned}$$

3. The unreachable region $Unr(X; X_0) = U(X; X_0, X)$ *consists of all $x \in X$ that* cannot be reached *from any point in X_0 by a dipath, i.e.,*

$$\begin{aligned} Unr(X; X_0) &= \{x \in X | \vec{P}_1(X; X_0, x) = \emptyset\} \\ &= X \setminus (\uparrow X_0) = \{x \in X | (\downarrow x) \cap X_0 = \emptyset\}. \end{aligned}$$

Remark 3.12 *1. The symbols $\uparrow$ and $\downarrow$ above have to be interpreted with respect to the partial order $\prec$ from Def. 3.8.*

2. Consider the po-space associated to a PV-program discussed in Sect. 1.2 with final state **1**. *Then $Uns(X;\mathbf{1})$ corresponds exactly to the unsafe region of those states that can only reach a deadlock (different from* **1***).*

3.5.2 Detection of deadlocks and unsafe areas for mutual exclusion models

Unsafe and unreachable regions can be algorithmically determined in the PV model [4] and we recap here the basic idea of the algorithm.

Suppose the semantics of a PV program is given in terms of a forbidden region $F \subset I^n$ in a hypercube containing forbidden *hyperrectangles* $R^i = \prod_{j=1}^n [a_j^i, b_j^i] \subset I^n$ (with $n \geq 2$). Each of those hyperrectangles models a region that only a limited number of processes can enter simultaneously. We assume moreover that the coordinates a_j^i are pairwise different for every $1 \leq j \leq n$ (geometrically, this is a genericity assumption). The relevant state space is $X = I^n \setminus int(F)$.

For any nonempty index set $J = \{i_1, \ldots, i_k\}$ define

$$R^J = R^{i_1} \cap \cdots \cap R^{i_k} = [a_1^J, b_1^J] \times \cdots \times [a_n^J, b_n^J]$$

with $a_j^J = \max\{a_j^i | i \in J\}$ and $b_j^J = \min\{b_j^i | i \in J\}$. This set is again an $n\Leftrightarrow$rectangle unless it is empty (if $a_j^k > b_j^l$ for some $1 \leq j \leq n$ and $k, l \in J$). Let $\mathbf{a}^J = [a_1^J, \ldots, a_n^J] = \min R^J$ denote the *minimal* point in that hyperrectangle.

For every $1 \leq j \leq n$, we choose $\widetilde{a_j^J}$ as the "second largest" of the $a_j^{i_l}$, i.e., $\widetilde{a_j^J} = a_j^{i_s}$ with $a_j^{i_l} \leq a_j^{i_s} < a_j^J$ for all $a_j^{i_l} \neq a_j^J$, and consider the associated hyperrectangle $U^J = [\widetilde{a_1^J}, a_1^J] \times \cdots \times [\widetilde{a_n^J}, a_n^J]$ "below" R^J, the interior of which is unsafe with respect to $\mathbf{a}^J$. Usually, it models a large number of "states"; this is where we exploit higher-dimensionality.

Deadlock points in the interior of I^n are then exactly the minimal points $\min R^J$ of intersections with index set J *of cardinality* n (the number of processes, i.e. the dimension of the geometric shape we are studying) such that $R^J \neq \emptyset$ and $\min R^J$ not contained in any R^i with $i \notin J$. Deadlock points on the boundary ∂I^n can be found using the same recipe after modification of the hyperrectangles used in the description (cf. [3, 4]).

This description allows to find the set $\mathcal{D}$ of deadlocks in X and, for every deadlock $\mathbf{a} \in \mathcal{D}$ corresponding to a set of indices $J_\mathbf{a}$, the unsafe hyperrectangle $U^{J_\mathbf{a}}$ "just below". To detect the *entire* unsafe region, let

$F_1 = F \cup \bigcup_{\mathbf{a}\in\mathcal{D}} U^{J_\mathbf{a}}$. Find the set $\mathcal{D}_1$ of deadlocks in $X_1 = X \setminus int(F_1) \subset X$, and, for every deadlock $\mathbf{a} \in \mathcal{D}_1$, the unsafe corresponding hyperrectangle $U^{J_\mathbf{a}}$. Let $F_2 = F_1 \cup \bigcup_{\mathbf{a}\in\mathcal{D}_1} U^{J_\mathbf{a}}$ etc. (see Fig. 5 – 8 for an example).

The algorithm stops after a finite number l of loops ending with a set $U = F_l$ and such that $X_l = X \setminus int(U)$ does no longer contain any deadlocks. The set U consists precisely of the forbidden and of the unsafe points.

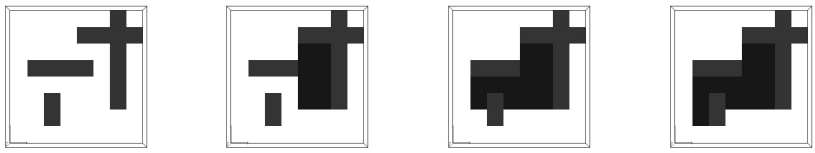


Figure 5: The forbidden region

Figure 6: First step of the algorithm

Figure 7: Second step of the algorithm

Figure 8: Last step of the algorithm

Literally the same algorithm willl find the *unreachable* regions after a time reversal (reflection in the barycenter of I^n).

3.6 Further topics

Application 2-phase locked protocols

Related concepts Dicoverings. Po-structure. Homotopy history. Dicomponents.

Models: Cubical complexes with local partial order

More structure higher dihomotopy, structure(s), dihomology, functoriality and applications, relations to classical paradigma in concurrency

References

[1] E.G. Coffman, M.J. Elphick, and A. Shoshani, *System deadlocks*, Comput. Surveys **3** (1971), no. 2, 67 – 78.

[2] E.W. Dijkstra, *Co-operating sequential processes*, Programming Languages (F. Genuys, ed.), Academic Press, New York, 1968, pp. 43–110.

[3] L. Fajstrup, É. Goubault, and M. Raussen, *Detecting Deadlocks in Concurrent Systems*, DTA/LETI/DEIN/SLA 98-61, LETI (CEA - Technologies Avancées), Saclay, France, August 1998, 25 pp.

[4] ———, *Detecting Deadlocks in Concurrent Systems*, CONCUR '98; Concurrency Theory (Nice, France) (D. Sangiorgi and R. de Simone, eds.), Lect. Notes Comp. Science, vol. 1466, Springer-Verlag, September 1998, 9th Int. Conf., Proceedings, pp. 332 – 347.

[5] ———, *Algebraic topology and concurrency*, Tech. Report R-99-2008, Department of Mathematical Sciences, Aalborg University, DK-9220 Aalborg Øst, June 1999.

[6] Lisbeth Fajstrup, *Loops, ditopology, and deadlocks*, Tech. Report R-99-2023, Department of Mathematical Sciences, Aalborg University, DK-9220 Aalborg Øst, 1999, 30 pages, to appear in Math. Structures Comput. Sci.

[7] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D.S. Scott, *A Compendium of Continuous Lattices*, Springer-Verlag, 1980.

[8] J. Gunawardena, *Homotopy and concurrency*, Bulletin of the EATCS **54** (1994), 184–193.

[9] Alan Hatcher, *Algebraic Topology*, to appear at Cambridge University Press; currently available at http://www.math.cornell.edu/ hatcher/# AT1, 2000.

[10] L. Nachbin, *Topology and order*, Van Nostrand, Princeton, 1965.

[11] M. Raussen, *On the classification of dipaths in geometric models for concurrency*, Tech. Report R-99-2025, Dept. of Mathematics, Aalborg University, Aalborg, Denmark, 1999, 42 pages, to appear in Math. Structures Comput. Sci.

[12] A. Shoshani and E.G. Coffman, *Sequencing tasks in multiprocess systems to avoid deadlocks*, Eleventh Annual Symposium on Switching and Automata Theory (Santa Monica, CA, USA), no. 225 - 235, IEEE, 1970.

[13] S. Sokołowski, *Investigation of concurrent processes by means of homotopy functors*, Manuscript. Kansas State University. Submitted to Math. Structures Comput. Sci., August 1999.

Infinitely running concurrent processes with loops from a geometric viewpoint

Lisbeth Fajstrup [1]

Department of Mathematical Sciences, Aalborg University
Fredrik Bajers Vej 7E, DK-9220 Aalborg Ø, Denmark
e-mail: fajstrup@math.auc.dk.

Stefan Sokołowski [2]

Institute of Computer Science, Polish Academy of Sciences
Gdańsk Div., ul. Abrahama 18, 81-825 Sopot, Poland
e-mail: stefan@ipipan.gda.pl.

Abstract

This report gives a formal topological semantics to inductively defined concurrent systems and investigates the properties of such systems. We allow loops and infinitely running computations, which is new in the topological investigations of concurrency. In this more general setting, we prove the equivalent to the result from [2] that deadlocks and unsafe points can be found using a finite number of deloopings.

1 Introduction

The idea of using geometric methods for concurrency is not new. The geometric viewpoint referred to in the title goes back to Dijkstra [1], who introduces higher dimensional geometric objects, progress graphs, and abstracts a process to be a series of actions locking and releasing a set of resources, which may then be shared with other processes thus giving rise to coordination problems. This idea has later been refined or independently reinvented by a number of authors. For an overview see, for instance, [3].

Concurrent systems, as most things in computer science, operate in discrete time. One way of applying geometric and topological methods is to come up

[1] Partially supported by the Danish National Science Research Council.
[2] Supported by the Dept. of Computing and Information Sciences, Kansas State University, and by the ICS PAS.

with a discrete counterpart of such notions as connectedness or homotopy, as do, for instance, [6,9]. An alternative approach is to assign topological spaces to concurrent systems and to work directly in topology [7,3,4,2]. One contribution of this paper is to make such an assignment explicit. Concurrent systems are defined in the computer science tradition as syntax objects and a "topological semantics" is defined by structural induction on the syntax. Certain good properties of that semantics are proved.

Along with concurrent system computations, that begin in a certain time point and end in another, we are studying computations which may run for ever, such as operating systems. In such systems, termination can only happen when something goes wrong. In computer science applications (unlike in physics — cf. [8]), time does not run from $-\infty$ to $+\infty$. There is a beginning but no ending; the past is finite, while the future is not. In other words, at a certain well-defined point in time all the processes are started off and never stopped again. Our formalism covers infinite computations in both discrete and topological settings.

In [4], geometric methods were used to develop an algorithm for detecting deadlocks and associated unsafe areas from which no executions could finish. In that approach, loops were not allowed. A later [2] extended the technique to investigating processes with loops via their loopless realizations (deloopings) and proved that deadlocks and unsafe points could be found using a finite number of deloopings even though the configuration space of a system with nontrivial loops was infinite. However, computations were not allowed to run indefinitely. Another contribution of the present paper is the proof that when the safe states are the ones from which there is a computation which runs indefinitely, these safe states can be identified by studying a finite set of deloopings of the system (cf. Thm. 5.5).

2 Concurrent systems and their executions

We are given a set $\mathcal{O}$ of *resources* that the processes may *lock* or *release*. In compliance with a longstanding tradition, locking a resource $A \in \mathcal{O}$ will be denoted by P_A and releasing a resource $A \in \mathcal{O}$ will be denoted by V_A. We assume that a process that has locked a resource cannot lock it again before releasing this resource; and that a process cannot release a resource without having it locked.

But there may be more than one process locking a given resource. Every resource $A \in \mathcal{O}$ has a certain *capacity* s_A with the intended meaning that it can be used simultaneously by not more than s_A different processes. The simplest resources protected by the classical critical regions have capacity 1.

2.1 Looping processes

When studying cooperation, it is customary to abstract from private actions by particular processes. By this abstraction, a process is a sequence of communications, i.e., in our setting, of actions P_A and V_A for various $A \in \mathcal{O}$. This is the sequence of communications that the process "wants" to perform; or "would" perform if no other process got in its way. After completing this sequence of actions, the process terminates.

Definition 2.1 Consider the set of strings of actions given by the following production:

$$t : T_0 ::= \mathbf{1} \mid t.P_A \mid t.V_A \mid t_1.(t_2)^* \tag{1}$$

The operation $\_.\_$ is the *concatenation*; we may extend it to arbitrary strings, with the second argument not necessarily single-action as in (1), by decreeing that it is associative and that $\mathbf{1}$ (empty string) is its right unit [3]. *Loop* $\_.(\_)^*$ is another formal operation on T_0; intuitively, $t_1.(t_2)^*$ describes the processes that perform t_1 and then run 0 or more times the sequence t_2. By another decree, $t.(\mathbf{1})^* = t$.

In accordance with the usual understanding of grammars, all elements of T_0 are finite strings. The way they define infinite executions, is discussed in Sec. 2.5 on page 9.

T_0 is too large for our set of processes. For instance, $P_A.P_A \in T_0$, while we do not want to allow any process to lock same resource twice without releasing it. We define its subset, $T \subset T_0$, which will be referred to as the set of looping processes.

We want every looping process $t \in T$ to satisfy the following (informal) constraints:

(i) between any two actions P_A in t, there is an intervening action V_A,

(ii) between any two actions V_A in t, there is an intervening action P_A,

(iii) for every contiguous subsequence $t_1.(t_2)^*$ of t, the numbers of P_A's and of V_A's in t_2 are equal,

(iv) before every action V_A in t, there must occur a corresponding action P_A.

These constraints take care of the assumptions in the beginning of Sec. 2.

We define *resource use characteristics* of a process as the number of locks acquired on a resource $A \in \mathcal{O}$ by the process t, for instance $r_A(P_A.V_A.P_B) = 0$. We only allow such t_2 in $t_1.(t_2)^*$ that $r_A\, t_2 = 0$.

[3] It follows easily that $\mathbf{1}$ is its left unit too. By (1), each element of T_0 must begin with $\mathbf{1}$, but we will often take the liberty of simplifying the initial $\mathbf{1}.t$ to t.

2.2 Looping vs. loopless processes

Definition 2.2 A *loopless process* is a looping process without the operation $\_.(\_)^*$. Again, we distinguish two sets:
$D_1 \stackrel{\text{def}}{=} \{t \in T_1 \mid \text{no occurrence of } \_.(\_)^* \text{ in } t\}$ and $D \stackrel{\text{def}}{=} D_1 \cap T$

A loopless process does not have to eventually release a resource. E.g., it may lock it for ever; or it may never release a resource before acquiring another. For instance, P_A is a valid loopless process.

Looping processes are a convenient way of describing infinite sets of related loopless processes. This is done by means of a relation between the one and the other, as described below:

Definition 2.3 Let $\triangleright \subset D_1 \times T_1$ be the least relation defined by the following inference system:

$$\frac{}{\mathbf{1} \triangleright \mathbf{1}} \qquad \frac{d \triangleright t}{d.P_A \triangleright t.P_A} \qquad \frac{d \triangleright t}{d.V_A \triangleright t.V_A} \qquad \frac{d_0 \triangleright t_0 \quad d_1 \triangleright t_1 \quad \ldots \quad d_k \triangleright t_1}{d_0.d_1.\ldots.d_k \triangleright t_0.(t_1)^*} \; (k \geq 0)$$

Whenever $d \triangleright t$ for a certain $d \in D$ and a certain $t \in T$, the process d is called a *delooping* of the process t.

Proposition 2.4 *If $d_1 \triangleright t_1$ and $d_2 \triangleright t_2$ then $d_1.d_2 \triangleright t_1.t_2$. If $d \triangleright t$ then $r_A\, d = r_A\, t$.* ∎

Two loopless processes d and d', which deloop the same looping process t, may be compared on the number of "turns" of the t's loops needed to generate them.

Definition 2.5 Define $\lessdot \subset D_1 \times T_1 \times D_1$ as the least (ternary!) relation such that:

(i) $\mathbf{1} \lessdot_{\mathbf{1}} \mathbf{1}$,

(ii) if $d \lessdot_t \bar{d}$ then $d.P_A \lessdot_{t.P_A} \bar{d}.P_A$ and $d.V_A \lessdot_{t.V_A} \bar{d}.V_A$ for every $A \in \mathcal{O}$,

(iii) if $t_1 \neq \mathbf{1}$ and

$$d_0 \lessdot_{t_0} \bar{d}_0 \qquad d_1 \lessdot_{t_1} \bar{d}_1 \qquad \ldots \qquad d_k \lessdot_{t_1} \bar{d}_k \tag{2}$$

then $d_0.d_1.\ldots.d_\ell \lessdot_{t_0.(t_1)^*} \bar{d}_0.\bar{d}_1.\ldots.\bar{d}_k$ for every $\ell \leq k$.

$d \lessdot_t \bar{d}$ reads: *$\bar{d}$ is a further delooping of t than d.* Informally, $d \lessdot_t \bar{d}$ means that a loop within t is run more times to produce $\bar{d}$ than to produce d.

Pt. (iii) in the definition above describes the only possibility of two loopless processes to be $\lessdot_t$-related and not equal: this happens whenever in some derivation of the delooping relations i.e., sequences of the basic inferences, of

$d_1 \triangleright t$ and $d_2 \triangleright t$ we have$\ell < k$, i.e., some of the deloopings are skipped at the left-hand side .Two such derivations giving $d_1 \lessdot_t d_2$ are said to realize $d_1 \lessdot_t d_2$.

Proposition 2.6 *If $d_1 \lessdot_t d_2$ then $d_1 \triangleright t$ and $d_2 \triangleright t$.*

Lemma 2.7 *If $d_1 \lessdot_t d_2$ then length $d_1 \leq$ length d_2.*

Proposition 2.8 *For every $t \in T_1$, the relation $\lessdot_t$ is a partial order on the set*

$\{d \in D_1 \mid d \triangleright t\}$.

Whenever $d = d_1.d_2 \in D_0$, the loopless process d_1 is called a *prefix* of d, denoted $d_1 \sqsubseteq d$. The prefix relation is a partial order in D_0. The set of prefixes of a loopless process d is denoted by $Pref_d$.

A bit artificially, the notion of prefix may be generalized to looping processes.

Definition 2.9 For any $t \in T_0$, the set $Pref_t \subset \bigcup_{i=1}^{\infty} T_0^i$ (the union of Cartesian powers of T_0) of *prefixes* of t is defined inductively as follows:

$$Pref_{\mathbf{1}} \stackrel{\text{def}}{=} \{\mathbf{1}\} \quad Pref_{t.P_A} \stackrel{\text{def}}{=} Pref_t \cup \{t.P_A\} \quad Pref_{t.V_A} \stackrel{\text{def}}{=} Pref_t \cup \{t.V_A\}$$
$$Pref_{t_1.(t_2)^*} \stackrel{\text{def}}{=} (Pref_{t_1} \cup \{\langle t_1, t\rangle \mid t \in Pref_{t_2}\})/t_1 = \langle t_1, \mathbf{1}\rangle = \langle t_1, t_2\rangle$$

Note that this boils down to the former prefix in the absence of loops in t. The prefix partial order $\sqsubseteq$ on D_0 induces a relation in $Pref_t$, but this relation is not a partial order any more; still, we are going to denote it by $\sqsubseteq$.

Proposition 2.10 *(i) For every derivation of the delooping relation $d \triangleright t$, induction over the basic inferences defines a mapping $\Phi_{d \triangleright t} : Pref_d \to Pref_t$ translating the partial order $\sqsubseteq$ to the induced relation in $Pref_t$. (This actually defines the relation in $Pref_t$.*

(ii) Given derivations of $d \triangleright t$ and $\bar{d} \triangleright t$ realizing $d \lessdot_t \bar{d}$, there exists a natural mapping $\Psi_{d \lessdot_t \bar{d}} : Pref_{\bar{d}} \to Pref_d$ "forgetting" the extra turns of the loops in $\bar{d}$.

(iii) For any realization of $d \lessdot_t \bar{d}$, $\Phi_{\bar{d} \triangleright t} \circ \Psi_{d \lessdot_t \bar{d}} = \Phi_{d \triangleright t}$

Do not confuse the different partial orders on D: $\sqsubseteq$ — the prefix order, and $\lessdot_t$ for a given t — number-of-turns order. Note also that for $t_1 \neq t_2$, the orders $\lessdot_{t_1}$ and $\lessdot_{t_2}$ are, in general, different.

2.3 Concurrent systems and their configurations

Definition 2.11 A *concurrent system* $\mathcal{C} = (\mathcal{O}, s, C)$ consists of

- a set $\mathcal{O}$ of *resources*,
- a *capacity* function $s : \mathcal{O} \to \mathbb{N}$ (natural numbers),
- a finite set C of (looping) processes in T defined over $\mathcal{O}$ — i.e., whenever a P_A or a V_A occurs in a $t \in C$, $A \in \mathcal{O}$.

Definition 2.12 A *configuration* of a concurrent system $\mathcal{C} = (\mathcal{O}, s, C)$ is a function κ assigning to every looping process $t \in C$ a prefix: $\kappa\, t \in \mathit{Pref}_t$. The set of configurations of a system $\mathcal{C}$ will be denoted by $\mathit{Conf}_{\mathcal{C}}$. The *initial configuration* is defined by $\bar{\mathbf{1}}\, t \stackrel{\text{def}}{=} \mathbf{1}$ for every $t \in C$.

Intuitively, every configuration κ is an account of how the particular processes in $\mathcal{C}$ procede. Whenever a process t performs an action

$$a \in \mathit{Act}_{\mathcal{O}} \stackrel{\text{def}}{=} \{P_A \mid A \in \mathcal{O}\} \cup \{V_A \mid A \in \mathcal{O}\} \cup \{\mathbf{1}\} \tag{3}$$

a corresponding configuration κ_1 moves to another configuration κ_2.

Definition 2.13 For every process $t \in C$, define a *computation step by* t as the following relation:

$$\begin{aligned} \kappa_1 \overset{t}{\mapsto} \kappa_2 \iff\ & \forall_{t' \neq t}\, \kappa_2\, t' = \kappa_1\, t'\ \& \\ & \exists_{a \in \mathit{Act}_{\mathcal{O}}}\, \kappa_2\, t = (\kappa_1\, t).a\ \& \\ & \kappa_2\, t \in \mathit{Pref}_t \end{aligned}$$

The union $\mapsto \stackrel{\text{def}}{=} \bigcup_t \overset{t}{\mapsto}$ is called a *computation step*. Whenever $\kappa_1 \mapsto \kappa_2$, the configuration κ_2 is called a *successor* of the configuration κ_1. The transitive closure of the successor relation $\mapsto$ is denoted by $\sqsubseteq$. In a loopless concurrent system, the relation $\sqsubseteq$ is a partial order.

Proposition 2.14 *For all configurations κ_1 and κ_2,*

$$\kappa_1 \sqsubseteq \kappa_2 \iff \forall_{t \in C}\ \kappa_1\, t \sqsubseteq \kappa_2\, t$$

(the symbol $\sqsubseteq$ at the right hand side denotes the prefix relation).

If $\mathcal{C}$ is a system of only one process, t, then $\mathit{Conf}_{\mathcal{C}} = \mathit{Pref}_t$ with the relation $\sqsubseteq$ The functions r_A describing the number of locks on a resource A acquired by a given process, are extended to configurations of a concurrent system:

Definition 2.15 *Resource use characteristics of a configuration* is defined by

$$r_A\, \kappa \stackrel{\text{def}}{=} \sum_{t \in C} r_A(\kappa\, t)$$

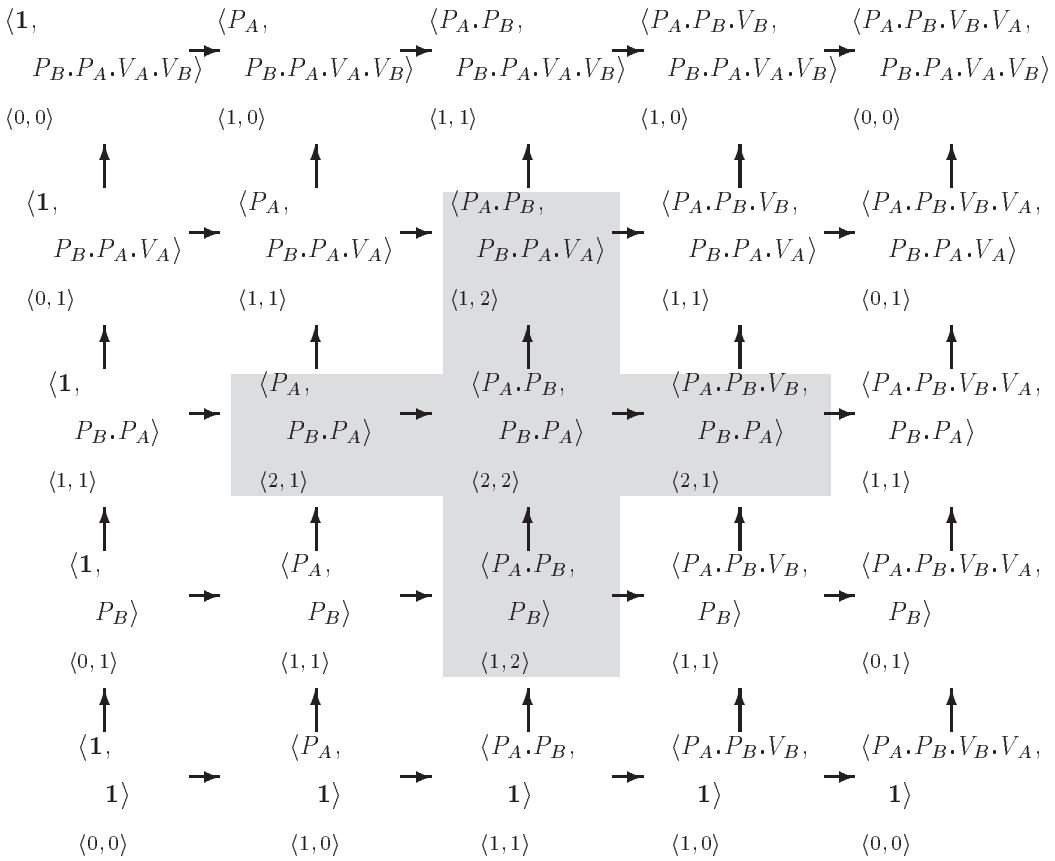


Fig. 1. System $\mathcal{C}_1$ from Ex. 2.16.

A configuration κ is *forbidden* if it is using resources beyond their capacities; i.e., if $r_A\,\kappa > s_A$ for some resource $A \in \mathcal{O}$. An *allowed* configuration is one which is not forbidden. The set of all allowed configurations of a system $\mathcal{C}$ is denoted by $\mathcal{A}_\mathcal{C}$:

$$\mathcal{A}_\mathcal{C} \stackrel{\text{def}}{=} \{\kappa \in \mathit{Conf}_\mathcal{C} \mid \forall_{A\in\mathcal{O}}\, r_A\kappa \leq s_A\}$$

Example 2.16 Consider a more complex system $\mathcal{C}_1 \stackrel{\text{def}}{=} (\mathcal{O}, s, C)$ with two resources $\mathcal{O} \stackrel{\text{def}}{=} \{A, B\}$ whose capacities are $s_A \stackrel{\text{def}}{=} 1$ and $s_B \stackrel{\text{def}}{=} 1$, and two processes

$$C \stackrel{\text{def}}{=} \{P_A.P_B.V_B.V_A\,,\; P_B.P_A.V_A.V_B\}$$

(see Fig. 1). The pairs of numbers under the configurations are the values of r_A and of r_B. The five forbidden configurations are shaded.

This system and other similar are often referred to as the *Swiss flag*.

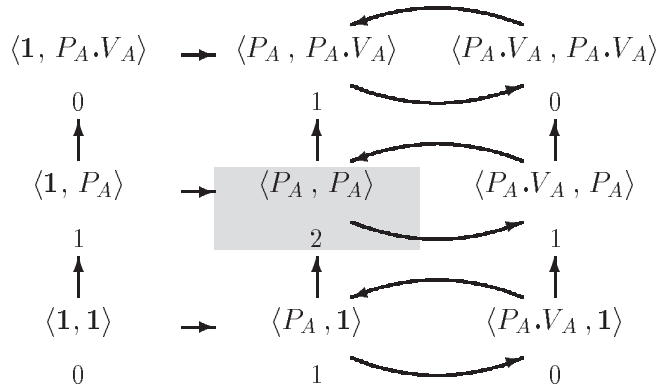


Fig. 2. System $\mathcal{C}_2$ from Ex. 2.17.

Example 2.17 Take the case, where one of the processes contains a loop: $\mathcal{C}_2 \stackrel{\text{def}}{=} (\mathcal{O}, s, C)$ with a single resource $\mathcal{O} \stackrel{\text{def}}{=} \{A\}$ whose capacity is $s_A \stackrel{\text{def}}{=} 1$, and with two processes:

$$C \stackrel{\text{def}}{=} \{P_A.(V_A.P_A)^* \, , \, P_A.V_A\}$$

The set of configurations of $\mathcal{C}_2$ is depicted in Fig. 2.

Forbidden configurations are the ones that cannot be entered in a normal "life" of a concurrent system. On the other hand, such "life" may only procede increasingly with respect to the partial order $\sqsubseteq$ on configurations. There may, therefore, exist configurations from which there is no way out, corresponding to deadlocks. For instance, in Example 2.16, $\langle P_A, P_B \rangle$ is a deadlock — one process has claimed the resource A and waits for B; the other has locked B and waits for A. Dually, there may exist allowed configurations with no way in: cf. $\langle P_A.P_B.V_B \, , \, P_B.P_A.V_A \rangle$ in Ex. 2.16.

Deadlocks and other related notions will be discussed in Sec. 2.5.

The notion of delooping $\triangleright$ from Def. 2.3 on p. 4 is extended to concurrent systems as follows:

Definition 2.18 Assume $\mathcal{C} = (\mathcal{O}, s, C)$ and $\mathcal{C}' = (\mathcal{O}, s, C')$ are concurrent systems sharing the set of resources, and $\mathcal{C}$ is loopless, i.e., all its processes are loopless. Let $f : C \to C'$ be a bijection such that $d \triangleright f\, d$ for every $d \in C$. Then the system $\mathcal{C}$ is called an *f-delooping* of the system $\mathcal{C}'$, denoted $\mathcal{C} \triangleright_f \mathcal{C}'$.

The partial orders $\preccurlyeq_t$ from from Def. 2.5 on p. 4 are extended to concurrent systems as follows:

Definition 2.19 Assume $\mathcal{C}_1 = (\mathcal{O}, s, C_1)$ and $\mathcal{C}_2 = (\mathcal{O}, s, C_2)$ are loopless and $\mathcal{C} = (\mathcal{O}, s, C)$ is a looping system and all three systems share the set of resources. Let $\mathcal{C}_1 \triangleright_{f_1} \mathcal{C}$ and $\mathcal{C}_2 \triangleright_{f_2} \mathcal{C}$ for some bijections f_1 and f_2. The pair

$(\mathcal{C}_2, f_2)$ is said to be a *further delooping of* $\mathcal{C}$ than the pair $(\mathcal{C}_1, f_1)$, denoted $(\mathcal{C}_1, f_1) \lessdot_{\mathcal{C}} (\mathcal{C}_2, f_2)$, if $d_1 \lessdot_{f_1 d_1} f_2^{-1}(f_1\, d_1)$ for all $d_1 \in C_1$.

2.4 Morphisms of concurrent systems

Definition 2.20 A *morphism* between concurrent systems $\mathcal{C} = (\mathcal{O}, s, C)$ and $\mathcal{C}' = (\mathcal{O}', s', C')$ is a triple (f, g, φ) consisting of:

- $f : C \to C'$ — assignment of processes in $\mathcal{C}'$ to processes in $\mathcal{C}$,
- $g : \mathcal{O} \to \mathcal{O}'$ — assignment of resources in $\mathcal{C}'$ to resources in $\mathcal{C}$,
- $\varphi : \mathit{Conf}_{\mathcal{C}} \to \mathit{Conf}_{\mathcal{C}'}$ — mapping of configurations

such that

(i) $\varphi\, \bar{\mathbf{1}} = \bar{\mathbf{1}}'$,

(ii) if $\kappa_1 \sqsubseteq \kappa_2$ then $\varphi\, \kappa_1 \sqsubseteq' \varphi\, \kappa_2$ for $\kappa_1, \kappa_2 \in \mathit{Conf}_{\mathcal{C}}$,

(iii) $\sum\{s_A \mid g\, A = B\} \leq s'_B$ for $B \in \mathcal{O}'$,

(iv) $r_B(\varphi\, \kappa\, t') \leq \sum\{r_A(\kappa\, t) \mid f\, t = t' \;\&\; g\, A = B\}$ for $B \in \mathcal{O}'$, $\kappa \in \mathit{Conf}_{\mathcal{C}}$ and $t' \in C'$.

It is clear that concurrent systems with their morphisms form a category.

Proposition 2.21 *A morphism takes allowed configurations to allowed configurations,*
i.e., if $(f, g, \varphi) : \mathcal{C} \to \mathcal{C}'$ *then* $\varphi(\mathcal{A}_{\mathcal{C}}) \subset \mathcal{A}_{\mathcal{C}'}$.

Proposition 2.22 *Assume* $\mathcal{C} = (\mathcal{O}, s, C)$ *is an* f*-delooping of a looping system* $\mathcal{C}' = (\mathcal{O}, s, C')$ *with the same set of resources, for a certain bijection* $f : C \to C'$*, i.e.,* $\mathcal{C} \triangleright_f \mathcal{C}'$*. Then for every derivation of the delooping relation, there exists a natural morphism of concurrent systems* $(f, Id_{\mathcal{O}}, \varphi)$*.*

Example 2.23 Let $\mathcal{C} = (\mathcal{O}, s, C))$ be a concurrent system of which t is one of the processes. There is an inclusion morphism $i_t : (\mathcal{O}, s, \{t\}) \to \mathcal{C}$: f is the inclusion $t \to C$, φ is defined by $\varphi(\mu)\, t = \mu$ and $\varphi(\mu)\, t' = \mathbf{1}$ for $t' \neq t$. g is the identity.

Similarly there is a projection $\pi_t : \mathcal{C} \to (\mathcal{O}, s, \{t\})$: $f(t') = t$ for all $t' \in \mathcal{C}$, $\varphi(\kappa) = \kappa\, t$ and g is the identity.

When $F = (f, g, \varphi) : \mathcal{C} \to \mathcal{C}'$ is a morphism of two concurrent systems and t is one of the processes in $\mathcal{C}$, we define the restriction $F_{|t} = \pi_{f(t)} \circ F \circ i_t$: $(\mathcal{O}, s, \{t\}) \to (\mathcal{O}', s', \{f(t)\})$

2.5 Execution trajectories

Definition 2.24 Given a concurrent system $\mathcal{C} = (\mathcal{O}, s, C)$, a *trajectory from a configuration* κ_0 is any sequence $\kappa_0 \kappa_1 \kappa_2 \ldots$ of allowed configurations, such

that $\kappa_{i-1} \mapsto \kappa_i$ for $i = 1, 2, \ldots$. The *length* of a trajectory $\kappa_0\kappa_1\kappa_2\ldots$ is the cardinality of the set $\{\kappa_0, \kappa_1, \kappa_2, \ldots\}$ minus 1. A trajectory is *finite* if its length is a natural number and it is *infinite* if it is $+\infty$.

Each trajectory may be viewed as a possible history of the "life" of a given system. Every concurrent system, in which a process contains a true loop, i.e., such $t_1.(t_2)^*$ that $t_2 \neq \mathbf{1}$, has a potential for infinite trajectories; but this potential may not be used if there are too many forbidden configurations. We could have excluded trajectories with repeated configurations. But this would restrict the framework to infinite trajectories only, depriving us of the capability of discussing some unwelcome phenomena, such as deadlocks.

Definition 2.25 For a finite trajectory $\kappa_0\kappa_1\kappa_2\ldots\kappa\kappa\kappa\ldots$, where κ is the last (infinitely repeated) configuration, κ_0 is called its *beginning*, κ its *end*, and the trajectory is said to *go from* κ_0 *to* κ. An existence of a trajectory from κ_1 to κ_2 is denoted by $\kappa_1 \prec \kappa_2$. An existence of an infinite trajectory from κ is denoted by $\kappa \prec \infty$.

Because of the requirement that all the intervening configurations be allowed, $\kappa_1 \sqsubseteq \kappa_2$ does not necessarily imply $\kappa_1 \prec \kappa_2$.

Definition 2.26 For an arbitrary allowed configuration $\kappa \in \mathcal{A}_\mathcal{C}$, we define

- the *future*: $\uparrow\kappa \stackrel{\text{def}}{=} \{\kappa' \in \mathcal{A}_\mathcal{C} \cup \{\infty\} \mid \kappa \prec \kappa'\}$, and
- the *past*: $\downarrow\kappa \stackrel{\text{def}}{=} \{\kappa' \in \mathcal{A}_\mathcal{C} \mid \kappa' \prec \kappa\}$.

Definition 2.27 Assume a certain set $\mathcal{F} \subset \mathcal{A}_\mathcal{C} \cup \{\infty\}$ of allowed configurations, called *final* configurations, is given. A configuration $\kappa \in \mathcal{A}_\mathcal{C}$ is a *deadlock with respect to* $\mathcal{F}$ if $\uparrow\kappa = \{\kappa\}$ and $\kappa \notin \mathcal{F}$. A configuration κ is *safe with respect to* $\mathcal{F}$ if $\uparrow\kappa \cap \mathcal{F} \neq \emptyset$. It is *unsafe* if it is not safe.

Informally, a configuration is a deadlock if it is allowed, not final and there is no outgoing trajectory. The life of a concurrent system, that has reached a deadlock configuration, ends there. A configuration is unsafe if there is no way of reaching a final state from it and no way to continue indefinitely, if the set $\mathcal{F}$ allows for this. Every deadlock is, of course, unsafe.

Example 2.28 Consider the system $\mathcal{C}_1$ in Example 2.16 on page 7. With $\mathcal{F} = \emptyset$, there are two deadlock configurations,

$$\langle P_A, P_B \rangle$$

$$\langle P_A.P_B.V_B.V_A \ , \ P_B.P_A.V_A.V_B \rangle$$

and every configuration is unsafe. With $\mathcal{F} = \{\langle P_A.P_B.V_B.V_A \ , \ P_B.P_A.V_A.V_B \rangle\}$, the only unsafe configuration, which is also a deadlock, is $\langle P_A, P_B \rangle$. And if

$\mathcal{F} = \{\langle P_A, P_B\rangle\}$, the only deadlock is $\langle P_A.P_B.V_B.V_A\ ,\ P_B.P_A.V_A.V_B\rangle$ while all configurations with the exception of $\langle \mathbf{1}, \mathbf{1}\rangle$, $\langle P_A, \mathbf{1}\rangle$, $\langle \mathbf{1}, P_B\rangle$ and $\langle P_A, P_B\rangle$ are unsafe.

Definition 2.29 Call a finite trajectory from κ_1 to κ_2 *left-maximal* [resp., *right-maximal*] if it cannot be extended to the left [resp., to the right], i.e., $\downarrow\kappa_1 = \{\kappa_1\}$ [resp., $\uparrow\kappa_2 = \{\kappa_2\}$].

Proposition 2.30 *Let $\mathcal{C} = (\mathcal{O}, s, C)$ be a loopless system, i.e., $C \subset D$. A configuration $\kappa \in \mathcal{A}_\mathcal{C}$ is unsafe if and only if every right-maximal trajectory beginning in κ ends in a deadlock.*

Example 2.31 Consider the system $\mathcal{C}_2$ in Example 2.17 on page 8. Whatever the set $\mathcal{F}$, there are no deadlocks. If $\infty \in \mathcal{F}$ then all configurations are safe. If $\mathcal{F} = \emptyset$ then every configuration is unsafe. This shows that Prop. 2.30 is not true for looping systems.

Proposition 2.32 *Let $\mathcal{C} = (\mathcal{O}, s, C)$ be a concurrent system (either loopless or looping). A configuration $\kappa \in \mathcal{A}_\mathcal{C}$ is unsafe if and only if every right-maximal trajectory beginning in κ either ends in a deadlock, or is infinite and $\infty \notin \mathcal{F}$.*

3 Geometry of concurrency

In this section, we are studying the geometric and topological notions which will later be used for giving the topological semantics of the processes from Section 2.

3.1 Ditopology

Definition 3.1 A partial order $\leq$ on a topological space X is called *closed* if $\leq$ is a closed subset of $X \times X$ in the product topology. In that case, $(X, \leq)$ is called a *po-space*.

Definition 3.2 Let X be a topological space. A collection $\mathcal{U}(X)$ of pairs $(U, \leq_U)$ with partially ordered open subsets U covering X is a *local partial order* on X if for every $x \in X$ there is a nonempty open neighbourhood $W(x) \subset X$ with a partial order $\leq_{W(x)}$ such that the restrictions of $\leq_U$ and $\leq_{W(x)}$ to $U \cap W(x)$ coincide for all $U \in \mathcal{U}(X)$ with $x \in U$, i.e.,

$$y \leq_U z \iff y \leq_{W(x)} z \qquad \begin{array}{l}\text{for all } U \in \mathcal{U}(X) \text{ such that } x \in U \\ \text{and for all } y, z \in W(x) \cap U\end{array} \tag{4}$$

A neighbourhood $W(x)$ with the partial order as in Def. 3.2 is called a *po-neighbourhood* of x.

Example 3.3 The circle $\mathbb{S}^1 \stackrel{\text{def}}{=} \{e^{i\theta} \in \mathbb{C} \mid 0 \leq \theta \leq 2\pi\}$ has a local partial order: the open subsets $U_1 = \{e^{i\theta} \mid 0 < \theta < 2\pi\}$ and $U_2 = \{e^{i\theta} \mid -\pi < \theta < \pi\}$ are (partially) ordered by the order on the θ's [4]. Notice that the transitive closure of the union of these local partial orders is $x \leq y$ for any pair x, y. Hence we do not take transitive closure of the local orders!

Definition 3.4 Two local partial orders $\mathcal{U}(X)$ and $\mathcal{V}(X)$ on X are *equivalent* if their union $\mathcal{U}(X) \cup \mathcal{V}(X)$ is a local partial order; in other words, $\mathcal{U}(X)$ and $\mathcal{V}(X)$ are equivalent if and only if for every $x \in X$ there is a nonempty open neighbourhood $W(x) \subset X$ such that the restrictions of $\leq_U$ and $\leq_V$ to $W(x)$ coincide for all $U \in \mathcal{U}(X)$ and $V \in \mathcal{V}(X)$ with $x \in U$ and $x \in V$. A topological space X together with an equivalence class of local partial orders is called a *locally partially ordered space*. If, moreover, there is a covering $\mathcal{U}$ in the equivalence class such that all $(U, \leq_U) \in \mathcal{U}$ are po-spaces, then X is a *local po-space*.

We let $(X,\mathcal{U})$ denote the locally partially ordered space which has $\mathcal{U}$ as a representative of the equivalence class of local partial orders. When $(X,\mathcal{U})$ is a local po-space, we always assume that the representative $\mathcal{U}$ is in fact a covering by po-spaces. As will be seen (Sec. 4.1), po-spaces correspond to loopless processes while local po-spaces correspond to looping processes.

A local po-space is Hausdorff by the usual argument for a po-space.

Some standard operations on topological spaces and on partial orders carry over to local po-spaces.

Definition 3.5 If $(X,\mathcal{U})$ is a local po-space and $A \subset X$, then define the *restriction* $(A, \{U \cap A \mid U \in \mathcal{U}\}$ of $(X,\mathcal{U})$ with partial order on $U \cap A$ inherited from U. If $(X,\mathcal{U})$ and $(Y,\mathcal{V})$ are local po-spaces, then define their *Cartesian product* $(X \times Y, \{U \times V \mid U \in \mathcal{U}, V \in \mathcal{V}\}$ with partial order on $U \times V$ given by $\langle u_1, v_1\rangle \leq \langle u_2, v_2\rangle \stackrel{\text{def}}{\iff} u_1 \leq_U u_2 \;\&\; v_1 \leq_V v_2$ If $(X_0,\mathcal{U}_0)$ and $(X_1,\mathcal{U}_1)$ are local po-spaces, then define their *disjoint union* $X_0 \sqcup X_1 \stackrel{\text{def}}{=} (X_0 \times \{0\} \cup X_1 \times \{1\}, \{U_0 \times \{0\} \mid U_0 \in \mathcal{U}_0\} \cup \{U_1 \times \{1\} \mid U_1 \in \mathcal{U}_1\}$ with partial order on $U_i \times \{i\}$ given by $\langle u_1, i\rangle \leq_{U_i} \langle u_2, i\rangle \stackrel{\text{def}}{\iff} u_1 \leq u_2$ for $i = 0, 1$.

Proposition 3.6 *The constructions of restriction, Cartesian product and disjoint union of local po-spaces in Def. 3.5 do not depend on the selection of a representative covering from the equivalence classes. Moreover, when applied*

[4] The condition (4) differs slightly from the one in [3]. For instance, the cover $\{U_1, U_2\}$ in Example 3.3 does not satisfy the old definition.

to local po-spaces, these constructions result in local po-spaces; and when applied to po-spaces, these constructions result in po-spaces.

Another construction that we need is identification of a pair of points. In general, a quotient of a topological space does not inherit the topological properties of the original space. Even a quotient of a metric space may fail to be T0. But, as shown below, the identification of two points in a local po-space results in a local po-space.

Definition 3.7 Assume $(X,\mathcal{U})$ is a local po-space and $x_1, x_2 \in X$. Let $X/x_1 \sim x_2$ have the quotient topology. Define a local po-structure $\widetilde{\mathcal{U}}$ on $X/x_1 \sim x_2$ as follows. If $x_1 = x_2$, let $\widetilde{\mathcal{U}} = \mathcal{U}$. If $x_1 \neq x_2$, let $W(x_1)$ and $W(x_2)$ be disjoint po-neighbourhoods of x_1 and x_2 and let $\widetilde{U} = (W(x_1) \cup W(x_2))/x_1 \sim x_2$ with partial order given by the transitive hull of the relations in $\leq_{W(x_1)}$ and $\leq_{W(x_2)}$. Then define $\widetilde{\mathcal{U}} \stackrel{\text{def}}{=} \{\widetilde{U}\} \cup \{U \setminus \{x_1, x_2\} \mid U \in \mathcal{U}\}$ where the partial order on $U \setminus \{x_1, x_2\}$ is the restriction of the partial order on U.

Notice that the only relations which are in the quotient and not in the original space are induced by these: let $y \in W(x_1)$ and $z \in W(x_2)$; then

- if $y \leq_{W(x_1)} x_1$ and $x_2 \leq_{W(x_2)} z$ then $y \leq_{\widetilde{U}} z$,
- if $y \geq_{W(x_1)} x_1$ and $x_2 \geq_{W(x_2)} z$ then $y \geq_{\widetilde{U}} z$.

Proposition 3.8 *Let $(X,\mathcal{U})$ be a local po-space and let $x_1, x_2 \in X$. Then:*

(i) $X/x_1 \sim x_2$ with cover as above is a local po-space.

(ii) If $\mathcal{U}$ and $\mathcal{V}$ are equivalent local po-structures on X then the local po-structures $\widetilde{\mathcal{U}}$ and $\widetilde{\mathcal{V}}$ are equivalent local po-structures on $X/x_1 \sim x_2$.

(iii) The restriction of $\mathcal{U}$ to $X \setminus \{x_1, x_2\}$ is equivalent to the restriction of $\widetilde{\mathcal{U}}$ to

$$(X/x_1 \sim x_2) \setminus \{[x_1]\}$$

3.9 Remark If X is a po-space, then an identification of two points will usually result in a local po-space which is not a po-space. For instance, the circle with the local po-structure in Ex. 3.3 could be thought of as coming from identifying the endpoints on an interval. But the disjoint union with amalgamation $X_1 \sqcup X_2/x_1 \sim x_2$ (where $x_1 \in X_1$ and $x_2 \in X_2$) of two po-spaces is a po-space.

3.2 Morphisms of local po-spaces

Definition 3.10 Let $(X,\mathcal{U})$ and $(Y,\mathcal{V})$ be locally partially ordered spaces. A continuous map $f : X \to Y$ is called a *dimap* (directed map) if for any $x \in X$

there are po-neighbourhoods $W(x)$ and $W(f(x))$ such that $x_1 \leq_{W(x)} x_2 \Rightarrow f(x_1) \leq_{W(f(x))} f(x_2)$ whenever $x_1, x_2 \in f^{-1}(W(f(x))) \cap W(x)$.

It is not hard to see, that dimaps are well defined, i.e., that the definition does not depend on the choice of representative $\mathcal{U}$ of the equivalence class of local po-structures. In the case of po-spaces, dimaps are the same as monotone continuous maps. It is also straightforward to see that local po-spaces and dimaps form a category.

A dimap with an inverse which is also a dimap is called a *dihomeomorphism*.

3.3 Dipaths

The topological counterpart of execution trajectories in a local po-space X are dimaps from the half-straight line $\mathbb{R}^+$ (with the usual topology and order) to X:

Definition 3.11 Let X be a local po-space. Every dimap $\varphi : \mathbb{R}^+ \to X$ is called a *dipath* in X. The point $\varphi\, 0 \in X$ is called the *beginning* of the dipath. If there exists an $x \in X$ such that the counterimage $\varphi^{-1}(x)$ contains a half-straight line, the dipath is referred to as *finite* and that point is called its *end*. The existence of a finite dipath with beginning x and end y is denoted $x \prec y$.

We are also interested in infinite dipaths, but only the ones that do not shrink big subsets of $\mathbb{R}^+$ to small subsets of X. This corresponds to the execution trajectories (see Sec. 2.5 on p. 9) proceeding with a constant "speed", i.e., not ending with an infinite sequence of repetitions. Actually, we do not care for that speed to be constant, we only want to make sure that it never goes close to zero. Formally, this is expressed as follows:

Definition 3.12 A dipath is called *proper* if it does not converge to a point, i.e., if every $y \in Y$ has a po-neighbourhood $W(y)$ such that the counterimage $\varphi^{-1}(W(y))$ does not contain a half-straight line. The existence of a proper dipath with beginning x is denoted $x \prec \infty$.

Definition 3.13 Assume X is a local po-space and $x \in X$. Then we define

- the *future*: $\uparrow x \stackrel{\text{def}}{=} \{y \in X \cup \{\infty\} \mid x \prec y\}$, and
- the *past*: $\downarrow x \stackrel{\text{def}}{=} \{y \in X \mid y \prec x\}$.

4 Topological semantics of concurrent systems

We have separately discussed the discrete concurrent systems (Sec. 2) and the local po-spaces (Sec. 3) which are supposed to model them in a continuous way. Here, we are giving formal details pertaining to that modeling.

4.1 Geometric realization of concurrent processes

Definition 4.1 To every looping process $t \in T_1$, as defined in Sec. 2.1, assign:

- a local po-space G_t called the *geometric realization* of t,
- points $b_t, e_t \in G_t$ (for *begin* and *end*),
- *resource-use characteristics* $r_{t,A} : G_t \to \mathbb{Z}$ yielding the number of locks process t holds on resource A at a given point — this number may a *priori* be less than 0 or greater than 1, and it requires a proof that this is not the case for $t \in T$.

This is done by structural induction on T_0 (cf. the defining production (1) on page 3) in the following way [5] :

$$G_{\mathbf{1}} \stackrel{\text{def}}{=} \{\star\} \quad \text{(singleton)}, \quad b_{\mathbf{1}} \stackrel{\text{def}}{=} \star \qquad e_{\mathbf{1}} \stackrel{\text{def}}{=} \star, \quad r_{\mathbf{1},A}\, x \stackrel{\text{def}}{=} 0.$$

$$\begin{cases} G_{P_A} \stackrel{\text{def}}{=} [0,1] \\ b_{P_A} \stackrel{\text{def}}{=} 0 \qquad e_{P_A} \stackrel{\text{def}}{=} 1 \\ r_{P_A,B}\, x \\ \quad \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } B \neq A \\ 0 & \text{if } B = A \text{ and } x = 0 \\ 1 & \text{if } B = A \text{ and } x > 0 \end{cases} \end{cases} \qquad \begin{cases} G_{V_A} \stackrel{\text{def}}{=} [0,1] \\ b_{V_A} \stackrel{\text{def}}{=} 0 \qquad e_{V_A} \stackrel{\text{def}}{=} 1 \\ r_{V_A,B}\, x \\ \quad \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } B \neq A \\ 0 & \text{if } B = A \text{ and } x < 1 \\ -1 & \text{if } B = A \text{ and } x = 1 \end{cases} \end{cases}$$

$$\begin{cases} G_{t_1 \cdot t_2} \stackrel{\text{def}}{=} G_{t_1} \sqcup G_{t_2} / b_{t_2} \sim e_{t_1} \\ b_{t_1 \cdot t_2} \stackrel{\text{def}}{=} b_{t_1} \qquad e_{t_1 \cdot t_2} \stackrel{\text{def}}{=} e_{t_2} \\ r_{t_1 \cdot t_2, A}\, x \\ \quad \stackrel{\text{def}}{=} \begin{cases} r_{t_1,A}\, x & \text{if } x \in G_{t_1} \\ r_{t_1,A}\, e_{t_1} + r_{t_2,A}\, x & \text{if } x \notin G_{t_1} \end{cases} \end{cases} \qquad \begin{cases} G_{t_1 \cdot (t_2)^*} \stackrel{\text{def}}{=} G_{t_1} \sqcup G_{t_2} / e_{t_2} \sim b_{t_2} \sim e_{t_1} \\ b_{t_1 \cdot (t_2)^*} \stackrel{\text{def}}{=} b_{t_1} \qquad e_{t_1 \cdot (t_2)^*} \stackrel{\text{def}}{=} e_{t_2} \\ r_{t_1 \cdot (t_2)^*, A}\, x \\ \quad \stackrel{\text{def}}{=} \begin{cases} r_{t_1,A}\, x & \text{if } x \in G_{t_1} \\ r_{t_1,A}\, e_{t_1} + r_{t_2,A}\, x & \text{if } x \notin G_{t_1} \end{cases} \end{cases}$$

According to the above definition, a command P_A acquires the resource A at the beginning of its execution; similarly, a command V_A releases the resource A at the end of its execution.

Proposition 4.2 *If $t \in T$ then $\forall_{x \in G_t} \forall_{A \in \mathcal{O}}\ 0 \leq r_{t,A}\, x \leq 1$.*
If $t \in T$ then $\forall_{A \in \mathcal{O}}\ r_{t,A}\, e_t = r_A\, t$ (as defined in Sec. 2.1, page 3).

[5] $I = [0,1]$ is the unit interval with the obvious po-structure. For local po-structures on the disjoint union and quotient, see Sec. 3.1.

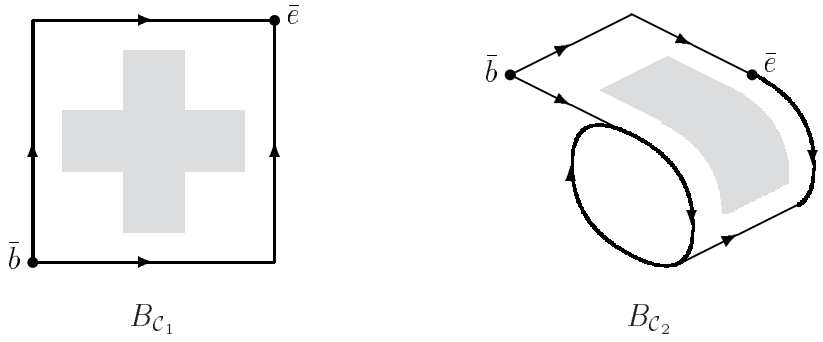


Fig. 3. Geometric realizations of concurrent systems, see Ex. 4.4.

We get a geometric representation of all configurations by taking $G_{\mathcal{C}} \stackrel{\text{def}}{=} \prod_{t\in C} G_t$ but we are only interested in the allowed states and hence the following definition.

Definition 4.3 *The geometric realization* $B_{\mathcal{C}}$ of a concurrent system $\mathcal{C} = (\mathcal{O}, s, C)$ (cf. Sec. 2.3) is the local po-space

$$B_{\mathcal{C}} \stackrel{\text{def}}{=} \{\bar{x} \in \prod_{t\in C} G_t \mid \forall_{A\in\mathcal{O}}\ 0 \le \sum_{t\in C} r_{t,A}\, x_t \le s_A\}$$

($\prod$ denotes the Cartesian product; x_t is the t-th component of $\bar{x}$).

Example 4.4 Fig. 3 presents the geometric realizations of the concurrent system with two resources from Ex. 2.16 (Swiss flag), and of the concurrent looping system with one resource from Ex. 2.17. The shaded areas correspond to forbidden configurations (the removed complement $\prod_{t\in C} G_t \setminus B_{\mathcal{C}}$).

Definition 4.5 Let μ be a configuration (a prefix) for a process t. The point in G_t corresponding to μ is an endpoint found inductively by $\mathbf{1} \to e(G_{\mathbf{1}})$, $\{t.P_A\} \to e(G_{t.P_A})$, $\{t.V_A\} \to e(G_{t.V_A})$ and for loops: $t_1.(t_2)^*$, the prefix $\langle t_1, t\rangle \to e(t)$ (which is included in $G_{t_1.t_2^*}$ in the obvious way.).For more than one process, $G_{\mathcal{C}}$ is a cartesian product and configurations are tuples of endpoints.

Geometric realization is a functor:

Definition 4.6 Let $F : (\mathcal{O}, s, \{t\}) \to (\mathcal{O}', s', \{t'\})$ be a morphism between two concurrent systems, each consisting of only one process. We define $B_F : G_t \to G_{t'}$ iteratively using the map of configurations.

Let $F : \mathcal{C} \to \mathcal{C}'$ be a morphism of concurrent systems. Then we define a morphism of the geometric representation of all configurations, $B_F : G_{\mathcal{C}} \to G_{\mathcal{C}'} = \prod_{t\in C} B_{F|t}$.

Since the morphisms of concurrent systems map allowed states to allowed states, $B_F : B_{\mathcal{C}} \to B_{\mathcal{C}'}$

To see that this definition makes sense, for the systems of one proces, notice that the configurations in that case are the prefixes. Hence we have defined a map from all endpoints of the intervals constituting G_t. This gives a continuous map from G_t to $G_{t'}$, since the glueings in the construction of the geometric realizations correspond to the identifications made in the prefixes in case of a loop and to the successor relations, which are preserved by morphisms of concurrent systems. [6]

It is not hard to see that the finite trajectories go to finite dipaths and infinite trajectories map to proper dipaths.

4.2 General properties of the (local) po-spaces arising from concurrent processes

Proposition 4.8 *For an arbitrary concurrent system $\mathcal{C}$, the space $B_{\mathcal{C}}$ is compact.*

Proposition 4.9 *If $\mathcal{C} = (\mathcal{O}, s, C)$ and $C \subset D$ (i.e., there are only loopless processes in system $\mathcal{C}$) then $B_{\mathcal{C}}$ is a po-space, with the points $\bar{b}, \bar{e} \in B_{\mathcal{C}}$, defined*

[6] We spell out the geometric realization of the morphism from a delooping to the looped system:

Definition 4.7 Let $\mathcal{C}_1 \triangleright_f \mathcal{C}_2$ be an f-delooping. Then for any sequence s of the inferences from Def. 2.3 on p. 4 which derives the relation $\mathcal{C}_1 \triangleright_f \mathcal{C}_2$, we define a map $B_s : B_{\mathcal{C}_1} \to B_{\mathcal{C}_2}$ of the geometric realizations by induction on the geometric realization of the basic inferences (we omit the obvious generalization from 1-dimensional to higher-dimensional systems).

- The geometric realization of $B_{\mathbf{1}\triangleright\mathbf{1}} : B_{\mathbf{1}} \to B_{\mathbf{1}}$ is the identity.
- Given $B_{d\triangleright t}$ we define $B_{d.P_A \triangleright t.P_A}$. This is
  (i) A map from $G_{d.P_A} = G_d \sqcup G_{P_A}$ to $G_{t.P_A} = G_t \sqcup G_{P_A}$, which we set to $B_{d\triangleright t}$ on the first component and the identity on G_{P_A},
  (ii) A map of beginnings and ends: $B_{d.P_A}(b_{d.P_A}) = b_{t.P_A}$, $B_{d.P_A}(e_{d.P_A}) = e_{t.P_A}$.
- Given $B_{d\triangleright t}$ we define $B_{d.V_A \triangleright t.V_A}$ in the same way.
- Given $B_{d_0\triangleright t_0}$, $B_{d_1\triangleright t_1}$, $B_{d_2\triangleright t_1}$, $\ldots$, $B_{d_k\triangleright t_1}$, we have to define $B_{d_0.d_1\ldots d_k \triangleright t_0.(t_1)^*}$. Again this is defined componentwise on $G_{d_0.d_1\ldots d_k} = G_{d_0} \sqcup \ldots \sqcup G_{d_k}$ mapping G_{d_0} to G_{t_0} and the last k components (if any) to G_{t_1}. The beginning, $b_{d_0.d_1\ldots d_k}$ maps to $b_{t_0.(t_1)^*}$ and the end $b_{d_0.d_1\ldots d_k}$ maps to $e_{t_0.(t_1)^*}$.

This gives a map from $G_{\mathcal{C}_1}$ to $G_{\mathcal{C}_2}$, and we have to see that it restricts to a map from $B_{\mathcal{C}_1}$ to $B_{\mathcal{C}_2}$. For this, it suffices to show that the resource use characteristic commutes with B_s and that is not hard to see. Remember that the effect on the resource use characteristic from traversing a loop is trivial.

One has to check that these maps are well defined, i.e., that identifications of beginnings and ends made in the iterative construction of $G_{\mathcal{C}_1}$ are preserved upon mapping to $G_{\mathcal{C}_2}$, but this is easy to see.

by $\bar{b}\,t \stackrel{\text{def}}{=} b_t$ *and* $\bar{e}\,t \stackrel{\text{def}}{=} e_t$ *for* $t \in C$*, being, respectively, its least and greatest elements. Moreover,* $B_{\mathcal{C}}$ *is dihomeomorphic to* $[0,1]^{card(C)} \setminus F$*, for a certain subset* F *which is the union of a finite set of open rectangles:*

$$F = \bigcup_{i=1}^{n} \prod_{t \in C} U_{it} \qquad \text{where } t \in C \text{ and } U_{it} \subset [0,1] \text{ are open intervals}$$

5 Geometric study of run-time properties of concurrency

5.1 Deadlocks and unsafe areas

Definition 5.1 Let X be a local po-space. Let $\mathcal{F} \subset X \cup \{\infty\}$ be a set of *final* points. Then $x \in X$ is a *deadlock with respect to* $\mathcal{F}$ if $\uparrow x = \{x\}$ and $x \notin \mathcal{F}$. A point $x \in X$ is *safe with respect to* $\mathcal{F}$ if $\uparrow x \cap \mathcal{F} \neq \emptyset$. A point is *unsafe with respect to* $\mathcal{F}$ if it is not safe.

We leave it to the reader to see that the geometric realization functor realizes the notions safe, unsafe, deadlock etc. as one would want it.

Proposition 5.2 *Let* $\mathcal{C}_1 \triangleright_f \mathcal{C}_2$ *be an* f*-delooping. Then the maps* B_s *from* $B_{\mathcal{C}_1}$ *to* $B_{\mathcal{C}_2}$ *defined by choosing a derivation* s *of the relation* $\mathcal{C}_1 \triangleright_f \mathcal{C}_2$ *are dimaps which map deadlocks to deadlocks and for all* $x \in B_{\mathcal{C}_1}$ $B_s(\uparrow x) \subseteq \uparrow B_s(x)$*.*

5.3 Remark If $\mathcal{C}$ has no loops, then by Prop. 4.9 there are no proper dipaths in $B_{\mathcal{C}}$, since a dimap γ from $\mathbb{R}^+$ to $B_{\mathcal{C}}$ would be an increasing path in a compact subset of $\mathbb{R}^n$ so it would converge to a point p. For any neighbourhood $W(p)$, $\gamma^{-1}(W(p))$ contains a half-straight line.

5.2 Minimal finitary approximation of a looping process

In [2] it is proven that when we only allow finite trajectories, it suffices to consider finitely many deloopings:

Theorem 5.4 *Let* $\mathcal{C}$ *be a concurrent system. Assume a set of final states* $\mathcal{F} \subset B_{\mathcal{C}}$ *is non-empty and* $\infty \notin \mathcal{F}$*. Then the unsafe area of* $B_{\mathcal{C}}$ *to* $\mathcal{F}$ *can be found as the intersection of the projections of the unsafe areas* $B_{\mathcal{C}_i}$ *of finitely many deloopings* $\mathcal{C}_i$ *of* $\mathcal{C}$*.*

When $\infty \in \mathcal{F}$, we need infinite trajectories and thus po-proper dipaths.

Theorem 5.5 *Let* $B_{\mathcal{C}}$ *be the realization of a concurrent system* $\mathcal{C}$*. A point* $x \in B_{\mathcal{C}}$ *is safe with respect to the set* $\mathcal{F} = \{\infty\}$ *if and only if there is a delooping* $\mathcal{C}_1 \triangleright_f \mathcal{C}$ *and a corresponding projection* $\Pi : B_{\mathcal{C}_1} \to B_{\mathcal{C}}$ *such that*

there is an $\widetilde{x} \in B_{\mathcal{C}_1}$ *with* $\Pi(\widetilde{x}) = x$ *and a dipath* $\gamma : I \to B_{\mathcal{C}_1}$ *with* $\gamma(t_0) = \widetilde{x}$ *and* $t_1, t_2 \in I$ *such that* $t_0 < t_1 < t_2$ *and* $\Pi(\gamma\, t_1) = \Pi(\gamma\, t_2)$.

For the proof of Theorem 5.5, we need an auxiliary definition and three lemmas.

Definition 5.6 A proper dipath $\gamma : \mathbb{R}_+ \to B_{\mathcal{C}}$ is *eventually periodic* if there are non negative real numbers p and T such that for all $t \geq T$, $\gamma(t+p) = \gamma(p)$.

We subdivide $B_{\mathcal{C}}$ into finitely many k-dimensional cubes, $k \leq n$ and get a translation from "continuous to discrete":

Lemma 5.7 *Let* $\gamma : \mathbb{R}^+ \to B_{\mathcal{C}}$ *be a dipath in* $B_{\mathcal{C}}$, *where* $B_{\mathcal{C}}$ *is the geometric realization of a concurrent system. Then there is a (non-unique) choice of an ordered set* $L_1, L_2, \ldots, L_k, \ldots$ *of cubes in the canonical subdivision of* $B_{\mathcal{C}}$ *such that*

- $\gamma(I) \subset \bigcup_i L_i$ *and the ordering on the cubes is by the order in which* γ *traverses them.*
- *There is a dipath which traverses all these cubes in the same order as* γ *and intersects their central points* c_i.
- *Let* $x \in L_k$. *Then there is a dipath* γ *which intersects* x *and traverses the cubes* L_i *in the specified order.*

Lemma 5.8 *Let* $x \in B_{\mathcal{C}}$. *If there is a proper dipath* $\gamma : \mathbb{R}_+ \to B_{\mathcal{C}}$ *with* $x \in B_{\mathcal{C}}$ *then there is an eventually periodic dipath through* x.

Hence, all safe points can be found in the finite deloopings, even if we allow and in fact prescribe infinite behaviour. Moreover, since a point p is safe only if there is a cube L with $p \in L$ and such that the central point of L is safe, we only have to consider central points of cubes. But how do we know when to stop looking for more safe points, i.e., to stop delooping further? This is covered in the following proposition:

Proposition 5.9 *Let* p *be a central point of a cube in* $B_{\mathcal{C}}$. *If* $B_{\mathcal{C}'}$ *is the geometric realization of a delooping* $\mathcal{C}' \triangleright_f \mathcal{C}$ *and* $\Pi : B_{\mathcal{C}'} \to B_{\mathcal{C}}$ *is a corresponding projection such that there is a dipath* $\gamma : I \to B_{\mathcal{C}'}$ *which projects to a periodic path:* $p = \Pi(\gamma(t_1)) = \Pi(\gamma(t_2))$, $t_1 \neq t_2$, *and if there are no smaller (wrt.* $\preccurlyeq_{\mathcal{C}}$*) deloopings with a dipath projecting to a periodic path through* p, *then there is an increasing sequence of deloopings* $\mathcal{C}_1 \preccurlyeq_{\mathcal{C}} \mathcal{C}_2 \preccurlyeq_{\mathcal{C}} \ldots \mathcal{C}_{m-1} \preccurlyeq_{\mathcal{C}} \mathcal{C}_m = \mathcal{C}'$ *such that*

- $\mathcal{C}_1$ *has at most one copy of each loop in* $\mathcal{C}$.
- *If* $\mathcal{C}_i \preccurlyeq_{\mathcal{C}} \mathcal{C}_* \preccurlyeq_{\mathcal{C}} \mathcal{C}_{i+1}$, *then* $\mathcal{C}_* = \mathcal{C}_i$ *or* $\mathcal{C}_* = \mathcal{C}_{i+1}$.

- *The future of p is increased at each further delooping in the following sense:* $\Pi_i(\uparrow \Pi_i^{-1}(p)) \subsetneq \Pi_{i+1}(\uparrow \Pi_{i+1}^{-1}(p))$ *where* $\Pi_i : B_{\mathcal{C}_i} \to B_{\mathcal{C}}$.

This answers the question when to stop delooping further: when the future is not increasing anymore. Hence, to see if there is a periodic dipath containing a point p, we need to study the future of $\Pi^{-1}(p)$ in further and further deloopings, see if the projections of these sets are increasing and if one contains p. Finding futures is a reachability question, and this can be studied using the deadlock algorithm on $B_{\mathcal{C}}$ with the local partial order (i.e. time) reversed.

Acknowledgement

It is our pleasure to thank Martin Raussen for many very helpful discussions along the way. Moreover, the development of ditopology itself has been and still is a joint work with Martin, Eric Goubault and with others.

References

[1] E. W. Dijkstra, Co-operating sequential processes, in: F. Genuys ed., *Programming Languages* (Academic Press, New York, 1968) 43–110.

[2] L. Fajstrup, Loops, ditopology and deadlocks, *Mathematical Structures in Computer Science* **10** (2000) 1–22.

[3] L. Fajstrup, E. Goubault and M. Raußen, *Algebraic Topology and Concurrency*, Report R-99-2008 (Department of Mathematical Sciences, Aalborg University, 1999), 47 pp.

[4] L. Fajstrup, E. Goubault and M. Raußen, Detecting deadlocks in concurrent systems, in: D. Sangiorgi and R. de Simone, eds., *CONCUR'98; Proceedings of the 9th Int. Conf. on Concurrency Theory, Nice, France* (Lect. Notes Comp. Science **1466**, Springer-Verlag, 1998) 332–347.

[5] G. Giertz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove and D. S. Scott, *A Compendium of Continuous Lattices*, (Springer-Verlag, Berlin Heidelberg New York, 1980).

[6] E. Goubault, *The Geometry of Concurrency*, Ph.D. thesis, Ecole Normale Superieure, Paris (1995).

[7] J. Gunawardena, Homotopy and concurrency, *Bulletin of the EATCS* **54** (1994) 184–193.

[8] R. Penrose, *Techniques of Differential Topology in Relativity*, Conference Board of the Mathematical Sciences, Regional Conference Series in Applied Mathematics **7** (SIAM, Philadelphia, USA, 1972).

[9] S. Sokołowski, Investigation of concurrent processes by means of homotopy functors, (to appear in *Mathematical Structures in Computer Science*, 2000).

A Study on Semi-Sheaves Associated to Transition Systems Representing Reactive Systems

Ana Isabel de Azevedo Spinola

Dept. of Analysis
Universidade Federal Fluminense
Niterói, RJ, Brazil
Email: spinola@vm.uff.br

Edward Hermann Haeusler

Dept. of Informatics
Pontifícia Universidade Católica
Rio de Janeiro, RJ, Brazil
Email: hermann@inf.puc-rio.br

Abstract

The aim of this paper is to develop a study on the behaviour of reactive systems by means of sheaf theory. A concept of semi-topology can be defined in order to categorically transform transition systems modeling reactive systems into "semi-sheaves". This induces a category $\mathcal{SS}$(Act,I) that is an elementary topos. We present in this paper the idea behind this construction and the proof that this category is really a topos exhibiting its terminal object and its subobject classifier. Then we investigate the internal logic of $\mathcal{SS}$(Act,I) proving that it is classical, as the topos is boolean.

1 Introduction

The idea of using the theory of sheaves to model concurrent computational systems is not a very recent one. For example, Monteiro in [7] (1986), Goguen in [2] (1992) and Winskel in [5](1993) and in [1] (1996) have all used sheaves (or pre-sheaves) in order to model situations of concurrency. We also wanted to study reactive systems from the viewpoint of sheaf theory, although in a different approach from those authors. A *sheaf* (of germs) over a topological space I (called *base space*) is a pair (A, f) where A is a topological space (called *stalk space*) and $f : A \longrightarrow I$ is a continuous function that is

This is a preliminary version. The final version can be accessed at
URL: http://www.elsevier.nl/locate/entcs/volume39.html

a local homeomorphism.[1] "Homeomorphic" means "topologically indistinguishable", so the local homeomorphism of this definition is responsible for the fact that sheaves are always suitable for studying geometric applications where one goes from local to global properties. We can think of A as being a pasting of little pieces, each one homeomorphic to an open set of I.[2] There is a close relation between the concepts of reactivity and concurrency. When we put the enviroment of a reactive system and the system itself together into a larger system, we have a concurrent system. Conversely, in any concurrent system composed by independent communicating processes, each process can be thought of as a reactive system in the sense that each process mantains a reactive interaction with the other processes, receiving stimuli and reacting to them through actions. In order to conduct a precise study of the external behaviour of reactive systems we use transition systems as a first semantics, and then we add the geometric notion of "semi-sheaf" to broaden our intuition. The geometric meaning behind the concept of a "semi-sheaf" is given by the "semi-topology" of the stalk space and the topology of the base space. We introduce those notions in this paper.

In 1976 Dana Scott introduced in [10] a mathematical theory of computation where the values involved in a computation should live in a specific structure called *(Scott) Domain*. In this way he created the theory of domains. We point two important references about this subject. A book written by Stoy [14] using the theory of domains in the semantic of programming languages and [13], with a mathematical theory of domains. Each state of a transition system is described by its attributes, which are data represented by variables having certain types. The base space of the semi-sheaf will be defined as the set of all the possible values assumed by variables of all data types. So it seems natural to choose the domain structure for the base space as it models the notion of approximation and computation.

Definition 1.1 A complete partial order (cpo) D is a *Scott-Ershov Domain*, or simply a *Domain* if D is an algebraic cpo and if the set $\{a, b\} \subseteq D_c$ is consistent and $a \sqcup b = \sqcup\{a, b\}$ exists in D (where D_c consists of the compact (finite) elements of D).

Definition 1.2 The *Scott Topology* for a domain D=$(D, \sqsubseteq, \perp)$ is given by $U \subseteq D$ is open if:

Alexandrof condition $x \in U$ and $x \sqsubseteq y \Rightarrow y \in U$

Scott condition $A \subseteq D$ directed and $\sqcup A \in U \Rightarrow (\exists x \in A)((x \in U)$.

Furthermore, the collection of semi-sheaves over a topological space I to-

[1] This formulation was first proposed and developed by Henri Cartan in 1950. There are good historical notes about the origin of sheaves, as well as of topoi, in [4]

[2] There is an analogy between the sheaf and the topological manifold, which is the generalization of the concept of surface. The identification between the two concepts is very well explained by Mac Lane in [6].

gether with the collection of morphisms between such semi-sheaves form a category $\mathcal{SS}(\text{Act}, I)$, which, in turn, is a topos. This means, among other things, that this category has an internal logic which will be interpreted in this work. We have drawn the conclusion that the topos is boolean (although not bivalent), which means that the logic governing truth in $\mathcal{SS}(\text{Act}, I)$ is the usual (classical) logic.

2 Studying Reactive Systems through the Category of Transition Systems

We restrict ourselves to computational reactive systems. Thus we can assume the existence of an enumeration $k : \mathbb{N} \longrightarrow RS$ of RS, the set of computational reactive systems. Reactive systems interact with their environment, receive stimuli and react to them through actions, or rather, their actions are the reactions. Each action changes the state of the system. A run of a reactive system may never stop, and its set of inputs and outputs can be infinite. Classical examples are operational systems. Their behaviour can be very well captured by transition systems. There are different kinds of transition systems (simple, labelled, parametrized, with dependency, timed, etc.). For us, as the description of a reactive system will be a pair (A, f) – a semi-sheaf – the structure of the system should be well reflected in the structure of (A, f). So, our definition of transition system is a little richer than the standard one in the sense that it contains information about the attributes that describe the states during the evolution of the system. In other words, a structure storing the values of the variables which represent those attributes of the states, and of course, basic information about the set of states and the transition relation between states.

Definition 2.1 A *transition system* is a structure

$$S = (\text{K}, q_1, \text{Act}, \rightarrow, \text{Atrib}, \{v_{a_i} : a_i \in Atrib\}), \quad \text{where:}$$

K $=\{ q_k \}_{k \in \mathbb{N}}$ is an enumerable set whose elements are the states of the system.

$q_1 \in$ K is the initial state. It determines where the run of the system begins.

Act is the set of transition labels. Its elements are the possible names of the actions performed by the system, and it includes τ and **idle**:

- τ is the label of any internal action performed by the system. We mean by "internal" an action that cannot be observed by an external observer.
- **idle** is the label of any transition without an action, any transition that doesn't change the configuration of the state.

$\rightarrow \subseteq K \times Act \times K$ is a relation between states such that the sucessor of a state is determined when an action is performed. $\rightarrow$ is said to be the transition relation and if $(q_i, t, q_k) \in \rightarrow$ then we represent that by $q_i \stackrel{t}{\rightarrow} q_k$. When $t = idle$ then $q_i \stackrel{idle}{\rightarrow} q_i$.

Atrib $=\{ a_1, \ldots, a_n \}$ is the set of attributes that describe each state of the

system.

v_{a_i} :K $\to [\![\tau_{S_i}]\!]$ is a function that maps a state q to the value of the data type of the variable that describes attribute a_i of state q. $v_{a_i}(q) \in [\![\tau_{S_i}]\!]$ where τ_{S_i} is the data type of the variable S_i which describes attribute a_i of q. $[\![\tau_{S_i}]\!]$ is naturally a Scott domain with partial order $\sqsubseteq$.

If $q \xrightarrow{a} q'$ then it is not the case that $v_a(q') \sqsubseteq v_a(q)$, because the transitions represent computable actions, and thus there cannot be any loss of information. A reactive system has the following structure:

$$S = (\mathrm{K}, q_1, \mathrm{Act}, \to, \mathrm{Attrib}, \{v_{a_i} : a_i \in \mathit{Attrib}\})$$

Definition 2.2 Let us consider

$$T_1 = (K_1, q_{11}, \mathrm{Act}_1, \longrightarrow_1, \mathrm{Attrib}_1, \{v_{a_i} : a_i \in \mathrm{Attrib}_1\})$$
$$T_2 = (K_2, q_{12}, \mathrm{Act}_2, \longrightarrow_2, \mathrm{Attrib}_2, \{v_{a_j} : a_j \in \mathrm{Attrib}_2\})$$

transition systems. A *morphism* $f : T_1 \longrightarrow T_2$ is a triple $f = (\sigma, \lambda, \alpha)$ where:

$\sigma : K_1 \longrightarrow K_2$ is such that $\sigma(q_{11}) = q_{12}$

$\lambda : \mathbf{Act}_1 \longrightarrow \mathbf{Act}_2$ is a partial function such that:

$$(q, a, q') \in \longrightarrow_1 \text{ then } \begin{cases} (\sigma(q), \lambda(a), \sigma(q')) \in \longrightarrow_2 & \text{if } \lambda(a) \text{ is defined} \\ \sigma(q) = \sigma(q') & \text{if } \lambda(a) \text{ is undefined} \end{cases}$$

$\alpha : \mathbf{Attrib}_1 \longrightarrow \mathbf{Attrib}_2$ is a function which associates attributes of states of T_1 to attributes of states of T_2.

We define the composition of two morphisms between transition systems componentwise, that is, if $f = (\sigma, \lambda, \alpha) : T_1 \longrightarrow T_2$ and $g = (\sigma', \lambda', \alpha') : T_2 \longrightarrow T_3$, then $g \circ f = (\sigma' \circ \sigma, \lambda' \circ \lambda, \alpha' \circ \alpha) : T_1 \longrightarrow T_3$. If $1_K : K \longrightarrow K$ is the identity function between states, $1_{\mathrm{Act}} : \mathrm{Act} \longrightarrow \mathrm{Act}$ the identity function between actions, and $1_{\mathrm{Attrib}} : \mathrm{Attrib} \longrightarrow \mathrm{Attrib}$, then $(1_K, 1_{\mathrm{Act}}, 1_{\mathrm{Attrib}})$ is the identity morphism. Let **TS** be the category whose objects are transition systems with these morphisms defined above. We restrict ourselves to the subcategory **TS(Act)** whose objects are transition systems over the same set, Act, of transition labels and with its morphisms defined as above. It is clear that in this case we have $\lambda : \mathrm{Act} \longrightarrow \mathrm{Act}$.

Definition 2.3 Let S be a transition system. The *unfolded tree* of S is a rooted tree whose vertices represent instances of states (finite runs) of the system. The root represents the initial state (empty run <>) of the system, and if there is a transition from one state to another then there is an edge between the respective vertices representing these instances of states.

Notation: $UT(S)$ = Unfolded tree of S

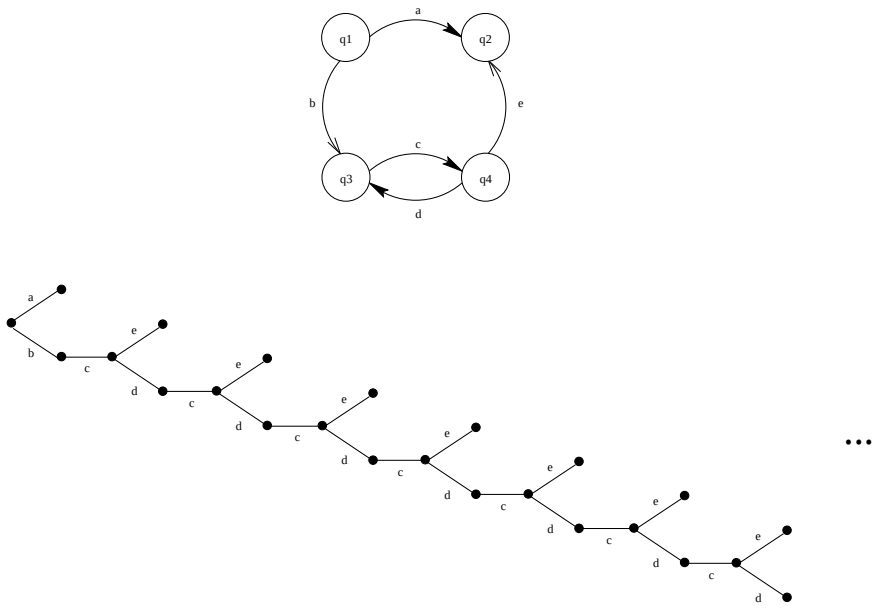


Fig. 1. The unfolded tree of a transition system with initial state q_1

3 Semi-Sheaf Associated to a Reactive System

3.1 The origin of semi-sheaves

To each reactive system S_j of the countabe set RS of all such systems we will associate a semi-sheaf (A_j, f). Each S_j has a formal version behaving as a transition system over a set of labels Act. For simplicity we refer to the unfolded tree of a reactive system, instead of referring to the unfolded tree of the transition system which represents the reactive system. Let A_j be the set of vertices of the unfolded tree of S_j. The topology of A_j will be based upon the actions performed by the system, reflecting the external behaviour of the system and a notion of proximity between computations. We have chosen a notion of proximity which is represented by the following family which is a topological base in A_j:

β = collection of vertices of all connected subgraphs of the unfolded tree of S_j

Definition 3.1 A topological base $\mathcal{B}$ in X generates a topology in X considering as open sets exactly the subsets of X that are unions of elements of $\mathcal{B}$.

So we have available the topology generated by the base. At this point a naturally arising question is why define "semi-topology" – as announced in the introduction – if we have a real topology in our hands? There is a technical problem here. According to the last definition, a set defined as the union of two disjoint connected subgraphs of the unfolded tree would be an open set. In this case, there would exist a pair x, y of vertices of this open

set such that there is no way connecting x to y (if x is in an open set disjoint from the one containing y). At this point we have to decide whether to stay with the topology and lose our idea of proximity. We have chosen to propose another concept very similar to that of a topology, in order to avoid open sets containing unconnected vertices. In this way we lose the topology and have to define the notion of *semi-topology*. Henceforth we will be dealing with a semi-topology, instead of a topology, for the stalk space A_j.

Definition 3.2 Let $X \neq \emptyset$ be a set. A family $\mathcal{B} \subseteq \mathcal{P}(X)$ is a *semi-topological base* in X if $\bigcup \mathcal{B} = X$ and (there exists an enumeration of $\mathcal{B}$ such that $(\bigcup_{j \leq n} B_j) \cap B_{n+1} \neq \emptyset, \quad \forall n$) and (if $B_1, B_2 \in \mathcal{B}$ and $x \in B_1 \cap B_2$ then $\exists B_3 \in \mathcal{B}$ such that $x \in B_3 \subseteq B_1 \cap B_2$).

Definition 3.3 Let A be a non-empty set. A *semi-topology* over A is a family $\tau \subseteq \mathcal{P}(A)$ such that:

(i) $A, \emptyset \in \tau$

(ii) If $U, V \in \tau$ then $U \cap V \in \tau$

(iii) Let J be an arbitrary set of indexes. If $U_i \in \tau$ for all $i, \in J$ and any two of them are not pairwise disjoint then $\bigcup_{i \in J} U_i \in \tau$.

Each element of a semi-topology is called an *s-open* set over A. A non-empty set A for which a semi-topology τ has been specified is called a *semi-topological space*.

Definition 3.4 A semi-topological base $\mathcal{B}$ in X generates a semi-topology in X considering as s-open sets exactly the subsets of X that are unions of pairwise not disjoint elements of $\mathcal{B}$.

The collection of vertices of all connected subgraphs of the unfolded tree of $S_j - \beta -$ is a semi-topological base generating a semi-topology for A_j. Now we should redefine the concepts of continuous functions. Although a "truth topology" is missing, we still have the notion of proximity. We will make precise the meaning of "continuous function" between two semi-topological spaces, and of "continuous function" between a semi-topological space and a topological one.

Definition 3.5 Let X, Y be semi-topological spaces. $f : X \longrightarrow Y$ is called an *s-continuous* function if given an s-open set V over Y then $f^{-1}(V)$ is s-open over X.

Definition 3.6 Let X be a semi-topological space and Y a topological space. $f : X \longrightarrow Y$ is called an *ss-continuous* function if given an open set V in Y then $f^{-1}(V)$ is s-open in X. Analogously, $g : Y \longrightarrow X$ is called an *ss-continuous* function if given an s-open set V in X then $g^{-1}(V)$ is open in Y. $f : X \longrightarrow Y$ is called a *semi-homeomorphism* if f is bijective and ss-continuous, and f^{-1} is ss-continuous. f is called a *local semi-homeomorphism* if for all $x \in X$, exists an s-open set $\mathcal{O}$ in X, $x \in \mathcal{O}$ such that $f_{|\mathcal{O}}$ (f restricted

to $\mathcal{O}$) is a semi-homeomorphism over an open set in Y.

Definition 3.7 Let I be a topological space and A a semi-topological space. A *semi-sheaf* is a pair (A, f) where $f : A \longrightarrow I$ is an ss-continuous function that is a semi-local homeomorphism. A is said to be the *stalk space* and I is said to be the *base space* of the semi-sheaf.

3.2 Stalk Space of the Semi-Sheaf

The set of all states of all systems in RS is an enumerable set. [3] We will display its elements in the following way:

$$\{ \underbrace{q_{11}, q_{21}, q_{31}, \ldots,}_{K(S_1) \text{ states of } S_1} \cdots \underbrace{q_{1j}, q_{2j}, q_{3j}, \ldots}_{K(S_j) \text{ states of } S_j} \}$$

The elements of A_j will be labelled as $A_j = \{v_t^{ij}\}_{t \in \text{Traces}(UT(S_j))}$ where ij is the index of the state q_{ij} represented by this vertex in $UT(S_j)$; and t is a trace; in other words, t is the sequence of actions performed from the initial state q_{1j} to q_{ij}. A_j is a semi-topological space with the semi-topology generated by the semi-topological base β (consisting of the collection of vertices of all conected subgraphs of the unfolded tree of S_j). A_j is the stalk-space of the semi-sheaf representing the reactive system S_j.

3.3 Base Space of the Semi-Sheaf

Let Attrib_j be the set of attributes of the states of S_j. Attrib_j is a finite set and we will write $\text{Attrib}_j = \{ a_1, \ldots, a_n \}$. Consider the following sets, for each $j \in \mathbb{N}$:

$$D_{Attrib_j} = \left\{ \begin{pmatrix} v_{a_1}(q_{kj}) \\ \vdots \\ v_{a_n}(q_{kj}) \end{pmatrix} / q_{kj} \in K(S_j) \right\}$$

Each coordinate of each element of D_{Attrib_j} ranges over the Scott domain $[\![\tau_{s_i}]\!]$ (i=1,...,n) of possible values that are assumed by the data type of variable s_i describing the attribute a_i of the state. For each system S_j of RS we have, thus, a domain D_{Attrib_j}. Let the base space I be:

$$I = D_{\text{Attrib}_1} \times D_{\text{Attrib}_2} \times \ldots = \Pi_{j=1}^{\omega} D_{\text{Attrib}_j}$$

I is an arbitrary product of domains, and hence is a Scott domain. This is proved in [11]. So, I is naturally a topological space with the Scott topology, which is the product topology (Tychonoff) of the Scott topologies of each domain, as it is well presented in [13]. We use a special notation to refer to

[3] as it is the enumerable union of enumerable sets (sets of states of each system in RS.)

some elements of I:

$$I_{q_{i1}} = \begin{pmatrix} v_{\alpha_1}(q_{i1}) \\ \vdots \\ v_{\alpha_l}(q_{i1}) \end{pmatrix} \in D_{\text{Attrib}_1} \text{ where } \text{Attrib}_1 = \{\alpha_1, \ldots, \alpha_l\}$$

$$\vdots$$

$$I_{q_{ij}} = \begin{pmatrix} v_{a_1}(q_{ij}) \\ \vdots \\ v_{a_n}(q_{ij}) \end{pmatrix} \in D_{\text{Attrib}_j} \text{ where } \text{Attrib}_j = \{a_1, \ldots, a_n\}$$

$$\vdots$$

When j varies we have the information of the values of the attributes of the same state q_i in different systems $S_1, S_2, S_3, \ldots$ of RS. Notice that if we define $I_{q_i} =< I_{q_{i1}}, I_{q_{i2}}, I_{q_{i3}}, \ldots >$, then each I_{q_i} is an element of I, because each $I_{q_{ij}} \in D_{\text{Attrib}_j}$.

$$\begin{aligned} I_{q_1} &= < I_{q_{11}}, I_{q_{12}}, I_{q_{13}}, \ldots > \\ I_{q_2} &= < I_{q_{21}}, I_{q_{22}}, I_{q_{23}}, \ldots > \\ &\vdots \end{aligned}$$

3.4 Function Between Stalk and Base Space

Let A_j be the stalk space associated to a transition system S_j. Let us define f in the following way:

$$\begin{aligned} f : A_j &\longrightarrow I \\ v_t^{ij} &\mapsto \uparrow < I_{q_{i1}}, \ldots, I_{q_{ij}}, \ldots > = \uparrow I_{q_i} \end{aligned}$$

where $\uparrow J = \{K / J \sqsubseteq K\}$. If q_i doesn't exist in S_j then $I_{q_{ij}}$ is undefined, in other words, is equal to $\{ \perp \}$.

Theorem 3.8 *f is ss-continuous and f is a semi-local homeomorphism.*

Proof. This is proved in [11]. □

4 The Category of Semi-Sheaves

Let A_j be the set of vertices of the unfolded tree of a transition system S_j and $f : A_j \longrightarrow I$ be a semi-local homeomorphism such that $f(v_t^{ij}) = \uparrow I_{q_i}$. We will now define a functor F from the category **TS(Act)** in order to construct a category of semi-sheaves. Having a transition system S_j, we can always find

a semi-sheaf $F(S_j) = (A_j, f)$. Let $\mathcal{SS}$(Act,I) be the category whose objects are semi-sheaves over I associated to transition systems with label set Act. The question remains: "what is the action of the functor F over a morphism $(\lambda, \sigma, \alpha)$ of **TS(Act)**? What should $k = F(\lambda, \sigma, \alpha)$ satisfy in order to be the image of a morphism via this functor F?" We answer those questions in the definition of the $\mathcal{SS}$(Act,I)-morphisms below.

$$\begin{aligned} F : \mathbf{TS(Act)} &\longrightarrow \mathcal{SS}(Act, I) \\ S_j &\mapsto \quad F(S_j) = (A_j, f) \\ S_j \overset{(\lambda,\sigma,\alpha)}{\longrightarrow} S_l &\mapsto \quad (A_j, f) \overset{k}{\longrightarrow} (A_l, g) \end{aligned}$$

Objects of $\mathcal{SS}$(Act,I): Semi-sheaves (A,f) over I such that

(i) A is the set of vertices of the unfolded tree of some transition system over Act as a semi-topological space (with the semi-topology described in section 3) [4] .

(ii) I is a set containing denotable values, with the Scott topology, as described in section 3.

(iii) The function f is defined as $f(v_t^{ij}) = \uparrow I_{q_i}$ where $\uparrow I_{q_i} =< I_{q_{i1}}, \ldots, I_{q_{ij}}, \ldots >$.

Morphisms of $\mathcal{SS}$(Act,I): $k : (A, f) \longrightarrow (B, g)$ s-continuous such that $f = g \circ k$.

Considering $(\lambda, \sigma, \alpha)$ as a morphism between the two transition systems S_j and S_l, $(\lambda, \sigma, \alpha) : S_j \longrightarrow S_l$ then we can define $F(\lambda, \sigma, \alpha)$ as:

$$\begin{aligned} F(\lambda, \sigma, \alpha) : F(S_j) &\longrightarrow F(S_l) \\ v_t^{ij} &\mapsto \quad v_r^{pl} \end{aligned}$$

(where v_r^{pl} is the vertex representing state q_{pl} of S_l in its unfolded tree with trace r) if:

- $\sigma(q_{ij}) = q_{pl}$
- $v_{a_n}(q_{ij}) = v_{\alpha(a_n)}(\sigma(q_{ij}))$, $\forall a_n \in \text{Attrib}_1$, because k should be such that $g \circ k = f$, thus vertices of A should be mapped by f to the same attribute values if mapped by $g \circ k$; and it signifies that states q_{ij} and $\sigma(q_{ij})$ assume the same attribute values in the corresponding attributes.
- Se $t =< b_1, \ldots, b_m >$ then we have three cases:

(i) $\lambda(b_l)$ is defined $\forall b_l$. In this case, $r =< \lambda(b_1), \ldots, \lambda(b_m) >$

(ii) $\lambda(b_l)$ is undefined for any b_l. In this case, $r =<>$.

(iii) $\exists y, b_y$ is undefined. In this case, $r =< \lambda(b_1), \ldots, \lambda(b_{y-1}), \lambda(b_{y+1}), \ldots, \lambda(b_m) >$.

5 $\mathcal{SS}$(Act,I) is a Topos

Theorem 5.1 *$\mathcal{SS}$(Act,I) has a terminal object.*

[4] sets of vertices of connected subgraphs

Proof(Sketch):

The terminal object of $\mathcal{SS}$(Act,I) is denoted by (Te, p) where Te is the set of vertices of the tree which contains all the unfolded trees of the other systems. For convenience, we consider Te as being the set of vertices of the unfolded tree of S_1, the first system of the enumeration. Labels of the original trees do not change in Te. Thus, if v_t^{ij} is the label of the vertex representing state q_i in the unfolded tree of S_j, then this vertex (which certainly exists in Te) receives the same label v_t^{ij}. The only exception is the root, whose label is $v_{<>}^{11}$. We specify the same semi-topology for Te, ie, the s-open sets are formed by vertices of connected subgraphs of the unfolded tree whose set of vertices is Te. Defining

$$\begin{aligned} p : Te &\longrightarrow \quad I \\ v_t^{ij} &\mapsto \quad \uparrow I_{q_i} \end{aligned}$$

in the same way as before one can easily prove that p is an ss-continuous function and a semi-local homeomorphism. In order to prove that (Te, p) is the terminal object of $\mathcal{SS}$(Act,I) one should first define a function

$$\begin{aligned} cp_{A_j} : A_j &\longrightarrow Te \\ v_t^{ij} &\mapsto \begin{cases} v_t^{ij} & \text{if } i \neq 1 \\ v_{<>}^{11} & \text{if } i = 1 \end{cases} \end{aligned}$$

that maps A_j to its copy in Te, for each given semi-sheaf (A_j, f).Observe that $p \circ cp_{A_j} = f$ and that cp_{A_j} is an s-continuous function. So, cp_{A_j} is an $\mathcal{SS}$(Act,I)-morphism. We can also prove that cp_{A_j} is the only morphism from A_j to Te.

□

Theorem 5.2 *$\mathcal{SS}$(Act,I) has pullbacks and exponentials.*

Proof. The proof is analogous to the one that Top(I) (spatial topos whose objects are sheaves) has pullbacks and exponentials. □

Theorem 5.3 *$\mathcal{SS}$(Act,I) has a subobject classifier.*

Proof(Sketch): We provide a guide to the proof and develop parts 1 and 3.

Part 1 Exhibit a potential classifier, an object $\Omega = (\widehat{I}, \widehat{p})$ of $\mathcal{SS}$(Act,I).

Part 2 Exhibit a morphism *true* $\top : 1 \longrightarrow \Omega$

Part 3 For each $k : A \hookrightarrow B$, exhibit the characteristic function χ_k.

Part 4 Show that the diagram of figure 2 commutes.

Part 5 Prove the final condition for being a pullback. In other words, show that whenever $Te \xleftarrow{cp_E} E \xrightarrow{g} B$ are such that $\chi_k \circ g = \top \circ cp_E$ then there exists a unique $\mathcal{SS}$(Act,I)-morphism $e : E \longrightarrow A$ such that $k \circ e = g$ and $cp_A \circ e = cp_E$.

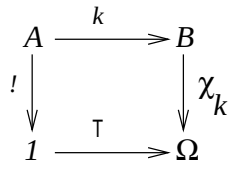


Fig. 2. Ω-axiom

□

Proof of Part 1:

We define an equivalence relation between the s-open sets of the terminal object Te. Consider $v_t^{ij} \in Te$. For $U, V \subseteq Te$, U, V s-open sets, we define:

$$U \sim_{v_t^{ij}} V \text{ iff } \exists W \text{ s-open }, v_t^{ij} \in W \text{ and } U \cap W = V \cap W$$

We can draw some conclusions about the classes:

(i) If $v_t^{ij} \in U \cap V$ then making $W = \{v_t^{ij}\}$ we have that $U \sim_{v_t^{ij}} V$.

(ii) If $v_t^{ij} \notin U \cup V$ then making $W = \{v_t^{ij}\}$ we have that $U \sim_{v_t^{ij}} V$.

(iii) If $v_t^{ij} \in U$ and $v_t^{ij} \notin V$ then we have that $U \not\sim_{v_t^{ij}} V$.

For each $v \in Te$, an equivalence relation $\sim_v$ divides Te into two classes: the one of s-open sets containing v and the other of s-open sets not containing v. In other words, given U an s-open set of Te, we have:

$$[U]_v = \begin{cases} \text{Class of s-open sets that contain } v & \text{if } v \in U \\ \text{Class of s-open sets that do not contain } v & \text{if } v \notin U \end{cases}$$

We write $[C]_v$ to designate the class of s-open sets that contain v, and $[NC]_v$ to the class of s-open sets which do not contain v. Considering

$$\Omega_{v_t^{ij}} = \{[U]_{v_t^{ij}} / U \text{ s-open set } Te\}$$

we can define the stalk space of the potential subobject classifier and $\widehat{p}$ as:

$$\widehat{I} = \{x / x \in \Omega_{v_t^{ij}}, \text{ for some } v_t^{ij}\}$$

$$\widehat{p} : \widehat{I} \longrightarrow I$$

$$\widehat{p}([U]_{v_t^{ij}}) \mapsto \uparrow I_{q_i}$$

As $\widehat{p}$ is ss-continuous and a semi-local homeomorphism, $\Omega = (\widehat{I}, \widehat{p})$ is an object of the category $\mathcal{SS}$(Act,I). Now, we should be able to see $\widehat{I}$ as the unfolded tree of some transition system. Each class of $\widehat{I}$ will label a vertex of the tree. Let us now describe the edges of the unfolded tree whose vertices are the classes of $\widehat{I}$. Consider the root of the universal tree whose set of vertices is Te, $v_{<>}^{11}$. Let the class $[C]_{v_{<>}^{11}}$ be the root of the tree we are generating. In this universal tree there exists an edge from $v_{<>}^{11}$ to $v_{a_i}^{l1}$ which is labelled by a_i, for all l which are indexes of vertices of the second level of the universal tree. In this way we have generated an edge in the tree whose set of vertices

is $\widehat{I}$, incident to the class $[C]_{v^{11}_{<>}}$ and to the class of s-open sets that contain $v^{l1}_{a_i}$. This means that:

$$\text{If } v^{11}_{<>} \xrightarrow{a_i} v^{l1}_{a_i} \text{ then } [C]_{v^{11}_{<>}} \xrightarrow{a_i} [C]_{v^{l1}_{a_i}}$$

Besides such edges which are incident to the root $[C]_{v^{11}_{<>}}$, we add one more, labelled by a_1, incident to $[NC]_{v^{11}_{<>}}$. From the third level on, we repeat the procedure (Figure 3).

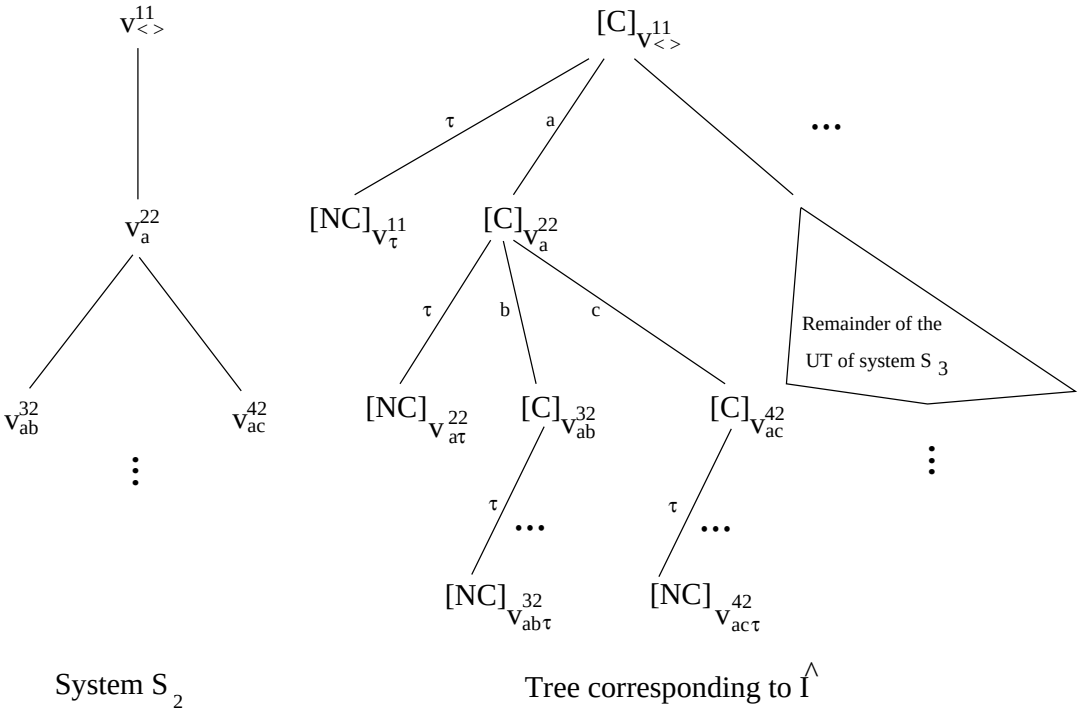


Fig. 3. Tree whose vertices are the elements of $\widehat{I}$

□

Proof of Part 3:

Let A be an open subset of B, that is, A is a set of vertices of a connected subgraph of B. Let k be $k : A \overset{k}{\hookrightarrow} B$ monic such that $q \circ k = f$. The characteristic function $\chi_k : (B, q) \longrightarrow (\widehat{I}, \widehat{p})$ should be such that $\widehat{p} \circ \chi_k = q$. Recall that as (B, q) is a semi-sheaf, then given $v^{ij}_t \in B$, we know that $\exists S_{v^{ij}_t}$ neighbourhood of v^{ij}_t such that $q_{|S_{v^{ij}_t}}$ is a semi-homeomorphism. And also we have that as Te is the terminal object, then there exists a unique morphism $cp_B : B \longrightarrow Te$.

Let χ_k be defined as:

$$\chi_k(v^{ij}_t) = [p^{-1}(q(A \cap S_{v^{ij}_t}))]_{cp_B(v^{ij}_t)}$$

□

The very first definition of topos presented by Lawvere and Tierney in

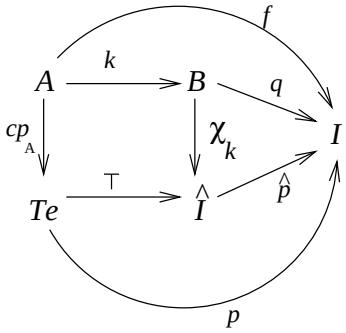


Fig. 4. Definition of χ_k

1969 was the following one:

Definition 5.4 An *(elementary) topos* is a category $\mathcal{C}$ satisfying:

(i) $\mathcal{C}$ is finitely complete.

(ii) $\mathcal{C}$ is finitely co-complete.

(iii) $\mathcal{C}$ has exponentiation.

(iv) $\mathcal{C}$ has a subobject classifier.

Remark that we can substitute condition (1) of this definition by "$\mathcal{C}$ has a terminal object and pullbacks". It was later shown that conditions (1),(3) and (4) impliy condition (2). Because of that, for some authors a topos is simply a cartesian closed category[5] with a subobject classifier.

In this way, we have proved above that $\mathcal{SS}$(Act,I) is an elementary topos.

6 Analysis of the Characteristic Function

We can now analyse the action of the characteristic function over each vertex v_t^{ij} (how it classifies such a vertex).

$$\chi_k(v_t^{ij}) = [p^{-1}(q(A \cap S_{v_t^{ij}}))]_{cp_B(v_t^{ij})}$$

Case $v_t^{ij} \in A$**:** Then $v_t^{ij} \in (A \cap S_{v_t^{ij}})$ and so $\chi_k(v_t^{ij}) = [C]_{v_t^{ij}}$.

Case $v_t^{ij} \notin A$**:** Then we have two subcases to analyse. First, if $A \cap S = \emptyset$ then $\chi_k(v_t^{ij}) = [\emptyset]_{v_t^{ij}} = [NC]_{v_t^{ij}}$. And finally, if $A \cap S \neq \emptyset$ then $S \cap A = R$ for some set of vertices R. Then $\chi_k(v_t^{ij}) = [p^{-1}(q(R))]_{v_t^{ij}} = [NC]_{v_t^{ij}}$.

We expected to capture a better classification according to how close a general vertex, v_t^{ij}, is to A, more subtle distinctions about the proximity of a vertex from a subsystem, but it does not seem to be the case.

[5] A category satisfying conditions (1) and (3)

7 Properties of the topos $\mathcal{SS}$(Act,I)

It is important to stress that although each stalk yields two classes of equivalence:

$$\top(v_t^{ij}) = [Te]_{v_t^{ij}} = [C]_{v_t^{ij}}, \qquad \forall v_t^{ij} \text{ in } Te$$
$$\bot\,(v_t^{ij}) = [NC]_{v_t^{ij}}, \qquad \forall v_t^{ij} \text{ in } Te$$

it is not the case that $\mathcal{SS}(\mathrm{Act}, I)$ is a bivalent topos. We recall that a bivalent topos (two-valued) is a non-degenerate one (true $\ncong$ false) in which $true$ and $false$ are the only truth-values (elements of Ω). But there are other $\mathcal{SS}(\mathrm{Act}, I)$-arrows $x : (Te, p) \longrightarrow (\widehat{I}, \widehat{p})$ (elements of Ω) different from $true$ and $false$.

We want now to characterize the behaviour of subobjects in $\mathcal{SS}$(Act,I).

The operations of complement, intersection and union on the collection of subobjects of a semi-sheaf (A, f) are defined in terms of the truth arrows – negation, conjunction, disjunction and implication – which are, in turn, character arrows in $\mathcal{SS}$(Act,I). We will only present as examples, negation and conjunction as $\mathcal{SS}$(Act,I)-arrows. We also present the operations of complement and intersection in (Sub(A,f),$\subseteq$).

(i) Two truth-arrows:

(a) Negation:

$\neg : (\widehat{I}, \widehat{p}) \longrightarrow (\widehat{I}, \widehat{p})$ is the unique $\mathcal{SS}$(Act,I)-arrow such that the diagram in Figure 5 is a pullback in $\mathcal{SS}$(Act,I). Thus, $\neg = \chi_{\bot}$ (where $\bot$ itself is the character of $! : \emptyset \longrightarrow Te$).

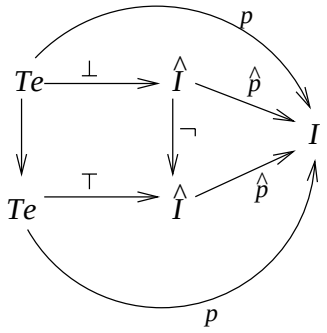


Fig. 5. Negation as a characteristic function

(b) Conjunction:

$\wedge : \widehat{I} \times \widehat{I} \longrightarrow \widehat{I}$ is the character in $\mathcal{SS}$(Act,I) of the product arrow $< \top, \top >: Te \longrightarrow \widehat{I} \times \widehat{I}$.

(ii) Complements:

Given $g : (C, h) \longrightarrow (A, f)$, the complement of g (relative to (A, f)) is the subobject $-g : -(C, h) \longrightarrow (A, f)$ whose character is $\neg \circ \chi_g$. Thus $-g$ is defined to be the pullback of $\top$ along $\neg \circ \chi_g$ yielding $\chi_{-g} = \neg \circ \chi_g$ by definition.

(iii) Intersection:

The intersection of $k : (C, g) \longrightarrow (A, f)$ and $h : (D, l) \longrightarrow (A, f)$ is the subobject $k \cap h : (C, g) \cap (D, l) \longrightarrow (A, f)$ obtained by pulling $\top$ back along $\chi_k \wedge \chi_h = \wedge \circ < \chi_k, \chi_h >$.

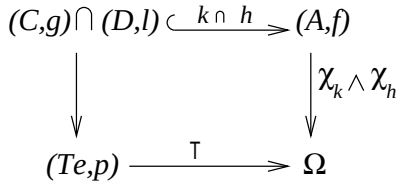


Fig. 6. Intersection of subobjects of (A, f)

Hence $\chi_{k \cap h} = \chi_k \wedge \chi_h$.

Definition 7.1 A topos $\mathcal{E}$ is said to be *boolean* if for each object d, (Sub(d),$\subseteq$) is a boolean algebra.

It is easy to prove that in $\mathcal{SS}$(Act,I) we have $\neg\neg = id$ and it is also easy to prove that the co-product arrow $[\top, \bot]$ is an iso arrow (a condition for a topos to be classical), ie, $Te + Te \cong \widehat{I}$. From an internal point of view it means that $Te + Te$ is the truth-value object of the topos. These results are known to be equivalent to saying that $\mathcal{SS}$(Act,I) is a boolean topos. This means that our topos is a boolean one (although not bivalent), and this is a proof that its internal logic is classical.

8 Conclusions

The motivation of this work was to understand the behaviour of reactive systems using a categorical formulation, more specifically, a sheaf-theoretical formulation. For each S_j of an enumeration of the set of all reactive systems seen as transition systems, we have constructed a semi-sheaf (A_j, f) associated to it. The stalk space A_j is formed by the vertices of the unfolded tree of S_j. The elements of A_j were appropriately labelled taking into account the trace (the sequence of actions performed from the root until reaching the vertex under consideration), the index of the represented state and the index of the system (in that case, j). The base space I is formed by infinite tuples, each one containing the values of the attributes of one specific state in all systems of the enumeration. And finally, the function $f : A_j \longrightarrow I$ was defined mapping one vertex of A_j to the ascending chain of tuples (with values) from the image tuple of such vertex. The concepts of semi-topological base, semi-topology, semi-topological space, s-continuous function, ss-continuous function and semi-local homeomorphism have been introduced to define semi-sheaf. We have specified a semi-topology for A_j based on the actions performed by the system, and we have also specified a topology for I (the Scott topology), and we have shown that $f : A_j \longrightarrow I$ is an ss-continuous function that is a semi-

local homeomorphism. Objects of type (A_j, f) are called semi-sheaves over I associated to transition systems whose set of labels of actions is Act, and form the category $\mathcal{SS}$(Act,I). The morphisms of this category were suitably defined in order to make $\mathcal{SS}$(Act,I) an elementary topos. The construction of the terminal object and of the subobject classifier were also exhibited. Then we analysed the action of the characteristic function over the vertices of Te in order to understand the Ω-axiom and conducted a first study of the internal logic of this topos. The classification made by the subobject classifier is coarser than we expected. Our very first intuition was that we could obtain thinner distinctions by classifing according to how close v_t^{ij} is to A. In other words, we expected to obtain a better notion to measure proximity between behaviours, but it does not seem to be the case. We, then, wanted to answer the question: "What does the logic of $\mathcal{SS}$(Act,I) look like?" Analysing the behaviour of the subobjects of an arbitrary object (A, f), which forms a boolean algebra, we could conclude that the logic governing truth in $\mathcal{SS}$(Act,I) is the (usual) classical logic.

We highlight some potential applications of our work. First, the study of the relation between monomorphisms $(A, f) \hookrightarrow (B, g)$ and the notion of (bi)simulation. We also point out that the intersection of two subobjects of (B, g) seems to reflect the synchronous product of two systems, and we wonder what the union operation might represent.

References

[1] Cattani,G.L. and G.Winskel, "*Presheaf Models for Concurrency*", Proc. of CSL 96, LNCS (1996), 58–75.

[2] Goguen,J.A. "*Sheaf Semantics for Concurrent Interacting Objects*", Mathematical Structures in Computer Science MSCS **(2)2**(1992), 159–191.

[3] Goldblatt,R. "Topoi: The Categorical Analysis of Logic", North-Holland Publishing Company, 1979.

[4] Gray,J.W. "*Fragments of the History of Sheaf Theory*", Applications of Sheaves in Lecture Notes in Mathematics LNM **753**(1977), 1–79.

[5] Joyal,A., M.Nielsen and G.Winskel "*Bisimulation from Open Maps*", LICS (1993), 418–427.

[6] Mac Lane,S. and I. Moerdijk, "Sheaves in Geometry and Logic - A first Introduction to Topos Theory", Springer-Verlag, 1992.

[7] Monteiro, L.F. and F.C.N. Pereira, "*A Sheaf-Theoretic Model of Concurrency*", Proc. of LICS(Symposium on Logic in Computer Science), IEEE Press, 1986, 66–76.

[8] Pnueli,A. "*Linear and Branching Structures in the Semantics and Logics of Reactive Systems*", Proceedings of the 12^{th} ICALP, LNCS **194**(1985),15–32, Springer-Verlag, New York.

[9] Pnueli,A. "*Applications of Temporal Logic to the Specification and Verification of Reactive Systems: a Survey of Current Trends*", in Current Trends in Cuncurrency (G.Goos and J.hartmanis Eds.), LNCS **224**(1986), 510–584.

[10] Scott, D. "*Data Types as Lattices*", SIAM J. Comput. vol **5** n^o 3 (1976), 522–587.

[11] Spinola, A.I.A. "Sistemas Reativos: Uma Abordagem Geométrica", PhD Thesis, Dep. Informática, PUC-Rio, 1999.

[12] Spinola, A.I.A. and E.H. Haeusler, "*A Semi-Sheaf-Theoretic Approach to Reactive Systems*", BEJMC-Brazilian Electronic Journal on Mathematics of Computation,**1**(1999). With associated web site `http://www.bejmc.tche.br`

[13] Stoltenberg-Hansen,V., I.Lindström and E.R.Griffor, "Mathematical Theory of Domains", Cambridge Unversity Press, 1994.

[14] Stoy, J.E. "Denotational Semantics: the Scott-Strachey Approach to Programming Language Theory", MIT Press , 1977.

An Overview of Synchronous Message-Passing and Topology

Maurice Herlihy

Brown University
Providence, RI 02912
herlihy@cs.brown.edu

Sergio Rajsbaum [1]

Compaq Computer Corporation
One Cambridge Center
Cambridge, MA 02142-1612
rajsbaum@crl.dec.com

Mark R. Tuttle

Compaq Computer Corporation
One Cambridge Center
Cambridge, MA 02142-1612
tuttle@crl.dec.com

Abstract

A slowly-growing number of computer scientists have found that ideas from topology can be used to analyze and understand problems in distributed computing. In this paper, we review one approach we have used in the past to write a succinct proof of the lower bound for the number of rounds needed to solve the k-set agreement problem in a synchronous, message-passing model of computation. The central idea in this approach is a simple combinatorial structure we call a *pseudosphere* in which each process from a set of processes is independently assigned a value from a set of values. Pseudospheres have a number of nice combinatorial properties, but their principal interest lies in the observation that the global states that arise in the synchronous, message-passing model can be viewed as simple unions of pseudospheres, and the fact that topological properties of unions of pseudospheres are so easy to prove. We choose this work to review because it is a simple example of how we model distributed systems with topology, and because it is the basis of on-going work to simplify the proof of this result.

This is a preliminary version. The final version can be accessed at
URL: http://www.elsevier.nl/locate/entcs/volume39.html

1 Introduction

Computer scientists have a long tradition of using ideas from topology in their work on problems from semantics and concurrency theory, but only recently have ideas from topology played a role in proving powerful new results in distributed computing. Beginning with a trio of papers independently proving the impossibility of solving the k-set agreement problem in asynchronous systems [BG93,HS99,SZ93], these ideas have been used to study other problems in many other models of computation [AR96,GK99,HR94,HR95,CHLT93,HRT98]. The purpose of this paper is to illustrate how topology is used to model computation in a distributed system, and how ideas from topology can be used to reason about distributed computation. We illustrate these ideas by sketching a recent proof [HRT98] we wrote of a known lower bound [CHLT93] on the number of rounds of communication needed to solve k-set agreement in a synchronous, message-passing model of computation. Our proof is the basis of work in progress to use topology to write proofs of this and other lower bounds that are as succinct as possible, deriving new topological tools for analyzing distributed computation along the way. Let us begin by illustrating why topology is a natural tool for proving the lower bound for k-set agreement, borrowing liberally from introduction to the original proof [CHLT93].

The *k-set agreement problem* [Cha93] is defined as follows. Each processor has a read-only input register and a write-only output register. Each processor begins with an arbitrary input value in its input register from a set V containing at least $k+1$ values $v_0, \ldots, v_k$, and nothing in its output register. A protocol solves k-set agreement if, in every execution, the nonfaulty processors halt after writing output values to their output registers that satisfy two conditions:

(i) *validity*: every processor's output value is some processor's input value, and

(ii) *agreement:* the set of output values chosen must contain at most k distinct values.

The first condition rules out trivial solutions in which a single value is hardwired into the protocol and chosen by all processors in all executions, and the second condition requires that the processors coordinate their choices to some degree. In the special case of $k = 1$, the 1-set agreement is equivalent to the well-known *consensus* problem [LSP82,PSL80,FL82,FLP85,Dol82,Fis83] in when all processors are required to choose the same output value. Consensus is known to be the "hardest" problem in distributed computing, in the sense that all other decision problems can be reduced to it.

We consider the k-set agreement problem in a *synchronous*, *message-passing* model with *crash failures*. In this model, n processors communicate

[1] On leave from Instituto de Matemáticas, U.N.A.M., D.F. 04510, México, rajsbaum@math.unam.mx

by sending messages over a completely connected network. Computation in this model proceeds in a sequence of rounds. In each round, processors send messages to other processors, then receive messages sent to them in the same round, and then perform some local computation and change state. This means that all processors take steps at the same rate, and that all messages take the same amount of time to be delivered. Communication is reliable, but up to f processors can fail by crashing in the middle of a round. When a processor crashes, it sends some subset of the messages it is required to send in that round by the protocol, and then sends no messages in any later round.

The primary contribution of this paper is a (tight) lower bound on the amount of time required to solve k-set agreement. We prove that any protocol solving k-set agreement requires $\lfloor f/k \rfloor + 1$ rounds of communication, where f is the bound on the number of processors allowed to fail in any execution of the protocol. Since consensus is just 1-set agreement, this lower bound implies the famous lower bound of $f + 1$ rounds for solving consensus [FL82]. More important, the running time $r = \lfloor f/k \rfloor + 1$ demonstrates that there is a smooth but inescapable tradeoff among the number f of faults tolerated, the degree k of coordination achieved, and the time r the protocol must run.

Suppose P is a protocol that solves k-set agreement and tolerates the failure of f out of n processors, and suppose P halts in $r < \lfloor f/k \rfloor + 1$ rounds. This means that all nonfaulty processors have chosen an output value at time r in every execution of P. In addition, suppose $n \geq f + k + 1$, which means that at least $k + 1$ processors never fail. Our goal is to consider the *global states* that occur at time r in executions of P, and to show that in one of these states there are $k + 1$ processors that have chosen $k + 1$ distinct values, violating k-set agreement and showing that P could not possibly have solve k-set agreement in only r rounds.

Since consensus is a special case of k-set agreement, it is helpful to review the standard proof of the $f + 1$ round lower bound for consensus [FL82,DS83,Mer85,DM90] to see why new ideas from topology are needed for k-set agreement. Suppose that the protocol P is a consensus protocol, which means that in all executions of P all nonfaulty processors have chosen the same output value at time r. Two global states g_1 and g_2 at time r are said to be *similar* if some nonfaulty processor p has the same local state in both global states. The crucial property of similarity is that the decision value of any processor in one global state completely determines the decision value for any processor in all similar global states. For example, if all processors decide v in g_1, then certainly p decides v in g_1. Since p has the same local state in g_1 and g_2, and since p's decision value is a function of its local state, processor p also decides v in g_2. Since all processors agree with p in g_2, all processors decide v in g_2, and it follows that the decision value in g_1 determines the decision value in g_2. A *similarity chain* is a sequence of global states, $g_1, \ldots, g_\ell$, such that g_i is similar to g_{i+1}. A simple inductive argument shows that the decision value in g_1 determines the decision value in g_ℓ. The

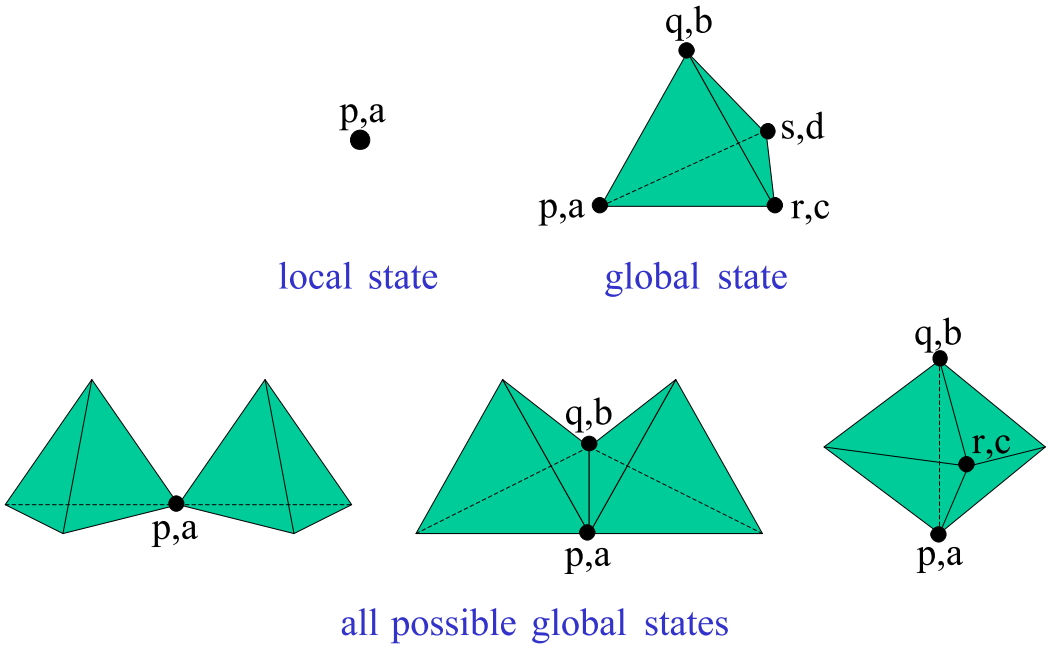


Fig. 1. Modeling global states with a simplicial complex.

lower bound proof involves showing that two time r global states of P, one in which all processors start with 0 and one in which all processors start with 1, lie on a single similarity chain. Since there is a similarity chain from one state to the other, processors must choose the same value in both states, violating the definition of consensus.

The problem with k-set agreement is that the decision values in one global state do not determine the decision values in similar global states. If p has the same local state in g_1 and g_2, then p must choose the same value in both states, but the values chosen by the other processors are not determined. Even if $n-1$ processors have the same local state in g_1 and g_2, the decision value of the last processor is still not determined. The fundamental insight in all proofs of this lower bound [CHLT93,HRT98] is that k-set agreement requires considering all "degrees" of similarity at once—similarity to one processor, to two processors, to three processors—focusing on the number and identity of local states common to two global states. While this seems difficult—if not impossible—to do using conventional graph theoretic techniques like similarity chains, the notions of a simplex and a simplicial complex provides a compact way of capturing all degrees of similarity simultaneously, and are the basis of our proof.

A simplex is just the natural generalization of a triangle to n dimensions: for example, a 0-dimensional simplex is a vertex, a 1-dimensional simplex is an edge linking two vertices, a 2-dimensional simplex is a solid triangle, and a 3-dimensional simplex is a solid tetrahedron. As illustrated in Figure 1, we can represent a local state for one processor p with a single vertex and a global state for four processors p, q, r, and s with a 3-dimensional simplex. We la-

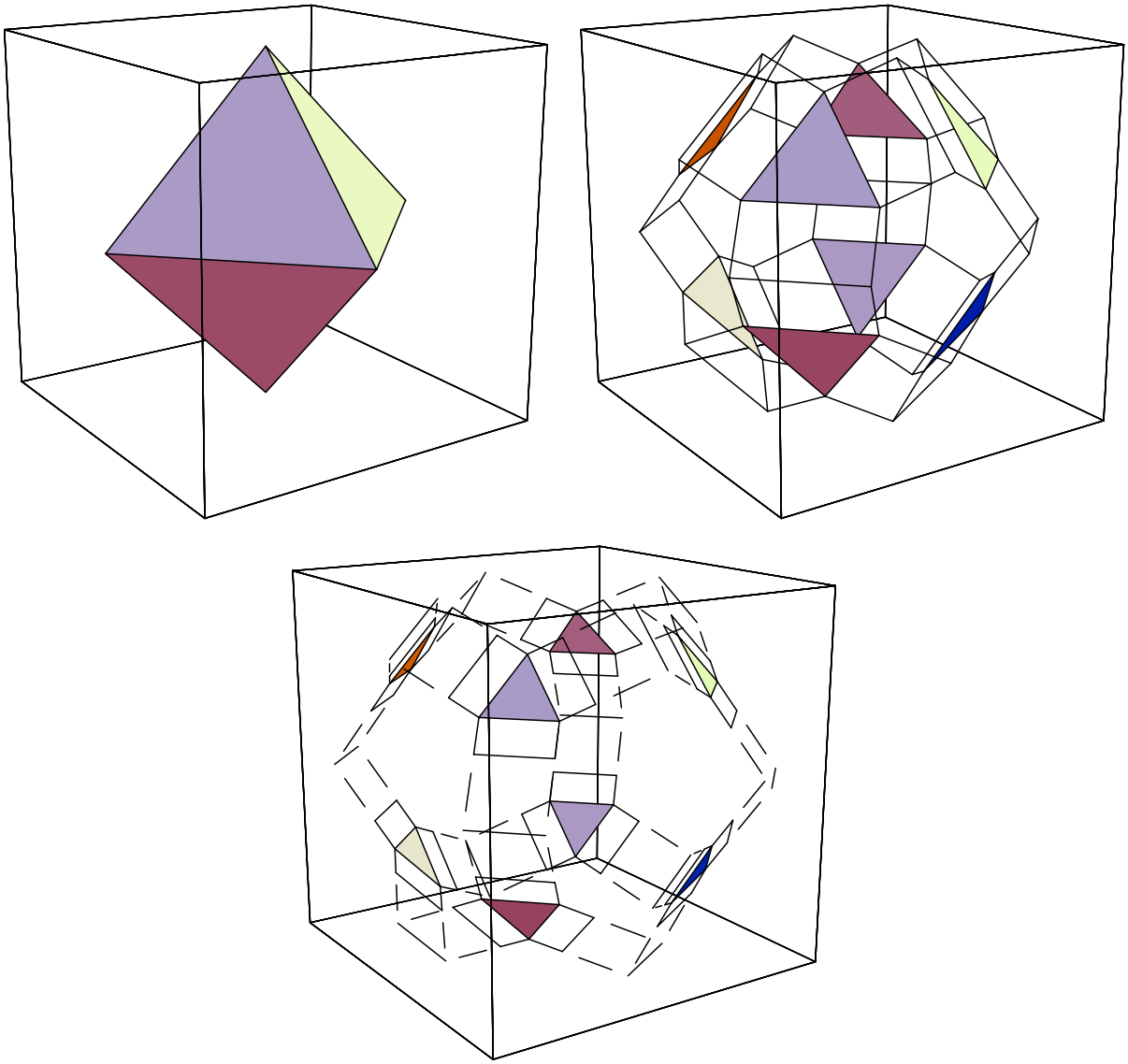


Fig. 2. Global states for zero, one, and two-round protocols.

bel a single vertex representing a processor's local state with the processor's name p and local state a, and we label a 3-dimensional simplex representing a global state for p, q, r, and s by labeling the vertexes corresponding to p, q, r, and s in the same way. Representing all global states as simplexes in this way, the intersection of two simplexes naturally captures the degree of similarity between the two corresponding global states. For example, referring again to Figure 1, two global states similar to p are represented by two simplexes intersecting only in p's vertex, two global states similar to p and q are represented by two simplexes intersecting in the edge between p and q, and two global states similar to p, q, and r are represented by two simplexes intersecting in the entire face containing p, q, and r.

Figure 2 shows the simplicial complexes — called *protocol complexes* —

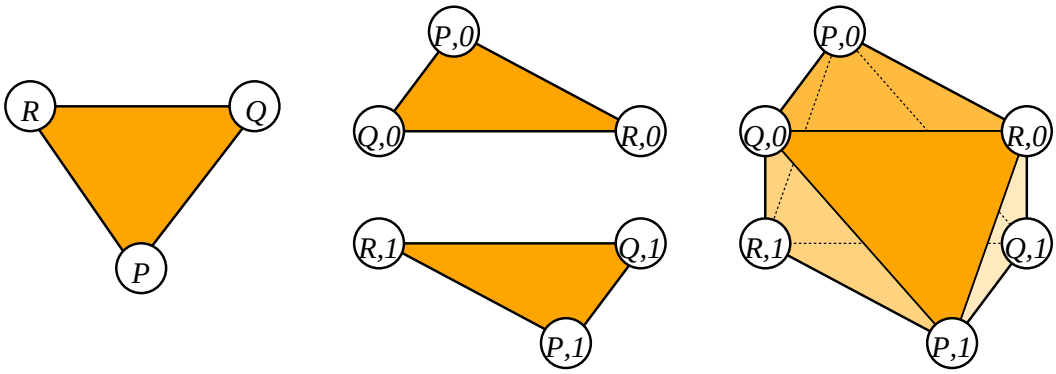


Fig. 3. Construction of a three-process binary pseudosphere.

representing the global states reachable after zero, one, and two rounds of computation in a simple protocol in which each of three processors repeatedly sends its state to the others. Each process begins with a binary input. The first picture shows the possible global states after zero rounds: since no communication has occurred, each processor's state consists only of its input. It is easy to check that the simplexes corresponding to these global states form an octahedron. The next picture shows the complex after one round. Each triangle corresponds to a failure-free execution, each free-standing edge to a single-failure execution, and so on. The third picture shows the possible global states after three rounds.

The connection between these protocol complexes and k-set agreement is the following theorem. Let P be a protocol, and let $\mathcal{C}$ be the simplicial complex representing the set of global states reachable by following P for r rounds of computation. The theorem states that if $\mathcal{C}$ is $(k-1)$-connected, then P cannot solve k-set agreement in r rounds. Proving our lower bound reduces to reasoning about the connectivity of such simplicial complexes.

The key to our proof is the notion of a *pseudosphere*, a simplicial complex in which each process from a set of processes is independently assigned a value from a set of values. Pseudospheres have a number of nice combinatorial properties, but their principal interest lies in the observation that protocol complexes in the synchronous model can be characterized as simple unions of pseudospheres. Because of the simple combinatorial properties of pseudospheres, reasoning about these unions can be accomplished by straightforward combinatorial arguments.

A pseudosphere can be defined very simply, as illustrated in Figure 3. Start with an n-dimensional simplex where each vertex is labeled with a process id, and choose a finite set of values taken from an arbitrary domain. The pseudosphere is the complex constructed by taking multiple copies of this simplex and independently labeling each vertex with a value from the domain. For example, Figure 3 shows how to construct a pseudosphere by independently assigning binary values to a set of three processes. The left-hand figure shows a triangle labeled with process ids P, Q, and R. The central figure shows

an intermediate stage where two copies of the triangle are each labeled with zeros and ones. The right-hand figure shows the complete construction, where copies of the triangle are labeled with all combinations of zeros and ones. We can just as easily assign values from a larger set than $\{0, 1\}$, although the result is harder to illustrate. We call this construct a pseudosphere because it is easily shown that the result of assigning binary values to $n + 1$ processes is topologically equivalent to an n-dimensional sphere.

The collection of initial global states for consensus or k-set agreement clearly forms a pseudosphere whose vertices are labeled with input values. For example, the right-hand figure in Figure 3 is the input complex for three-process consensus. The basic insight underlying the work presented in this paper is that protocol complexes in the synchronous model have natural representations as unions of pseudospheres, except that the vertices are labeled failure information instead of input values. Reasoning about these protocol complexes reduces to the purely combinatorial problem of reasoning about unions of pseudospheres. We express the one-round executions as the union of pseudospheres. An r-round execution is constructed by inductively replacing each simplex in the single-round execution with the union of pseudospheres produced by the $(r - 1)$-round protocol. The protocol complex produced by this iterative construction represents only a subset of the global states reachable in the model, but this set is large enough to prove the desired results for consensus, k-set agreement, renaming, and so on.

2 Basic Topology

A *vertex* $\vec{v}$ is a point in a high-dimensional Euclidian space. Vertexes $\vec{v}_0, \ldots, \vec{v}_n$ are *affinely independent* if $\vec{v}_1 - \vec{v}_0, \ldots, \vec{v}_n - \vec{v}_0$ are linearly independent. An *n-dimensional simplex* (or *n-simplex*) $S^n = (\vec{s}_0, \ldots, \vec{s}_n)$ is the convex hull of a set of $n + 1$ affinely-independent vertexes. For example, a 0-simplex is a vertex, a 1-simplex a line segment, a 2-simplex a solid triangle, and a 3-simplex a solid tetrahedron. Where convenient, we use superscripts to indicate dimensions of simplexes. We say that the $\vec{s}_0, \ldots, \vec{s}_n$ *span* S^n. By convention, a simplex of dimension $d < 0$ is an empty simplex. Simplex S^m is a (proper) *face* of T^n if the vertexes of S^m are a (proper) subset of the vertexes of T.

A *simplicial complex* (or complex) is a set of simplexes closed under containment and intersection. The *dimension* of a complex is the highest dimension of any of its simplexes. In this paper all the complexes of dimension n are *full* in the sense that every simplex is contained in some n-simplex. $\mathcal{L}$ is a *subcomplex* of $\mathcal{K}$ if every simplex of $\mathcal{L}$ is a simplex of $\mathcal{K}$. The *m-skeleton* of $\mathcal{K}$, denoted $skel^m(\mathcal{K})$, is the subcomplex consisting of all simplexes of $\mathcal{K}$ of dimension at most m. A map $\mu : \mathcal{K} \to \mathcal{L}$ carrying vertexes to vertexes is *simplicial* if it also carries simplexes to simplexes. Two complexes $\mathcal{K}$ and $\mathcal{L}$ are *isomorphic*, written $\mathcal{K} \cong \mathcal{L}$, if there is a surjective and one-to-one simplicial map $\iota : \mathcal{K} \to \mathcal{L}$.

Informally, a complex is k-connected if it has no holes in dimensions k or less. More precisely,

Definition 2.1 *A complex $\mathcal{K}$ is* k-connected *if every continuous map of the k-sphere to $\mathcal{K}$ can be extended to a continuous map of the $(k+1)$-disk [Spa66, p. 51]. (By convention, a complex is* (-1)-connected *if it is nonempty, and every complex is* k-connected *for $k < -1$.)*

This definition says that a complex is 0-connected if it is connected in the graph-theoretic sense. The following theorem is an elementary consequence of the Mayer-Vietoris sequence [Mun84, p. 142]. It allows us to reason about a complex's connectivity in terms of the connectivity of its components.

Theorem 2.2 *If $\mathcal{K}$ and $\mathcal{L}$ are complexes such that $\mathcal{K}$ and $\mathcal{L}$ are k-connected, and $\mathcal{K} \cap \mathcal{L}$ is nonempty and $(k-1)$-connected, then $\mathcal{K} \cup \mathcal{L}$ is k-connected.*

3 Model

A set of $n+1$ sequential *processes* communicate by sending messages to one another. At any point, a process may *crash*: it stops and sends no more messages. There is a bound f on the number of processes that can fail. In the *synchronous* model, processes take steps at the same rate, and messages take the same amount of time to be delivered, and message delivery is reliable and FIFO.

Each process starts with an *input value* taken from a set V, and then executes a deterministic *protocol* in which it repeatedly receives one or more messages, changes its local state, and sends one or more messages. After a finite number of steps, each process chooses a *decision value* and halts. At any instant, a process's local state is given by its *view*: the input value and the the sequence of messages received so far. A protocol is uniquely determined by its *message function* and its *decision function*. The message function determines which messages a process should send in a given state, and the decision function determines which output value a process should choose in a given state (if any). A protocol is a *full-information protocol* [Had83,FL82,PSL80] if the message function causes each process to send its entire local state when it sends a message. We can assume without loss of generality that all protocols $\mathcal{P}$ we consider are *full-information* protocols [Had83,FL82,PSL80,DM90].

In the k-set agreement task [Cha91], processes are required to (1) choose a decision value after a finite number of steps, (2) choose as decision value some process's input value, and (3) collectively choose no more than k distinct decision values. When $k = 1$, this problem is usually called *consensus*.

We now show how to apply concepts from combinatorial topology to this model. An initial local state of process P is modeled as a vertex $\vec{v} = \langle P, v \rangle$ labeled with P's process id and initial value v. An initial global state is modeled as an n-simplex $S^n = (\langle P_0, v_0 \rangle, \ldots, \langle P_n, v_n \rangle)$, where the P_i are distinct. We use $ids(S^n)$ to denote the set of process ids associated with S^n, and $vals(S^n)$

the set of values. The set of all possible initial global states forms a complex, called the *input complex*.

Any protocol has an associated *protocol complex* $\mathcal{P}$, defined as follows. Each vertex is labeled with a process id and a possible view for that process. A set of vertexes $\langle P_{i_0}, v_{i_0}\rangle, \ldots, \langle P_{i_d}, v_{i_d}\rangle$ spans a simplex of $\mathcal{P}$ if and only if there is some protocol execution in which $P_{i_0}, \ldots, P_{i_d}$ finish the protocol with respective views $v_{i_0}, \ldots, v_{i_d}$. Each simplex thus corresponds to an equivalence class of executions that "look the same" to the processes at its vertexes. The protocol complex $\mathcal{P}$ depends both on the protocol and on the timing and failure characteristics of the model.

We use $\mathcal{P}(S^m)$ to denote the subcomplex of $\mathcal{P}$ corresponding to executions in which only the processes in $ids(S^m)$ participate (the rest fail before sending any messages). If $m < n - f$, then there are no such executions, and $\mathcal{P}(S^m)$ is empty. More generally, if $\mathcal{I}$ is a subcomplex of the input complex, then we define $\mathcal{P}(\mathcal{I})$ to be the union of $\mathcal{P}(S^m)$ for all S^m in $\mathcal{I}$. A protocol *solves* k-set agreement if the protocol's decision map δ carries vertexes of $\mathcal{P}$ to values in V such that if $\vec{p} \in \mathcal{P}(S^n)$, then $\delta(\vec{p}) \in vals(S^n)$.

4 Pseudospheres

Informally, a pseudosphere is a combinatorial structure in which each process from a set of processes is independently assigned a value from a set of values.

Definition 4.1 *Let $S^m = (\vec{s}_0, \ldots, \vec{s}_m)$ be a simplex and $U_0, \ldots, U_m$ be a sequence of finite sets. The* pseudosphere *$\psi(S^m; U_0, \ldots, U_m)$ is the following complex. Each vertex is a pair $\langle \vec{s}_i, u_i\rangle$, where $\vec{s}_i$ is a vertex of S^m and $u_i \in U_i$. Vertexes $\langle \vec{s}_{i_0}, u_{i_0}\rangle, \ldots, \langle \vec{s}_{i_\ell}, u_{i_\ell}\rangle$ span a simplex of $\psi(S^m; U_0, \ldots, U_m)$ if and only if the $\vec{s}_i$ are distinct. A pseudosphere in which all U_i equal U is simply written $\psi(S^m; U)$.*

We call this construct a pseudosphere because if S^n is an n-dimensional simplex, then $\psi(S^n; \{0, 1\})$ is homeomorphic to an n-dimensional sphere. Pseudospheres are important because every complex considered here is either a pseudosphere or the union of pseudospheres. Because any process can start with any input from V, the input complex to k-set agreement is the pseudosphere $\psi(P^n; V)$, where P^n is a simplex whose vertexes are labeled with the $n + 1$ distinct process ids.

Lemma 4.2 *Pseudospheres satisfy the following simple combinatorial properties.*

(i) *If U is a singleton set, then $\psi(S^m, U) \cong S^m$.*

(ii) *Let $S^m = (\vec{s}_0, \ldots, \vec{s}_m)$, and $S^{m-1} = (\vec{s}_0, \ldots, \widehat{\vec{s}_i}, \ldots \vec{s}_m)$, where circumflex denotes omission. If $U_i = \emptyset$, then*

$$\psi(S^m; U_0, \ldots, U_m) \cong \psi(S^{m-1}; U_0, \ldots, \widehat{U}_i, \ldots, U_m).$$

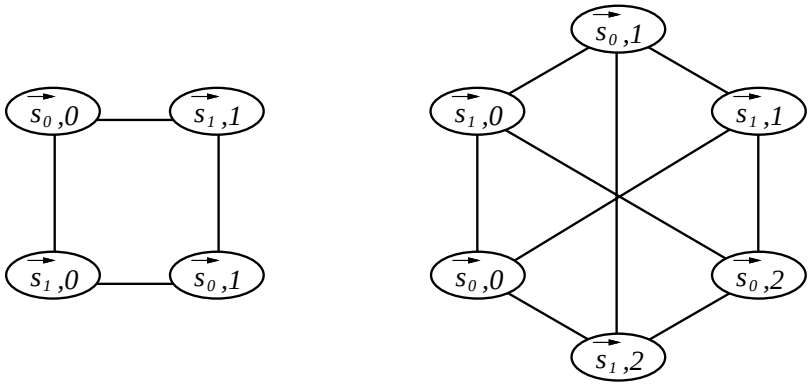


Fig. 4. Pseudospheres $\psi(\{P_0, P_1\}; \{0, 1\})$ and $\psi(\{P_0, P_1\}; \{0, 1, 2\})$.

(iii) $\psi(S_0; U_0, \ldots, U_m) \cap \psi(S_1; V_0, \ldots, V_m) \cong \psi(S_0 \cap S_1; U_0 \cap V_0, \ldots, U_m \cap V_m)$.

The next theorem shows how to exploit the nice combinatorial properties of pseudospheres. It states that if applying a protocol to a single simplex preserves connectivity below some dimension, then applying that protocol to any input pseudosphere also preserves that degree of connectivity. It is actually a theorem in topology, and so it applies to any model of computation.

Theorem 4.3 *Let $\mathcal{P}$ be a protocol, S^m be a simplex, and c be a constant. If for every face S^ℓ of S^m and for every sequence $V_0, \ldots, V_\ell$ of singleton sets the protocol complex $\mathcal{P}(\psi(S^\ell; V_0, \ldots, V_\ell))$ is $(\ell - c - 1)$-connected, then for every sequence $U_0, \ldots, U_m$ of nonempty sets the protocol complex $\mathcal{P}(\psi(S^m; U_0, \ldots, U_m))$ is $(m - c - 1)$-connected.*

A consequence of this theorem is that any n-dimensional pseudosphere is $(n-1)$-connected (just let $\mathcal{P}$ be the trivial protocol in which each process halts immediately):

Corollary 4.4 *If $U_0, \ldots, U_m$ are all nonempty, then $\psi(S^m; U_0, \ldots, U_m)$ is $(m-1)$-connected.*

Naively, one might think that S^m is always m-connected, but note that although the empty simplex has dimension -1, it is not (-1)-connected. We can generalize Theorem 4.3 to multiple pseudospheres.

Theorem 4.5 *Let $\mathcal{P}$ be a protocol satisfying the precondition of Theorem 4.3, and let $A_0, \ldots, A_\ell$ be a sequence of finite sets. If $\cap_{i=0}^{\ell} A_i \neq \emptyset$ then*

$$\mathcal{P}\left(\bigcup_{i=0}^{\ell} \psi(S^m; A_i)\right) \text{ is } (m - c - 1)\text{-connected.}$$

Letting $\mathcal{P}$ be the trivial protocol in which each process decides its input:

Corollary 4.6 *If $A_0, \ldots, A_\ell$ is a sequence of finite sets such that $\cap_{i=0}^{\ell} A_i \neq \emptyset$*

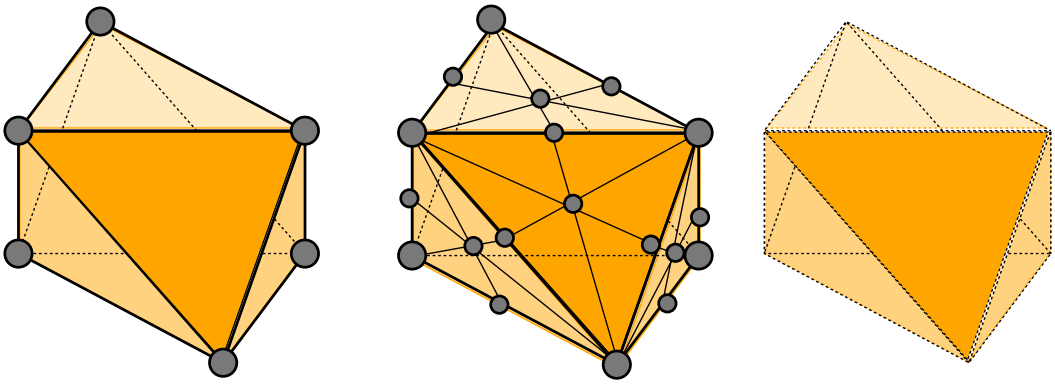


Fig. 5. Simplicial complex, subdivision, and polyhedron

then

$$\bigcup_{i=0}^{\ell} \psi(S^m; A_i) \text{ is } (m-1)\text{-connected.}$$

5 Connectivity vs k-Set Agreement

The notion of k-connectivity lies at the heart of all known lower bounds for k-set agreement. In this section, we prove a general theorem linking $(k-1)$-connectivity with impossibility of k-set agreement. This theorem is model independent in the sense that it depends on the connectivity properties of protocol complexes, not on explicit timing or failure properties of the model. This result was originally stated elsewhere [HR94], but for the sake of making this paper self-contained, we present the full proof here.

The point-set occupied by a complex $\mathcal{C}$ is called its *polyhedron*, and is denoted by $|\mathcal{C}|$. Any simplicial map $\phi : \mathcal{A} \to \mathcal{B}$ induces a piece-wise linear map $|\phi| : |\mathcal{A}| \to |\mathcal{B}|$ that agrees with ϕ on vertexes of $\mathcal{A}$.

A *subdivision* of a complex $\mathcal{A}$ is a complex $\mathcal{B}$ such that (1) each simplex of $\mathcal{B}$ is contained in a simplex of $\mathcal{A}$, and (2) each simplex of $\mathcal{A}$ is the union of finitely many simplexes of $\mathcal{B}$ [Mun84, p. 83]. This definition implies that $|\mathcal{A}| = |\mathcal{B}|$. If $\vec{b}$ is a vertex of $\mathcal{B}$, the *carrier* of $\vec{b}$ in $\mathcal{A}$, denoted $carrier(\vec{b}, \mathcal{A})$, is the smallest simplex of $\mathcal{A}$ that contains $\vec{b}$. Figure 5 illustrates a complex, a subdivision of that complex, and their underlying polyhedron.

We will need a step-by-step method for constructing subdivisions. Let $\mathcal{C}$ be a complex, and $\vec{w}$ a point with the property that any ray emanating from $\vec{w}$ intersects $|\mathcal{C}|$ in at most one point. Define the *cone* $\vec{w} \cdot \mathcal{C}$ to be the collection of all simplexes of the form $(\vec{w}, \vec{s}_0, \ldots, \vec{s}_k)$, where $(\vec{s}_0, \ldots, \vec{s}_k)$ is a simplex of $\mathcal{C}$, together with all faces of such simplexes. This cone is itself a complex, having $\mathcal{C}$ as a subcomplex [Mun84, p. 44]. Let σ be a subdivision of $skel^{\ell-1}(\mathcal{C})$, and $S_0^\ell, \ldots, S_L^\ell$ the ℓ-simplexes of $skel^\ell(\mathcal{C})$. For $0 \le i \le L$, let $\vec{w}_i$ be an interior point of $|S_i^\ell|$. Each cone $\vec{w}_i \cdot \sigma(S_i^\ell)$ is a subdivision of S_i^ℓ, and the union of these cones as i ranges from 0 to L is a subdivision of $skel^\ell(\mathcal{C})$ that agrees with σ on the $(\ell-1)$ skeleton [Mun84, p. 85]. The result is called the subdivision

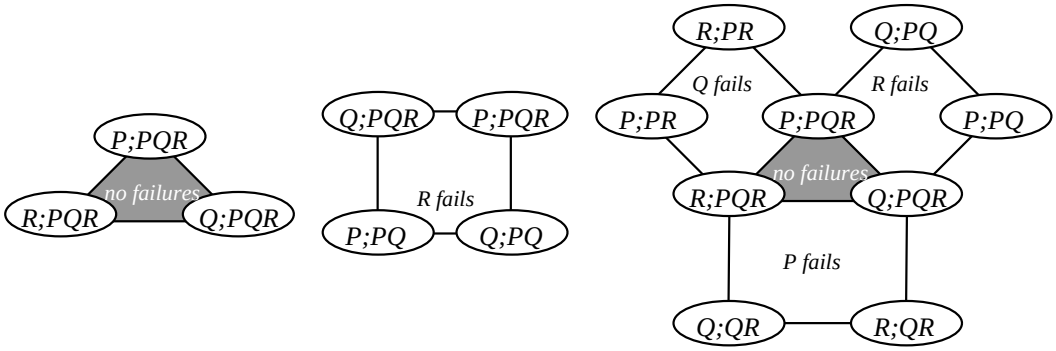


Fig. 6. Construction of a one-round three-process protocol complex.

of $skel^{\ell}(\mathcal{C})$ obtained by *starring* σ. The subdivision shown in Figure 5 is the result of successive starring.

We use the following variant of Sperner's Lemma [Lef49, Lemma 5.5]:

Lemma 5.1 (Sperner's Lemma) *Let $\sigma(S^n)$ be a subdivision of simplex S^n. If $F : \sigma(S^n) \to S^n$ is a map sending each vertex of $\sigma(S^n)$ to a vertex in its carrier, then there is at least one n-simplex $T^n = (\vec{t}_0, \ldots, \vec{t}_n)$ in $\sigma(S^n)$ such that the $F(\vec{t}_i)$ are all distinct.*

We also exploit the following extension lemma, which appears in Glaser [Gla70, Theorem IV.2].

Lemma 5.2 *Let $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ be complexes such that $\mathcal{A} \subset \mathcal{B}$, and $f : |\mathcal{B}| \to |\mathcal{C}|$ is a continuous map such that f restricted to $|\mathcal{A}|$ is simplicial. There exists a subdivision τ of $\mathcal{B}$ such that $\tau(\mathcal{A}) = \mathcal{A}$, and a simplicial map $\phi : \tau(\mathcal{B}) \to \mathcal{C}$ extending the restriction of f to $|\mathcal{A}|$.*

Theorem 5.3 *Let $V = \{v_0, \ldots, v_k\}$ be a set of $k+1$ possible input values, and $\mathcal{P}$ a protocol with input complex $\psi(P_0, \ldots, P_n; V)$. If $\mathcal{P}$ has the property that for every n-dimensional pseudosphere $\psi(P_0, \ldots, P_n; U)$, where U is a nonempty subset of V, $\mathcal{P}(\psi(P_0, \ldots, P_n; U)$ is $(k-1)$-connected, then $\mathcal{P}$ cannot solve k-set agreement.*

Theorems 4.3 and 5.3 imply

Corollary 5.4 *If $\mathcal{P}(\mathcal{S}^m)$ is $(m-(n-k)-1)$-connected for all m where with $n - f \leq m \leq n$, then $\mathcal{P}$ cannot solve k-set agreement in the presence of f failures.*

6 Synchronous Computation

We now define the r-round synchronous protocol complex $\mathcal{S}^r(S^m)$. Here too, we consider only a subset of all possible executions: executions in which no more than k processes fail in any round. We are interested in executions where no more than k processes fail in any round. Informally, we will show that the one-round protocol complex is the union of pseudospheres, where each pseudosphere corresponds to the set of executions in which a fixed set

of processes fail. For example, Figure 6 illustrates the possible executions of a one-round protocol for three processes, P, Q, and R, starting from a fixed input simplex, in which no more than one process fails. Here, each vertex is labeled with a process, followed by the processes from which it has received messages. The figure on the left represents the execution in which in which no processes fail: this is a (degenerate) pseudosphere in which each process receives the same set of messages. The figure in the middle represents the executions in which R alone fails. This complex is a pseudosphere: P and Q independently do or do not receive a message from R. The figure on the right represents the entire one-faulty protocol complex. It is the union of the failure-free pseudosphere with the three single-failure pseudospheres.

6.1 Single-Round Protocols

Let $\mathcal{S}^1(S^n)$ be the complex of one-round executions of an $(n+1)$-process protocol with input simplex S^n in which at most k processes fail. It is the union of complexes $\mathcal{S}^1_K(S^n)$ of one-round executions starting from S^n in which *exactly* the processes in K fail. Given a set K of process ids, let $S^n \backslash K$ be the face of S^n labeled with the process ids *not* in K. Our next result says that $\mathcal{S}^1_K(S^n)$ is a pseudosphere, which means that $\mathcal{S}^1(S^n)$ is a union of pseudospheres:

Lemma 6.1 *If $m \geq n - f$ and K is a subset of $ids(S^m)$ of size at most $f - (n - m)$, then*

$$\mathcal{S}^1_K(S^m) \cong \psi(S^m \backslash K; 2^K).$$

The one-round complex is a union of pseudospheres in the synchronous model (Lemma 6.1). To compute the connectivity of this union using Theorem 2.2, we need to understand the intersections. The next lemma shows that these intersections have a simple structure: they are themselves the union of pseudospheres. Order the process sets lexicographically: the empty set first, followed by singleton sets, followed by two-element sets, and so on. Let $K_0, \ldots, K_\ell$ be the sequence of sets of process ids less than or equal to K_ℓ, listed in lexicographic order.

Lemma 6.2 *Let $m \geq n - f$ and let $K_0, \ldots, K_k$ be the subsets of $ids(S^m)$ of size at most $f - (n - m)$ arranged in lexicographical order. If $K_0, \ldots, K_\ell$ is a prefix of this sequence, then*

$$\bigcup_{i=0}^{\ell-1} \mathcal{S}^1_{K_i}(S^m) \cap \mathcal{S}^1_{K_\ell}(S^m) = \bigcup_{p \in K_\ell} \psi(S^m \backslash K_\ell; 2^{K_\ell - \{p\}}).$$

Let $\mathcal{S}^1(S^n)$ denote the protocol complex for a one-round synchronous $(n+1)$-process protocol with input simplex S^n where no more than than k processes fail.

Lemma 6.3 *$\mathcal{S}^1(S^m)$ is $(m - (n - k) - 1)$-connected if $m \geq (n - f) + k$ and $n \geq 2k$.*

6.2 Multi-Round Protocols

Let $\mathcal{S}^r(S^n)$ be the protocol complex for an r-round synchronous $(n+1)$-process protocol with input simplex S^n where no more than than k processes fail in each round. We can decompose this complex as follows. Let $K_0, \ldots, K_\ell$ be a sequence of sets of k or fewer process ids in lexicographic order. Recall that $\mathcal{S}^1_{K_i}(S^n) = \psi(S^n \backslash K_i; 2^{K_i})$ is the complex of one-round executions in which exactly the processes in K_i fail. The set of r-round executions in which exactly the processes in K_i fail in the first round can be written as $\mathcal{S}^{r-1}_i(\mathcal{S}^1_{K_i}(S^n))$, where $\mathcal{S}^{r-1}_i$ is the complex for an $(r-1)$-round, $(f-|K_i|)$-faulty, $(n-|K_i|+1)$-process full-information protocol. The $\mathcal{S}^{r-1}_i$ are considered distinct protocols because the $\mathcal{S}^1_{K_i}(S^n)$ have varying dimensions. Taking the union over all the K_i, we have

$$\mathcal{S}^r(S^n) = \bigcup_{i=0}^{\ell} \mathcal{S}^{r-1}_i(\mathcal{S}^1_{K_i}(S^n)).$$

The connectivity of a protocol $\mathcal{P}$ depends on the *degree* of the protocol. Consider the multi-round executions of $\mathcal{P}$ in which f_i is the maximum number of processes that fail at round i. The *degree* of $\mathcal{P}$ is the minimum f_i for any round. Define $\widetilde{\mathcal{S}}^{r-1}_\ell$ to be the protocol identical to $\mathcal{S}^{r-1}_\ell$ except that it fails at most $k-1$ processes in its first round. While $\mathcal{S}^{r-1}_\ell$ has degree k, $\widetilde{\mathcal{S}}^{r-1}_\ell$ has degree $k-1$. Our next result implies that intersections of the complexes comprising $\mathcal{S}^r$ are equivalent to $\widetilde{\mathcal{S}}^{r-1}_\ell$ applied to a union of pseudospheres, which makes it possible to use Theorem 2.2 to analyze the connectivity of $\mathcal{S}^r$.

Lemma 6.4

$$\bigcup_{i=0}^{\ell-1} \mathcal{S}^{r-1}_i(\mathcal{S}^1_{K_i}(S^n)) \cap \mathcal{S}^{r-1}_\ell(\mathcal{S}^1_{K_\ell}(S^n)) = \widetilde{\mathcal{S}}^{r-1}_\ell \left(\bigcup_{j \in K_\ell} \psi(S^n \backslash K_\ell; 2^{K_\ell - \{j\}}) \right).$$

Define

$$\mathcal{S}^P_K(S^m) = \begin{cases} \mathcal{T} \text{ if } \mathit{ids}(S^m) \subseteq P \text{ and } P - \mathit{ids}(S^m) \subseteq K \\ \emptyset \text{ otherwise} \end{cases}$$

where $\mathcal{T}$ is the complex of one-round executions of the full-information protocol in which only the processes in K fail, starting with the processes in P and the input simplex S^m. The condition $\mathit{ids}(S^m) \subseteq P$ says that initial inputs are provided for some of the processes, and the condition $P - \mathit{ids}(S^m) \subseteq K$ says that processes for which no input is provided can be considered to have failed immediately before having sent a single message. In general, define

$$\mathcal{S}^P_{K_r, K_{r-1}, \ldots, K_1}(S^m) = \mathcal{S}^{P_r}_{K_r} \mathcal{S}^{P_{r-1}}_{K_{r-1}} \cdots \mathcal{S}^{P_1}_{K_1}(S^m) \qquad \text{where } P_i = P - \bigcup_{j=1}^{i-1} K_j.$$

A consequence of these definitions is that the complex $\mathcal{S}^P_{K_r,K_{r-1},\ldots,K_1}(S^m)$ is the empty set unless $ids(S^m) \subseteq P$ and $P - ids(S^m) \subseteq K_1$.

Define a *failure pattern* to be a sequence $\sigma = \sigma_r, \ldots, \sigma_1$ of integers representing upper bounds on the number of processes allowed to fail in each of the first r rounds of the full-information protocol. The failure pattern of most interest to us will be the failure pattern in which k processes fail in each round. We say that the failure pattern σ has *degree* k if it is a nondecreasing sequence of integers

$$k = \sigma_1 \leq \sigma_2 \leq \cdots \leq \sigma_r$$

beginning with k. We say that a sequence $\Sigma = K_r, \ldots, K_1$ of process sets *satisfies* σ if $|K_i| \leq \sigma_i$ for each i, and we write $\Sigma \sim \sigma$. Generalizing $\mathcal{S}^P_{K_r,K_{r-1},\ldots,K_1}(S^m)$, we define

$$\mathcal{S}^P_\sigma(S^m) = \bigcup_{\Sigma \sim \sigma} \mathcal{S}^P_\Sigma(S^m)$$

to be the complex of r-round executions of the full-information protocol with failure pattern σ, starting with the processes in P and the input simplex S^m. We say that $\mathcal{S}^P_\sigma(S^m)$ has degree k if σ has degree k.

Lemma 6.5 *Let $\sigma = (k_r, k_{r-1}, \ldots, k_1)$ be a failure pattern and P be a set of processors. Let $\tau = (k_r, k_{r-1}, \ldots, k_2)$ and $\tau' = (k_r, k_{r-1}, \ldots, k_2 - 1)$, and let $K_1, \ldots, K_k$ be the subsets of P of size at most k_1 listed in lexicographical order.*

$$\bigcup_{i=0}^{\ell-1} \mathcal{S}^{P-K_i}_\tau(\mathcal{S}^P_{K_i}(S^m)) \cap \mathcal{S}^{P-K_\ell}_\tau(\mathcal{S}^P_{K_\ell}(S^m))$$

$$= \mathcal{S}^{P-K_\ell}_{\tau'}\left(\bigcup_{p \in K_\ell} \psi(S^m \backslash K_\ell; 2^{K_\ell - \{p\}})\right).$$

Lemma 6.6 *Let $\sigma = (k_r, \ldots, k_1)$ be a failure pattern of degree k, and let P be a set of processors of size n. If $n \geq k_r + \cdots + k_1 + k$ and $ids(S^m) \subseteq P$, then $\mathcal{S}^P_\sigma(S^m)$ is $(m - (n-k) - 1)$-connected.*

The connectivity of this protocol complex implies the lower bound for synchronous k-set agreement [CHLT93]:

Theorem 6.7 *If $n \geq f + k$, then any synchronous f-resilient k-set agreement protocol requires $\lfloor f/k \rfloor + 1$ rounds. If $n < f + k$, then any synchronous f-resilient k-set agreement protocol requires $\lfloor f/k \rfloor$ rounds.*

Proof. If $n - k \geq f$, then $\mathcal{S}^{\lfloor f/k \rfloor}(\mathcal{I})$ is $(k-1)$-connected. If $n - k < f$, then $\mathcal{S}^{\lfloor f/k \rfloor - 1}(\mathcal{I})$ is $(k-1)$-connected. Either way, Theorem 5.3 states that the protocol cannot solve k-set agreement.

References

[AR96] Hagit Attiya and Sergio Rajsbaum. The combinatorial structure of wait-free solvable tasks. In *Proceedings of the 10th International Workshop*

on Distributed Algorithms, volume 1151 of *Lecture Notes in Computer Science*, pages 322–343. Springer-Verlag, Berlin, October 1996.

[BG93] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 91–100, May 1993.

[Cha91] Soma Chaudhuri. Towards a complexity hierarchy of wait-free concurrent objects. In *Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing*, December 1991.

[Cha93] Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, July 1993.

[CHLT93] Soma Chaudhuri, Maurice Herlihy, Nancy Lynch, and Mark R. Tuttle. A tight lower bound for k-set agreement. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, pages 206–215, November 1993.

[DM90] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a Byzantine environment: Crash failures. *Information and Computation*, 88(2):156–186, October 1990.

[Dol82] Danny Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, March 1982.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(3):656–666, November 1983.

[Fis83] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In Marek Karpinsky, editor, *Proceedings of the 10th International Colloquium on Automata, Languages, and Programming*, pages 127–140. Springer-Verlag, 1983.

[FL82] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, June 1982.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM*, 32(2):374–382, April 1985.

[GK99] Eli Gafni and Elias Koutsoupias. Three-processor tasks are undecidable. *SIAM Journal on Computing*, 28(3):970–983, 1999.

[Gla70] L. C. Glaser. *Geometrical Combinatorial Topology*, volume 1. Van Nostrand Reinhold, New York, 1970.

[Had83] Vassos Hadzilacos. A lower bound for Byzantine agreement with fail-stop processors. Technical Report TR–21–83, Harvard University, 1983.

[HR94] Maurice Herlihy and Sergio Rajsbaum. Set consensus using arbitrary objects. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, pages 324–333, August 1994.

[HR95] Maurice Herlihy and Sergio Rajsbaum. Algebraic spans. In *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing*, pages 90–99, August 1995. *Mathematical Structures in Computer Science*, to appear.

[HRT98] Maurice Herlihy, Sergio Rajsbaum, and Mark R. Tuttle. Unifying synchronous and asynchronous message-passing models. In *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing*, pages 133–142. ACM, June 1998.

[HS99] Maurice P. Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, November 1999.

[Lef49] S. Lefschetz. *Introduction to Topology.* Princeton University Press, Princeton, New Jersey, 1949.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[Mer85] Michael Merritt. Notes on the Dolev-Strong lower bound for byzantine agreement. Unpublished manuscript, 1985.

[Mun84] J. R. Munkres. *Elements Of Algebraic Topology.* Addison Wesley, Reading MA, 1984.

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[Spa66] Edwin H. Spanier. *Algebraic Topology.* Springer-Verlag, New York, 1966.

[SZ93] Michael Saks and Fotis Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 101–110, May 1993. *SIAM Journal on Computing*, to appear.

From concurrency to algebraic topology

Philippe Gaucher

Institut de Recherche Mathématique Avancée, ULP et CNRS, 7 rue René Descartes, 67084 Strasbourg Cedex, France, gaucher@math.u-strasbg.fr

Abstract

This paper is a survey of the new notions and results scattered in [13], [11] and [12]. Starting from a formalization of higher dimensional automata (HDA) by strict globular ω-categories, the construction of a diagram of simplicial sets over the three-object small category $- \leftarrow gl \rightarrow +$ is exposed. Some of the properties discovered so far on the corresponding simplicial homology theories are explained, in particular their links with geometric problems coming from concurrency theory in computer science.

1 Introduction

We have already argued in [13] for modeling higher dimensional automata (HDA) using strict globular ω-categories. To our knowledge, the link between globular ω-categories and concurrent automata was first noticed in [21]. Papers [13] [11] and [12] show that this way of formalizing HDA is very well adapted to getting interesting new functors deeply related to the computer-scientific properties of the HDA. We would like to explain here the construction of these functors, some of their known properties and some perspectives. Many explanations are given in a very informal way. We refer to the bibliography for more details.

In [21] and [14], HDA are formalized using cubical sets in the sense of [6]. The link between cubical sets and ω-categories will be described at the end of Section 3. There are two equivalent approaches of the notion of ω-category : a cubical one and a globular one [3]. The globular approach will be used everywhere in this paper except in Figure 6 where cubical subdivisions are depicted. An example of globular subdivision is depicted in Figure 3(a). An informal topological description of HDA is also used in Section 2. The link between all these descriptions of HDA is summarized from the point of view of the homotopy of HDA in Section 6.

This survey is intended to be readable by non-specialists in algebraic topology. Only a small background is required : the definition of simplicial set (their face maps will be denoted by ∂_i and their degeneracy maps by ϵ_i) and of the as-

This is a preliminary version. The final version can be accessed at
URL: http://www.elsevier.nl/locate/entcs/volume39.html

sociated simplicial homology H_*, and therefore the definition of the homology of a chain complex of abelian groups [20] [27] [23].

Some geometric intuitions are introduced in Section 2. The definition of globular ω-category is recalled in Section 3. This section also provides a description of the ω-categories associated to the n-cubes and to the n-simplexes for all n. The three nerves and the two morphisms h^- and h^+ are described in Section 4. In Section 5, we speculate about what should be a good invariant of HDA.

2 The topology of HDA in an informal way

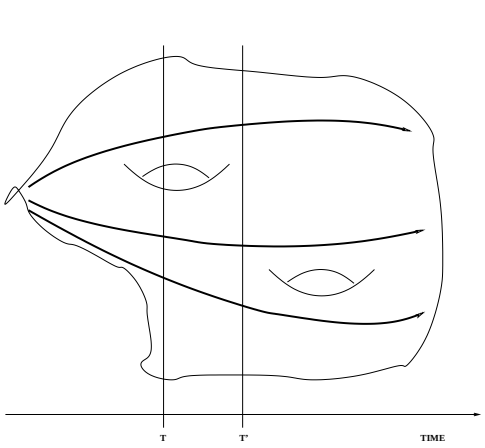
To explain the geometric intuition which underlies the constructions of this paper, let us make a digression by considering a continuous model instead of a discrete one. The main ideas are indeed much simpler to understand in a continuous setting for a reader not familiar with the ω-categorical style.

HDA can be seen as flows of execution paths on a topological space X as in Figure 1(a). The points of the topological space correspond to the states of the HDA. The closest formalism is that of locally partially ordered topological spaces developed in [10] for an algorithmic purpose.

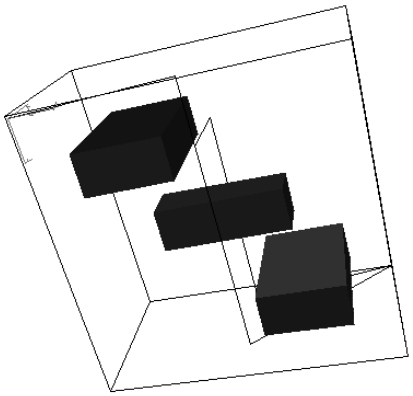
The easy case is when a clock is running concurrently to the HDA. This clock is supposed to represent an absolute time. This situation is drawn in Figure 1(a). The date map D_X, which associates to every point of the above topological space (that is to say a state of the corresponding HDA) the date when it occurs, is the projection map on the temporal line. Two achronal cuts (the term "achronal" is borrowed from [10]) $D_X^{-1}(T)$ and $D_X^{-1}(T')$ at the date T and T' are depicted in Figure 1(a). In this situation, the execution paths are the continuous map ϕ from $[0,1]$ to X such that $D_X \circ \phi$ is a non-decreasing map from $[0,1]$ to $\mathbb{R}$.

Deforming the HDA of the picture means deforming every cut of (X, D_X) in a continuous way. In other terms, in most cases, whenever f and g are two morphisms of topological spaces from X to Y (resp. from Y to X) such that for every $t \in \mathbb{R}$, the maps f and g induce reciprocal homotopy equivalences between $(D_X)^{-1}(t)$ and $(D_Y)^{-1}(t)$ and such that $D_X(X)$ and $D_Y(Y)$ are two homeomorphic subsets of $\mathbb{R}$, then one can say that f and g are reciprocal deformations between (X, D_X) and (Y, D_Y). In this case, both HDAs (X, D_X) and (Y, D_Y) are equal up to deformation.

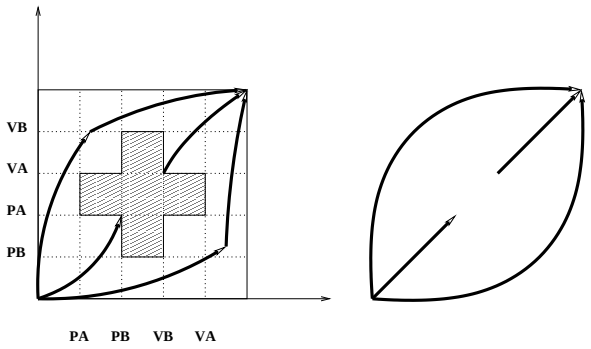
There are some exceptions anyway. In a good theory it is indeed almost sure that an oriented line from a state α to another state β does not represent the same HDA as the HDA corresponding to one point. In the first case, there is a computation and in the second case there is not. Therefore these two HDAs are not equal up to deformation. In other terms, one cannot contract a temporal line to one of these extremal points. There are other exceptions mentioned to me by Stefan Sokolowski. They do not matter because we only want to give a geometric intuition of the content of this paper. The precise



(a) A HDA as a flow of execution paths



(b) The room with three barriers



(c) The Swiss Flag

Fig. 1. Examples of HDA

link between the topological and the ω-categorical approach is still an open question (see the end of Section 6).

So a good way to think of invariants of HDA as in Figure 1(a) consists of thinking that an invariant of HDA is an invariant of cuts. And one can start constructing an invariant of HDA by choosing one real number t, one invariant coming from algebraic topology F, and by considering the map $(X, D_X) \mapsto F \circ (D_X)^{-1}(t)$.

Now let us remove the hypothesis of an absolute time. It becomes impossible to consider a map as above. The main idea of this paper is then as follows. There are three kind of geometric regions in a HDA : 1) the branching areas of execution paths, 2) the merging areas of execution paths, 3) and the oriented globes. And each of these regions gives rise to one simplicial set (the

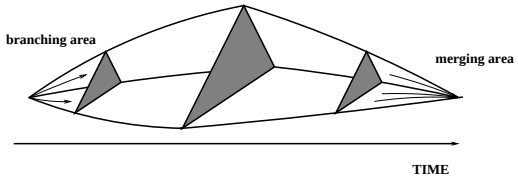


Fig. 2. The fundamental structure

branching nerve, the merging nerve and the globular nerve) which represents respectively the topology of all cuts close to branching or merging areas of execution paths or in the middle of the globes. Moreover, as depicted in Figure 2, each topological cut in a middle of a globe can also be considered as a topological cut close to a branching and a merging area of execution paths (it suffices to hide –for example– the right part of Figure 2 with the hand to see a branching area appearing). Therefore there will be two morphisms of simplicial sets from the globular nerve to respectively the branching and the merging nerves.

Considering only achronal cuts is not sufficient to characterize the homotopy type of an HDA. Loosely speacking, if it was sufficient, the simplicial set $N_*(X) = C^0_{achronal}(\Delta^*, X)$, where Δ^n would be here the usual topological n-simplex and where $C^0_{achronal}(\Delta^*, X)$ would be the continuous maps f from Δ^* to X such that $f(x) \leq f(y)$ if and only if $f(x) = f(y)$, would contain enough information to characterize the homotopy class of the HDA X. But the information that tells us how a given vertical simplex is related to another one along the flow of execution paths is missing. There are too many simplexes in this simplicial set and there is not enough information to make the required identifications. In the room with three barriers of Figure 1(b) (borrowed from [10]), there are two non-dihomotopic execution paths althrough all achronal cuts in this example are path-connected. We will explain later that the three simplicial nerves constructed in this paper do see this situation. This means that the three nerves will also contain information not corresponding to any achronal cut.

The interest of considering such invariants is detailed in many papers [14] [13]. Loosely speaking, such deformations leave the most interesting properties of a HDA unchanged. For example both HDAs of Figure 1(c) are essentially the same (beware of the fact that there is no absolute time in this latter example). As explained in [9], the state γ represents a deadlock and the state δ an unreachable state. Compare the possible execution paths on the left and the four execution paths on the right. These are essentially the same! This means that an algorithm reducing an HDA by a deformation before doing calculations would be more efficient than any other algorithm.

3 Modeling HDA by means of globular ω-categories

Roughly speaking, in a given globular ω-category $\mathcal{C}$, 0-morphisms represent the states of the corresponding automaton, 1-morphisms represent all possible execution paths in the HDA and higher dimensional morphisms represent homotopies between morphisms of lower dimension. They represent the execution of several tasks carried out at the same time. In particular 2-morphisms represent homotopies between execution paths. The composition of execution paths matches exactly the composition of 1-morphisms. Higher dimensional composition laws formalize the composition of higher dimensional homotopies (cf. Figure 3(a)). As already noticed in [21], the axioms of globular ω-categories encode the geometric properties of compositions of execution paths and homotopies between them.

Let us recall the definition of ω-category in three steps (see [5] [26] [24] for more details) :

Definition 3.1 A 1-*category* is a pair $(A, (*, s, t))$ satisfying the following axioms :

(i) A is a set

(ii) s and t are set maps from A to A respectively called the source map and the target map

(iii) for $x, y \in A$, $x * y$ is defined as soon as $tx = sy$

(iv) $x * (y * z) = (x * y) * z$ as soon as both members of the equality exist

(v) $sx * x = x * tx = x$, $s(x * y) = sx$ and $t(x * y) = ty$ (this implies $ssx = sx$ and $ttx = tx$).

Definition 3.2 A 2-*category* is a triple $(A, (*_0, s_0, t_0), (*_1, s_1, t_1))$ such that

(i) both pairs $(A, (*_0, s_0, t_0))$ and $(A, (*_1, s_1, t_1))$ are 1-categories

(ii) $s_0 s_1 = s_0 t_1 = s_0$, $t_0 s_1 = t_0 t_1 = t_0$, and for $i \geq j$, $s_i s_j = t_i s_j = s_j$ and $s_i t_j = t_i t_j = t_j$ (Globular axioms)

(iii) $(x *_0 y) *_1 (z *_0 t) = (x *_1 z) *_0 (y *_1 t)$ (Godement axiom)

(iv) if $i \neq j$, then $s_i(x *_j y) = s_i x *_j s_i y$ and $t_i(x *_j y) = t_i x *_j t_i y$.

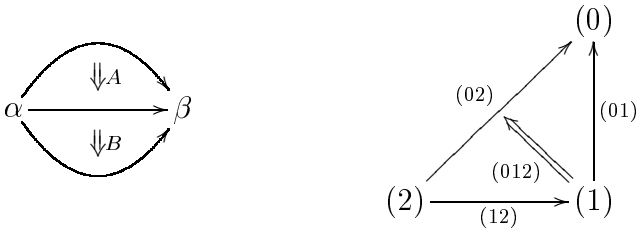
Definition 3.3 A *globular ω-category* $\mathcal{C}$ is a set A together with a family $(*_n, s_n, t_n)_{n \geq 0}$ such that

(i) for any $n \geq 0$, $(A, (*_n, s_n, t_n))$ is a 1-category

(ii) for any $m, n \geq 0$ with $m < n$, $(A, (*_m, s_m, t_m), (*_n, s_n, t_n))$ is a 2-category

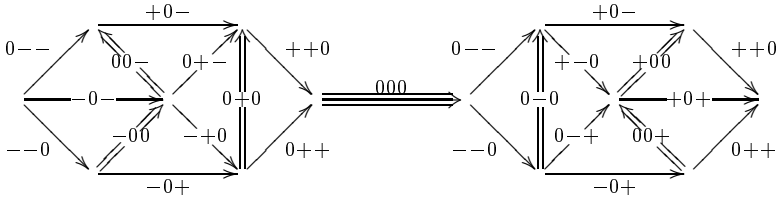
(iii) for any $x \in A$, there exists $n \geq 0$ such that $s_n x = t_n x = x$ (the smallest of these n is called the dimension of x).

A n-dimensional element of $\mathcal{C}$ is called a n-morphism. A 0-morphism is also called a state of $\mathcal{C}$, and a 1-morphism an arrow. If x is a morphism of an ω-category $\mathcal{C}$, we call $s_n(x)$ the n-source of x and $t_n(x)$ the n-target of x.



(a) Composition of two 2-morphims

(b) The ω-category Δ^2



(c) The ω-category I^3

Fig. 3. Some ω-categories (a k-fold arrow symbolizes a k-morphism)

The category of all ω-categories (with the obvious morphisms) is denoted by ωCat. The corresponding morphisms are called ω-functors.

As fundamental examples of ω-categories, there are the ω-category I^n associated to the n-dimensional cube and that of the n-dimensional simplex (this latter is denoted by Δ^n). For the cube, the older attempt of constructing a structure of ω-category on the set of faces of the n-cube is maybe in [1]. As for the n-simplex, the seminal work is [26]. Since then, many constructions have been proposed.

Both families of ω-categories can be characterized in the same way. The first step consists of labelling all faces of the n-cube and of the n-simplex. For the n-cube, this consists of considering all words of length n in the alphabet $\{-, 0, +\}$, one word corresponding to the barycenter of a face (with $00\ldots0$ (n times) $=: 0_n$ corresponding to its interior). As for the n-simplex, its faces are in bijection with strictly increasing sequences of elements of $\{0, 1, \ldots, n\}$. A sequence of length $p+1$ will be of dimension p. If x is a face, let $R(x)$ be the set of faces of x seen respectively as a sub-cube or a sub-simplex. If X is a set of faces, then let $R(X) = \bigcup_{x\in X} R(x)$. Notice that $R(X \cup Y) = R(X) \cup R(Y)$ and that $R(\{x\}) = R(x)$. Then I^n and Δ^n are the free ω-categories generated by the $R(x)$ with the rules

(i) For x p-dimensional with $p \geq 1$, $s_{p-1}(R(x)) = R(s_x)$ and $t_{p-1}(R(x)) = R(t_x)$ where s_x and t_x are the sets of faces defined below.

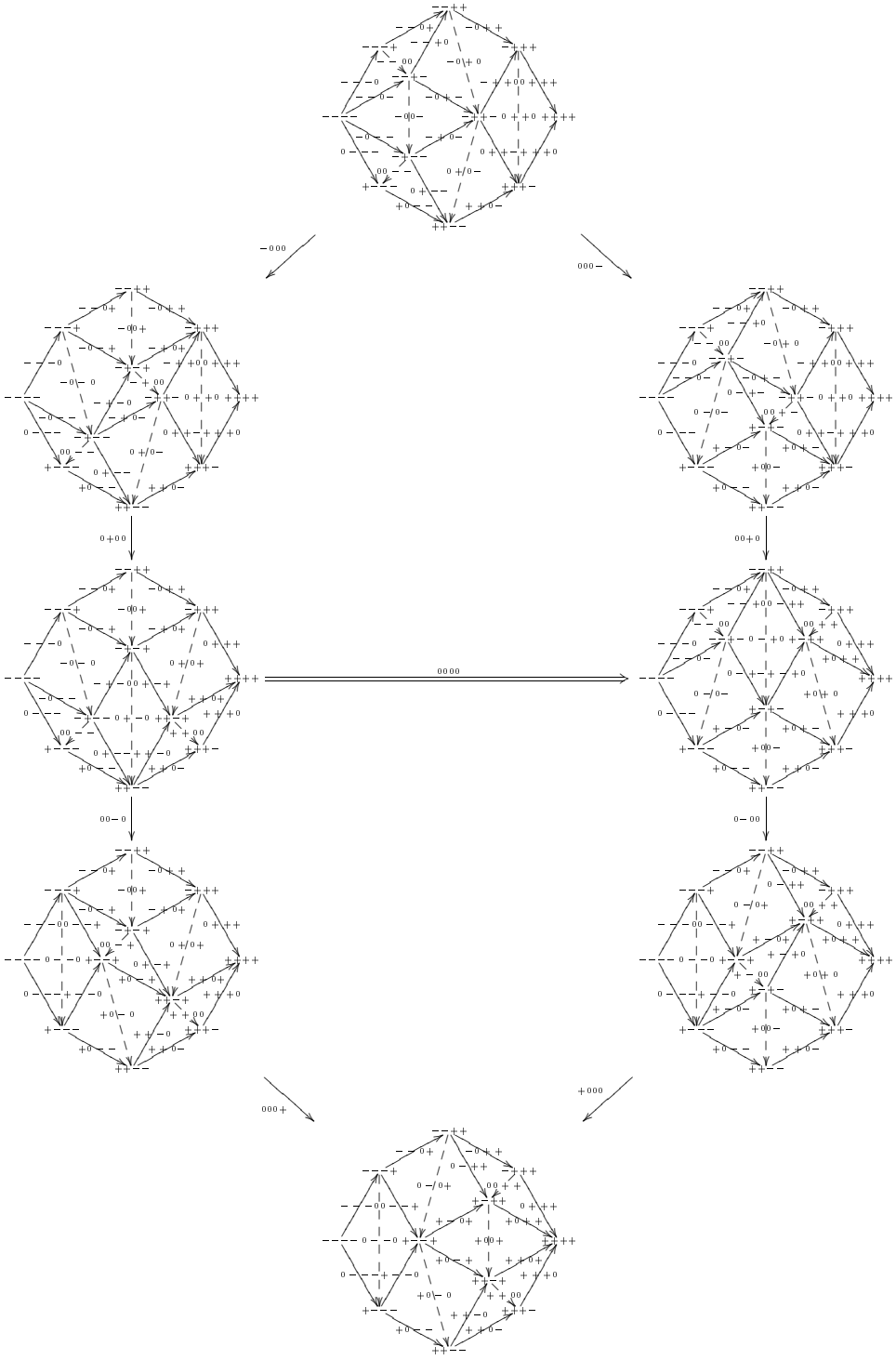


Fig. 4. The ω-category I^4

(ii) If X and Y are two elements of I^n (resp. Δ^n) such that $t_p(X) = s_p(Y)$ for some p, then $X \cup Y$ belongs to I^n (resp. Δ^n) and $X \cup Y = X *_p Y$. The slogan is : "Composition means union".

Only the definition of s_x and t_x differs from one case to the other one. Let us give the computation rule in some examples. For the cube, the i-th zero is replaced by $(-)^i$ (resp. $(-)^{i+1}$) for s_x (resp. t_x). For example, one has $s_{0+00} = \{\text{-+00}, 0\text{++0}, 0\text{+0-}\}$ and $t_{0+00} = \{\text{++00}, 0\text{+-0}, 0\text{+0+}\}$.

For the simplex, $s_{(04589)} = \{(4589), (0489), (0458)\}$ (the elements in odd position are removed) and $t_{(04589)} = \{(0589), (0459)\}$ (the elements in even position are removed).

The above constructions are examples of free ω-categories generated by some data (see for example [15], [17] or [25] for possible descriptions of these data). Using the construction of I^n, one can construct the free ω-category generated by a cubical set. A cubical set K is indeed a set-valued presheaf over some small category $\square$ whose objects are natural numbers and whose morphisms encode the axioms of cubical sets (exactly in the same way that the small category Δ does for simplicial sets) [7] [16]. It is a general fact that a cubical set K is in a canonical way the direct limit of its cubes : $K = \int^{\underline{n} \in \square} K_n.\square(-, \underline{n})$ where the integral sign is the coend construction [19] and $K_n.\square(-, \underline{n})$ means the sum of "cardinal of K_n" copies of $\square(-, \underline{n})$. It then suffices to paste the ω-categories associated to every n-cube of K in the same way that they are pasted in K, that is to consider $\int^{\underline{n} \in \square} K_n.I^n$, to obtain the *$\omega$-categorical realization* of the cubical set K. For instance the ω-categorical realization of a 1-dimensional cubical set is an ω-category whose 1-morphisms are exactly the arrows of the cubical set and all possible compositions of these arrows (further details in the informal part of [13]). Exactly in the same way, the topological space $\int^{\underline{n} \in \square} K_n.[0,1]^n$, where $[0,1]^n$ is the topological n-cube, is nothing else but the usual geometric realization of the cubical set K [19].

4 Fundamental constructions

First of all here are some important definitions. The $\mathbb{N}$-graded set $\mathcal{C}[1]$ is obtained from $\mathcal{C}$ by removing the 0-morphisms, by considering the 1-morphisms of $\mathcal{C}$ as the 0-morphisms of $\mathcal{C}[1]$, the 2-morphisms of $\mathcal{C}$ as the 1-morphisms of $\mathcal{C}[1]$, etc. with an obvious definition of the source and target maps and of the composition laws. The map $T : \mathcal{C} \mapsto \mathcal{C}[1]$ does not induce a functor from ωCat to itself because ω-functors can contract 1-morphisms and because with our conventions, a 1-source or a 1-target can be 0-dimensional. Hence the following definition :

Definition 4.1 [13] An ω-category $\mathcal{C}$ is *non-1-contracting* if $\mathcal{C}[1]$ is an ω-category (or equivalently if s_1x and t_1x are 1-dimensional as soon as x is not 0-dimensional). Let f be an ω-functor from $\mathcal{C}$ to $\mathcal{D}$. The morphism f is *non-1-contracting* if for any 1-dimensional $x \in \mathcal{C}$, the morphism $f(x)$ is a

1-dimensional morphism of $\mathcal{D}$.

Definition 4.2 The category of non-1-contracting ω-categories with the non-1-contracting ω-functors is denoted by ωCat_1.

Following [8], an augmented simplicial set is a simplicial set $(X_n)_{n\geq 0}$ endowed with an additional set X_{-1} and an additional set map ∂_{-1} from X_0 to X_{-1} such that $\partial_{-1}\partial_0 = \partial_{-1}\partial_1$ where ∂_0 and ∂_1 are the two face maps from X_1 to X_0. The "simplicial homology" functor H_* from the category of augmented simplicial sets $Sets_+^{\Delta^{op}}$ to the category of abelian groups Ab is defined as the usual one for $* \geq 1$ and by setting $H_0(X) = Ker(\partial_{-1})/Im(\partial_0 - \partial_1)$ and $H_{-1}(X) = \mathbb{Z}X_{-1}/Im(\partial_{-1})$ whenever X is an augmented simplicial set.

4.1 The branching and merging nerves

The branching and merging nerves are dual from each other. We set

$$\omega Cat(I^{n+1},\mathcal{C})^\eta := \{x \in \omega Cat(I^{n+1},\mathcal{C}), x(\eta \dots [0]_i \dots \eta) \text{ 1-dimensional}\}$$

where $\eta \in \{-,+\}$ and where the expression $\eta \dots [0]_i \dots \eta$ denotes the word on $\{\eta, 0\}$ with exactly one zero in the i-th position and for all (i,n) such that $0 \leq i \leq n$, the face maps ∂_i from $\omega Cat(I^{n+1},\mathcal{C})^\eta$ to $\omega Cat(I^n,\mathcal{C})^\eta$ are the arrows ∂^η_{i+1} defined by

$$\partial^\eta_{i+1}(x)(k_1 \dots k_{n+1}) = x(k_1 \dots [\eta]_{i+1} \dots k_{n+1})$$

and the degeneracy maps ϵ_i from $\omega Cat(I^n,\mathcal{C})^\eta$ to $\omega Cat(I^{n+1},\mathcal{C})^\eta$ are the arrows Γ^η_{i+1} defined by setting

$$\Gamma^-_i(x)(k_1 \dots k_n) := x(k_1 \dots \max(k_i, k_{i+1}) \dots k_n)$$
$$\Gamma^+_i(x)(k_1 \dots k_n) := x(k_1 \dots \min(k_i, k_{i+1}) \dots k_n)$$

with the order $- < 0 < +$.

Definition 4.3 [13] The *η-corner simplicial nerve* $\mathcal{N}^\eta$ is the functor from ωCat_1 to $Sets_+^{\Delta^{op}}$ defined by $\mathcal{N}^\eta_n(\mathcal{C}) := \omega Cat(I^{n+1},\mathcal{C})^\eta$ for $n \geq 0$ and with $\mathcal{N}^\eta_{-1}(\mathcal{C}) = \mathcal{C}_0$ and endowed with the augmentation map ∂_{-1} from $\mathcal{N}^\eta_0(\mathcal{C}) = \mathcal{C}_1$ to $\mathcal{N}^\eta_{-1}(\mathcal{C}) = \mathcal{C}_0$ defined by $\partial_{-1} = s_0$ (resp. $\partial_{-1} = t_0$) if $\eta = -$ (resp. $\eta = +$).

In the sequel, "$-$-corner" means *branching* and "$+$-corner" means *merging*. Set

$$H^\eta_{n+1}(\mathcal{C}) := H_n(\mathcal{N}^\eta(\mathcal{C}))$$

for $n \geq -1$. These homology theories are called branching and merging homology respectively. The abelian group $H^-_0(\mathcal{C})$ (resp. $H^+_0(\mathcal{C})$) is the free abelian group generated by the final (resp. initial) states of $\mathcal{C}$.

The evaluation map ev defined by $ev(x) = x(0_{n+1})$ for $x \in \omega Cat(I^{n+1},\mathcal{C})^\eta$ associates to any such x the label of the interior of x. A 2-simplex of the branching nerve (that is an ω-functor from I^3 to $\mathcal{C}$) is depicted in Figure 5. The "2-simplex part" is described by the dark triangle. The simplicial structure of these two nerves comes from the fact that close to a corner, the intersection

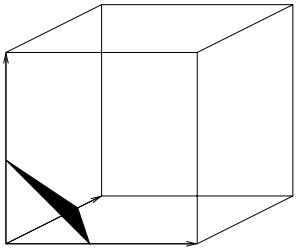


Fig. 5. model of 2-simplex in the branching nerve

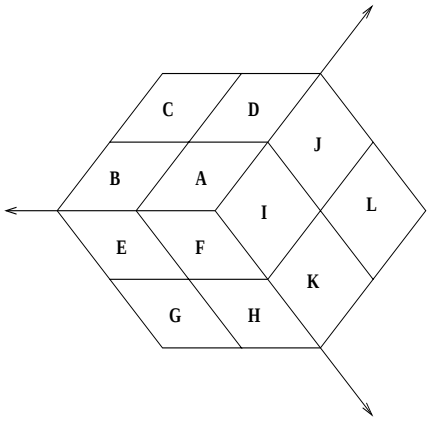


Fig. 6. A 2-dimensional branching area

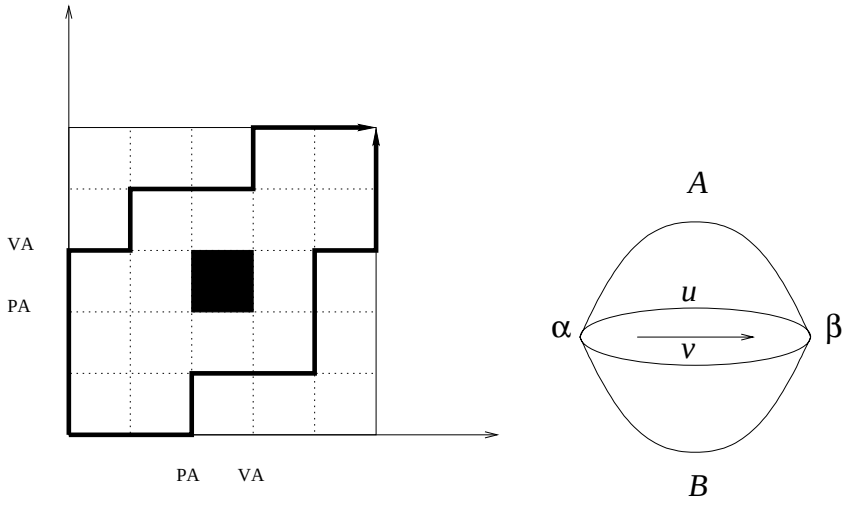
of an n-cube by an hyperplane is an $(n-1)$-simplex. Considering the maps Γ^-_{i+1} and Γ^+_{i+1} is not new (cf. the operations Γ_{i+1} in [6] and $\Gamma_{i+1}, \Gamma'_{i+1}$ in [2]). Our notations are adapted to the simplicial structure of Definition 4.3 noticed in [13] for the first time.

Figure 6 represents a 2-dimensional branching area. It corresponds to the homology class of the cycle $(A)-(F)+(I)$. One can prove that the cycles $(A,B,C,D)-(E,F,G,H)+(I,J,K,L)$, $(A)-(F,H)+(I,J)$ correspond to the same homology class as $(A)-(F)+(I)$. The exact statement can be found in [11]: it uses the cubical analogue of the globular composition laws $*_n$ of Definition 3.2 and Definition 3.3. It means that negative (resp. positive) corner homology theories describe the branching (resp. merging) areas of execution paths in a HDA. In other terms, the homology class does not depend on a cubification of the HDA. Therefore they correct the main drawback of the homology theories of [14].

4.2 The globular nerve

If $\mathcal{C}$ is an ω-category and if $x \in \omega Cat(\Delta^n, \mathcal{C}[1])$, one can set

$$\epsilon_i(x)(\sigma_0 < \ldots < \sigma_{n+1}) = x(\sigma_0 < \ldots < \widehat{i} < \sigma_k - 1 < \ldots < \sigma_{n+1} - 1)$$
$$\partial_i(x)(\sigma'_0 < \ldots < \sigma'_{n-1}) =$$



(a) Example from distributed database theory: two 1-morphisms are depicted

(b) A 3-dimensional hole

Fig. 7. Examples of cycles

$$x(\sigma'_0 < \ldots < \sigma'_{k-1} < \sigma'_k + 1 < \ldots < \sigma'_{n-1} + 1)$$

where $\sigma'_k, \ldots, \sigma'_{n-1} \geq i$ and where $\widehat{i}$ means that i is removed if it appears and that bigger σ_ℓ are replaced by $\sigma_\ell - 1$.

It can be checked that $\epsilon_i(x)$ (resp. $\partial_i(x)$) are ω-functors from Δ^{n+1} (resp. Δ^{n-1}) to $\mathcal{C}[1]$ and $(\omega Cat(\Delta^*, \mathcal{C}[1]), \partial_i, \epsilon_i)$ is a simplicial set which will be called the simplicial nerve of $\mathcal{C}[1]$.

Definition 4.4 [12] The *globular simplicial nerve* $\mathcal{N}^{gl}$ is the functor from ωCat_1 to $Sets_+^{\Delta^{op}}$ defined by $\mathcal{N}_n^{gl}(\mathcal{C}) = \omega Cat(\Delta^n, \mathcal{C}[1])$ for $n \geq 0$ and with $\mathcal{N}_{-1}^{gl}(\mathcal{C}) = \mathcal{C}_0 \times \mathcal{C}_0$, and endowed with the augmentation map ∂_{-1} from $\mathcal{N}_0^{gl}(\mathcal{C}) = \mathcal{C}_1$ to $\mathcal{N}_{-1}^{gl}(\mathcal{C}) = \mathcal{C}_0 \times \mathcal{C}_0$ defined by $\partial_{-1}x = (s_0x, t_0x)$.

The evaluation map ev defined for $x \in \omega Cat(\Delta^n, \mathcal{C}[1])$ by

$$ev(x) = x((0 \ldots n))$$

associates to any such x the label of the interior of x. Intuitively, elements of $\mathcal{N}_n^{gl}(\mathcal{C})$ are full $(n+1)$-globes. Figure 8 depicts a 2-simplex in the globular nerve. Indeed, if x is an ω-functor from Δ^n to $\mathcal{C}[1]$ for some n, then the set map from Δ^n to $\mathcal{C}$ given by $(\sigma_0 \ldots \sigma_r) \mapsto s_0x((\sigma_0 \ldots \sigma_r))$ (resp. $(\sigma_0 \ldots \sigma_r) \mapsto t_0x((\sigma_0 \ldots \sigma_r))$) is the constant map because of the globular equations in Definition 3.2. In other terms, the globular nerve contains some information about the simplexes included in the HDA and which are in vertical position.

Set

$$H_{n+1}^{gl}(\mathcal{C}) := H_n(\mathcal{N}^{gl}(\mathcal{C}))$$

for $n \geq -1$. This homology theory is called the globular homology. An

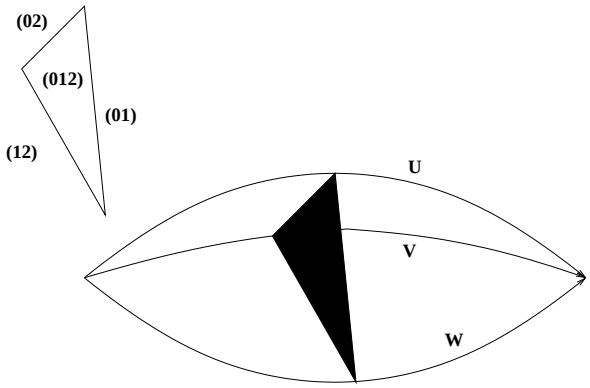
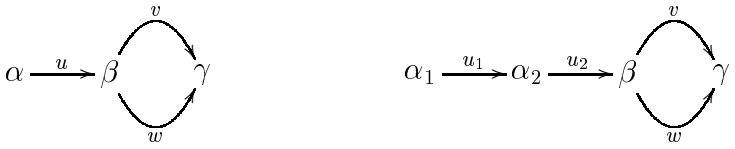


Fig. 8. Globular 2-simplex



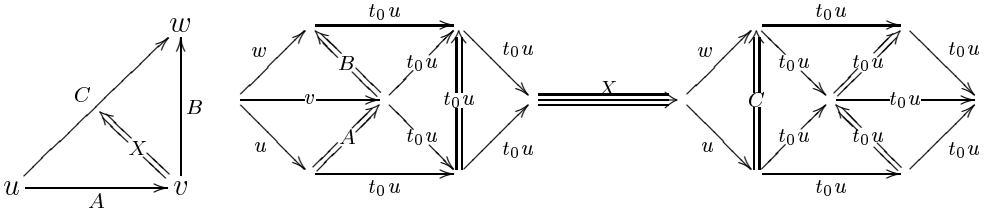
(a) $\mathcal{C}$ (b) Subdivision of u in $\mathcal{C}$

Fig. 9. Example of $T1$-deformation

example of globular cycle of dimension 1 is $\gamma_1 - \gamma_2$ of Figure 7(a) where γ_1 and γ_2 are the execution paths drawn from the point of coordinates $(0, 0)$ to the point of coordinates $(5, 5)$. We call it an oriented 1-dimensional loop. An example of a globular cycle of dimension 2 is $A - B$ of Figure 7(b) (more precisely, A means here the ω-functor from $\Delta^1 = I^1$ to $\mathcal{C}[1]$ such that the interior is labelled by A and idem for B). Looking back to Figure 1(b), we see that the corresponding first globular group does not vanish : this means in this case that the globular nerve contains information not related to any achronal cuts of the HDA.

Like for the corner homologies, we can arrive at similar conclusions with the globular homology. Subdividing p-morphisms with $p \geq 2$ (Figure 3(a) can be seen as the subdivision of a 2-morphism in two parts A and B) in a HDA leaves the globular homology unchanged.

As for the subdivision of 1-morphisms, one can prove that subdivisions of indecomposable 1-morphisms leave both corner homology theories unchanged. This is not the case for the globular homology. Indeed if both corner homologies of HDAs of Figure 9(a) and Figure 9(b) are equal (to $\mathbb{Z}$), this property fails for the globular homology : the first globular homology group of Figure 9(a) is equal to $\mathbb{Z}^{\oplus 2}$ (the free abelian group generated by $v - w$, $u *_0 v - u *_0 w$) and the first globular homology group of Figure 9(b) is equal to $\mathbb{Z}^{\oplus 3}$ (the free abelian group generated by $v - w$, $u_2 *_0 v - u_2 *_0 w$, $u_1 *_0 u_2 *_0 v - u_2 *_0 w$).



(a) A 2-globular simplex $\widetilde{X}$

(b) The 2-simplex $h^-(\widetilde{X})$

Fig. 10. Illustration of h^-

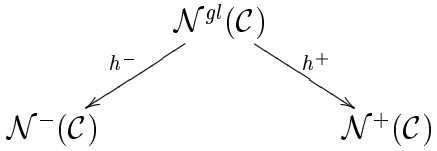


Fig. 11. The fundamental diagram

4.3 The morphisms from the globular to the corner nerves

Both morphisms of simplicial sets h^η from the globular nerve to the corner nerves arise from the canonical inclusion map from $\mathcal{N}^{gl}(\mathcal{C})$ to $\mathcal{N}^\eta(\mathcal{C})$. They can be characterized by the following statement:

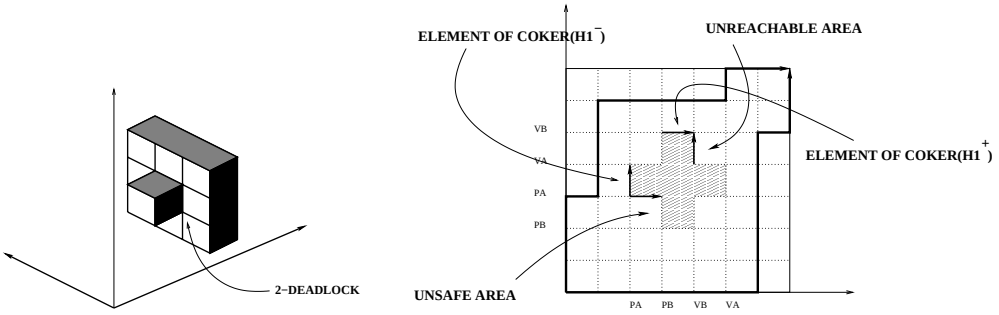
Theorem 4.5 [12] *Let $\eta \in \{-,+\}$. There exists one and only one natural transformation h^η from $\mathcal{N}^{gl}$ to $\mathcal{N}^\eta$ such that $ev \circ h^\eta = ev$.*

Take a globular 2-simplex as in Figure 8. It can be mapped to an ω-functor from I^3 to $\mathcal{C}$ as in Figure 5 by labelling the faces of I^3 with only 0 and $-$ in their description with the corresponding label of the original figure and by labelling all other faces of I^3 by β. For example Figure 10(b) represents the image of the globular 2-simplex of Figure 10(a) by h^- (see the convention of labelling in Figure 3(b) and Figure 3(c)): notice that $t_0u = t_0v = t_0w = t_0A = t_0B = t_0C = t_0X$.

In homology , the morphism h^- (resp. h^+) associates to any n-globe its corresponding n-dimensional branching (resp. merging) area of execution paths.

We already explained in [13] the link between the simplicial homology of the cone of h^- (resp. h^+) with the n-dimensional deadlocks (resp. the unreachable n-morphisms) for some classes of HDAs.

Indeed the morphisms h^- and h^+ associate in homology to any oriented loop of any dimension its corresponding negative or positive corners. We can immediately see an application of these maps. Looking back to the Swiss Flag example of Figure 1(c), it is clear that the cokernel of h_1^- does not vanish, because of the deadlock and the unsafe area. A negative corner which yields a



(a) Example of 2-deadlock

(b) Unsafe area and unreachable area in a concurrent machine with semaphores

Fig. 12. Applications in computer science

non trivial element in this cokernel is drawn in Figure 12(b). In the same way, the cokernel of h_1^+ in the Swiss Flag example still does not vanish, because of the unreachable state and the unreachable area. A positive corner which yields a non zero element of this cokernel is represented in Figure 12(b).

In Figure 12(a), execution paths are supposed to be the continuous maps γ from $[0,1]$ to the complement of the depicted obstacle such that the composite $\pi_x \circ \gamma$, $\pi_y \circ \gamma$ and $\pi_z \circ \gamma$ are non-decreasing maps from $[0,1]$ to $\mathbb{R}$, where π_x, π_y and π_z are the projections on the axes. One sees a non-trivial element of the cokernel of h_2^- which detects the presence of the 2-deadlock.

5 Deforming HDA : some speculations and perspectives

Now we would like to speculate about what should be a "good" invariant of HDA. An ω-category seen as a HDA can be deformed in different ways : by deforming p-morphisms with $p \geq 2$ (T2), by concatening or subdividing 1-morphisms (T1), i.e. execution paths in the corresponding automaton. Table 1 explains the behavior of the objects introduced in this paper with respect to these deformations. About the "almost" : the deformation of a p-morphism u for $p \geq 2$ corresponds in the globular nerve to a simplicial deformation of any simplex x such that the image of x contains u ; the same deformation corresponds to a simplicial deformation of an n-simplex y of, for example, the negative corner nerve only if the label u belongs to the negative part of y. In the branching nerve, the positive part of an ω-functor from I^{n+1} to $\mathcal{C}$ is a "dead part".

The non-invariance with respect to $T1$-deformations is also an obstacle to find appropriate invariants for locally partially ordered topological spaces [10]. Consider for instance the HDA of Figure 9(a). By thickening the 1-morphisms as in Figure 13, one obtains a HDA such that its first globular homology group is an infinite sum of copies of $\mathbb{Z}$. To understand this fact, let us come back

Deformation type	T2-invariant	T1-invariant
Globular nerve	yes	no
Corner nerves	almost	no
Globular homology	yes	no
Corner homologies	yes	yes

Table 1
Behavior of the constructions w.r.t. deformations of HDA

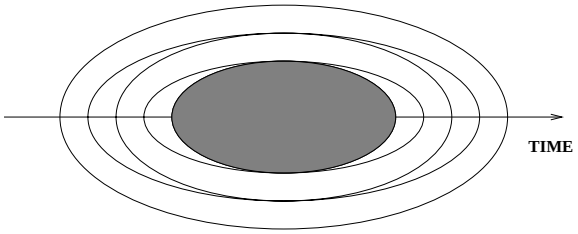


Fig. 13. Thick 1-dimensional oriented globe

to Figures 9. The more subdivisions in u there are , the bigger the rank of the first globular homology group is. At the limit with the real line, one obtains an infinite number of globular 1-cycles. It is therefore necessary to make supplemental identifications within the nerves in order to get "good" invariants. Here is now a fundamental fact :

Claim 5.1 *Suppose that $\mathcal{C}$ is an object of ωCat_1 such that its 1-morphisms are never invertible (or equivalently if x and y are two 1-morphisms, $x *_0 y$ is 1-dimensional if the expression makes sense). For $n \geq 0$ and $x \in \mathcal{N}_n^{gl}(\mathcal{C})$, let $S(x) := s_0 ev(x)$ and $T(x) := t_0 ev(x)$. Suppose that for another $y \in \mathcal{N}_n^{gl}(\mathcal{C})$, $T(x) = S(y)$. Let $x * y$ be the map from Δ^n to $\mathcal{C}[1]$ defined by*

$$(x * y)((\sigma_0 \dots \sigma_r)) := x((\sigma_0 \dots \sigma_r)) *_0 y((\sigma_0 \dots \sigma_r)).$$

Then :

(i) *For $n \geq 0$, $(\mathcal{N}_n^{gl}(\mathcal{C}) \cup \mathcal{C}_0, S, T, *)$ is a 1-category by considering elements of $\mathcal{C}_0$ as 0-morphisms. In particular $x * y$ is an ω-functor for $n \geq 0$. In this 1-category, 1-morphisms are still never invertible and therefore it can be seen as a small category by adding 1-dimensional identities.*

(ii) *With the obvious structure of a small category on $(\mathcal{C}_0 \times \mathcal{C}_0, \mathcal{C}_0, S, T, *)$ defined by setting $S(u, v) := u$, $T(u, v) := v$ and $(u, v) * (v, w) := (u, w)$, then the globular nerve becomes an augmented simplicial object in the category of small categories.*

Intuitively, for $n \geq 0$, the 1-category $(\mathcal{N}_n^{gl}(\mathcal{C}) \cup \mathcal{C}_0, S, T, *)$ represents the temporal structure of the $(n + 1)$-dimensional paths. Using the canonical

inclusion of the globular nerve in both corner nerves, both corner nerves can be endowed with a structure of augmented simplicial object in the category of 1-category with source and target maps only partially defined.

Let $\overrightarrow{N}$ be a functor from the category of small categories to any good category whose simplicial objects have satisfactory properties with respect to the usual structures in algebraic topology (for example the usual category of simplicial sets up to homotopy). We have a wide degree of freedom for the choice of this good category. Indeed Baues's book [4] explains that the main theorems of algebraic topology (as the Hurewicz theorem or some Whitehead theorems) can be recovered from the axiomatic theory of cofibration categories. Suppose that any subdivision of a morphism in two morphisms (as for example the canonical functor from the HDA of Figure 9(a) to that of Figure 9(b) such that $u \mapsto u_1 *_0 u_2$ and being the identity map elsewhere) induces an homotopy equivalence by $\overrightarrow{N}$. Then the composite functor $\overrightarrow{N} \circ \mathcal{N}^{gl}$ would be $T2$-invariant because of $\mathcal{N}^{gl}$ and $T1$-invariant because of $\overrightarrow{N}$. So far only corner homologies are invariant by all possible types of deformations of HDA. Street's simplicial nerve [26] of 1-categories (our 1-categories have no invertible 1-morphisms and therefore they can be viewed as small category by adding identity 1-dimensional elements) and the classifying space of small categories (see for example [22] for further details) satisfy this property. Unfortunately, the total homology of the bisimplicial set associated to the ω-category of Figure 9(a) is equal to $\mathbb{Z}$ in both cases, and not to $\mathbb{Z} \oplus \mathbb{Z}$ as required by the fact that one does not want the 1-morphisms of a given ω-category to be contracted in the same homotopy class.

6 Concluding remarks

Throughout this paper, we have presented a new approach of the topology of HDA based upon the introduction of three new simplicial nerves and two natural morphisms of simplicial sets for any HDA. It opens the perspective of deeply relating the geometry of HDA with homological algebra and other usual tools developed for algebraic topology. The construction has the following properties :

(i) Every orthogonal (or anti-diagonal) cut of a HDA behaves like a true topological space.

(ii) There are four fundamental types of cuts : close to branching or merging areas of execution paths, in the middle of the globes, and all cuts.

(iii) The three first types of cuts give rise to three new simplicial sets, and two morphisms of simplicial sets ; the last one is the direct limit of the diagram depicted in Figure 11.

There are two types of deformations of HDA (T1 and T2). As shown in the above table, most of the functors introduced here are $T2$-invariant but not $T1$-invariant. The $T1$-invariance could be related to finding a new nerve

of small categories (with the source and target maps not necessarily defined everywhere).

Getting three $T1$-invariant and $T2$-invariant simplicial nerves would enable us to define the notion of homotopy equivalent HDAs as follows : the maps f from a HDA X to a HDA Y and g from Y to X would be reciprocal homotopy equivalences of HDA if and only if $\mathcal{N}^{gl}(f)$ and $\mathcal{N}^{gl}(g)$ (resp . $\mathcal{N}^{-}(f)$ and $\mathcal{N}^{-}(g)$, $\mathcal{N}^{+}(f)$ and $\mathcal{N}^{+}(g)$) were reciprocal homotopy equivalences of simplicial sets. Similar constructions for locally partially ordered topological spaces would enable us to make precise the following idea : up to homotopy of HDA, the category of locally partially ordered topological spaces, the category of cubical sets satisfying some Kan conditions, and the category of non-1-contracting ω-categories with some additional technical conditions (for the three simplicial nerves to be Kan) should be equivalent. A similar equivalence in the framework of usual algebraic topology is proved in [18] between CW-complexes and weak ω-groupoids modulo weak equivalences.

References

[1] Aitchison, I. R., *The geometry of oriented cubes* (1986), Macquarie Mathematics Reports 86-0082.

[2] Al-Agl, F. A. A., "Aspects of multiple categories," Ph.D. thesis, University of Wales, Department of Pure Mathematics, University College of North Wales, Bangor, Gwynedd LL57 1UT, U.K. (1989).

[3] Al-Agl, F. A. A., R. Brown and R. Steiner, *Multiple categories: the equivalence of a globular and a cubical approach* (2000), arxiv:math.CT/0007009.

[4] Baues, H.-J., "Combinatorial foundation of homology and homotopy," Springer-Verlag, Berlin, 1999, xvi+362 pp.

[5] Brown, R. and P. J. Higgins, *The equivalence of ∞-groupoids and crossed complexes*, Cahiers Topologie Géom. Différentielle **22** (1981), pp. 371–386.

[6] Brown, R. and P. J. Higgins, *On the algebra of cubes*, J. Pure Appl. Algebra **21** (1981), pp. 233–260.

[7] Crans, S., *Pasting schemes for the monoidal biclosed structure on ωcat* (1995), Utrecht University.

[8] Duskin, J., *Simplicial methods and the interpretation of "triple" cohomology*, Mem. Amer. Math. Soc. **3** (1975), pp. v+135.

[9] Fajstrup, L., E. Goubault and M. Raußen, *Detecting deadlocks in concurrent systems* in: *CONCUR'98: concurrency theory (Nice)*, Springer, Berlin, 1998 pp. 332–347.

[10] Fajstrup, L., E. Goubault and M. Raußen, *Algebraic topology and concurrency* (June 1999), preprint R-99-2008, Aalborg University.

[11] Gaucher, P., *Combinatorics of branchings in higher dimensional automata* (1999), arxiv:math.CT/9912059.

[12] Gaucher, P., *About the globular homology of higher dimensional automata* (2000), arxiv:math.CT/0002216.

[13] Gaucher, P., *Homotopy invariants of higher dimensional categories and concurrency in computer science*, to be published in Math. Structures Comput. Sci. (2000).

[14] Goubault, E., "The Geometry of Concurrency," Ph.D. thesis, École Normale Supérieure (1995).

[15] Johnson, M., *The combinatorics of n-categorical pasting*, J. Pure Appl. Algebra **62** (1989), pp. 211–225.

[16] Kamps, K. H. and T. Porter, "Abstract homotopy and simple homotopy theory," World Scientific Publishing Co. Inc., River Edge, NJ, 1997, x+462 pp.

[17] Kapranov, M. M. and V. A. Voevodsky, *Combinatorial-geometric aspects of polycategory theory: pasting schemes and higher Bruhat orders (list of results)*, Cahiers Topologie Géom. Différentielle Catégoriques **32** (1991), pp. 11–27, international Category Theory Meeting (Bangor, 1989 and Cambridge, 1990).

[18] Kapranov, M. M. and V. A. Voevodsky, *∞-groupoids and homotopy types*, Cahiers Topologie Géom. Différentielle Catégoriques **32** (1991), pp. 29–46, international Category Theory Meeting (Bangor, 1989 and Cambridge, 1990).

[19] MacLane, S., "Categories for the working mathematician," Springer-Verlag, New York, 1971, ix+262 pp., Graduate Texts in Mathematics, Vol. 5.

[20] May, J. P., "Simplicial objects in algebraic topology," D. Van Nostrand Co., Inc., Princeton, N.J.-Toronto, Ont.-London, 1967, vi+161 pp., van Nostrand Mathematical Studies, No. 11.

[21] Pratt, V., *Modeling concurrency with geometry*, in: A. Press, editor, *Proc. of the 18th ACM Symposium on Principles of Programming Languages*, 1991.

[22] Quillen, D., *Higher algebraic K-theory. I* (1973), pp. 85–147. Lecture Notes in Math., Vol. 341.

[23] Rotman, J. J., "An introduction to algebraic topology," Springer-Verlag, New York, 1988, xiv+433 pp.

[24] Steiner, R., *Tensor products of infinity-categories* (1991), University of Glasgow.

[25] Steiner, R., *Pasting in multiple categories*, Theory Appl. Categ. **4** (1998), pp. No. 1, 1–36 (electronic).

[26] Street, R., *The algebra of oriented simplexes*, J. Pure Appl. Algebra **49** (1987), pp. 283–335.

[27] Weibel, C. A., "An introduction to homological algebra," Cambridge University Press, Cambridge, 1994, xiv+450 pp.

Occurrence Counting Analysis for the π-calculus

Jérôme Feret

Laboratoire d'Informatique de l'École Normale Supérieure
ENS-LIENS, 45, rue d'Ulm, 75230 PARIS cédex 5, FRANCE

Abstract

We propose an abstract interpretation-based analysis for automatically proving non-trivial properties of mobile systems of processes. We focus on properties relying on the number of occurrences of processes during computation sequences, such as mutual exclusion and non-exhaustion of resources.

We design a non-standard semantics for the π-calculus in order to explicitly trace the origin of channels and to solve efficiently problems set by α-conversion and non-deterministic choices. We abstract this semantics into an approximate one. The use of a relational domain for counting the occurrences of processes allows us to prove quickly and efficiently properties such as mutual exclusion and non-exhaustion of resources. At last, dynamic partitioning allows us to detect some configurations by which no infinite computation sequences can pass.

1 Introduction

We are interested in automatically proving non-trivial properties of mobile systems of processes. We focus on properties relying on a good description of the multiset of processes that occur inside computation sequences, such as mutual exclusion and non-exhaustion of resources, for instance.

We propose an abstract interpretation-based analysis for the full π-calculus [19,18]. Since, the π-calculus is a communication-based formalism, no analysis can be done without a good approximation of the control-flow. Following Venet's methodology [23], we introduce a non-standard semantics to explicitly capture the origin of channels and to describe non-uniform distributions of processes. Our semantics considers the full π-calculus and is optimized to deal efficiently with non-deterministic choices (no useless thread is created).

We use the abstract interpretation framework [8,6,10] to derive an approximate semantics to analyze both the interaction between processes and the number of occurrences of these processes. We propose a well adapted

This is a preliminary version. The final version can be accessed at
URL: http://www.elsevier.nl/locate/entcs/volume39.html

domain to detect quickly mutual exclusion and non-exhaustion of resources. Our approach relies on the use of a relational domain the height of which is quadratic on the number of distinct counted processes. This relational domain helps in calculating properties of interest in a non-relational domain, by reduction. Complexity problems are solved by using approximated algorithms for calculating this reduction. Number of occurrences of processes defines a good criterion for partitioning: we use dynamic partitioning [3] to abstract precisely the trace semantics of mobile systems, which allows us to detect a set of configurations such that no infinite computation sequence can pass by a configuration in this set. Our methodology may very likely be adapted to other formalisms, such as the mobile ambients [4] for instance.

In Section 3, we define the standard semantics of the π-calculus. We define our non-standard semantics in Section 4. We design a generic abstract analysis in Section 5, and instantiate it in Section 6. We show how to use dynamic partitioning in Section 7.

2 Related work

Counting occurrences and analyzing the control flow of a system are deeply related. We cannot count number of occurrences of processes in a system without a good knowledge of its control flow, but counting occurrences helps in getting a more precise knowledge of the control flow by detecting mutual exclusion between processes and by providing a good criterion for partitioning. Our analysis can directly be combined with our previous analysis [14] to infer a non-uniform description of the interactions between the processes, but can also be adapted to many other flow analyses [2,23].

Only a very few analyses for counting occurrences of processes have been published. [15] proposes an exponential analysis for counting occurrences of processes inside ambients. [20] uses context-dependent counts for infering a more accurate description of the internal structure of processes at the expense of a higher time complexity (an exponential number of processes are distinguished). These analyses encounter the same problem: when a process occurs several times, they cannot decide whether this process is still occuring one or several times after being computed, so they have to consider the two possible cases, which leads to both a loss of precision and an exponential explosion. The use of both a relational domain for globally abstracting sets of multisets of processes and an approximated reduction allows us to solve this problem efficiently. We obtain a very accurate analysis which is polynomial on the number of distinguished processes.

3 π-calculus

The π-calculus [18,19] is used for describing mobile systems of processes which communicate channel names via channels. We consider a lazy synchronous

version of the polyadic π-calculus, inspired by the lazy asynchronous version introduced by Turner [21] and the chemical abstract machine [1] in which communication primitives are very simple, while ensuring the same expressive power. Let *Channel* be a countable set of channel names, the standard semantics of the π-calculus, given in Figure 1, relies on the use of both a reduction relation to define results of process computations, and a congruence relation to reveal redexes.

Example 3.1 We model by a mobile system $\mathcal{S}$ an *ftp* protocol for a server which cannot establish more than three simultaneous connections:

$$\mathcal{S} := (\nu\ \text{port})(\textbf{Allocate} \mid \text{port}![] \mid \text{port}![] \mid \text{port}![])$$

where

$$\begin{aligned}\textbf{Allocate} := {} & *\text{port}?[](\nu\ in)(\nu\ out)(\nu\ query)\\ & (in![query]\\ & \mid in?[response].(out![response] \mid \text{port}![]))\end{aligned}$$

Each message port![] occurring at top level symbolizes an available connection. When a customer requests an available connection, three channel names *query*, *in* and *out* are created. The customer sends its query, represented by the channel *query*, via the channel *in*. The server computes this query and sends it back via the channel *out*. A new message port![] is then spawn, which symbolizes that the connection is released. We shall notice that many computational aspects are abstracted away, since we present only an approximation of a realistic server. We finally propose a short computation for $\mathcal{S}$ as follows:

$$\begin{aligned}\mathcal{S} \;=\; & (\nu\ \text{port})\\ & \quad (\textbf{Allocate} \mid \text{port}![] \mid \text{port}![] \mid \text{port}![])\\ \rightarrow\; & (\nu\ \text{port})(\nu\ in_1)(\nu\ out_1)(\nu\ query_1)\\ & \quad (\ \textbf{Allocate} \mid \text{port}![] \mid \text{port}![]\\ & \quad \mid in_1![query_1]\\ & \quad \mid in_1?[response].(out_1![response] \mid \text{port}![]))\\ \rightarrow\; & (\nu\ \text{port})(\nu\ in_1)(\nu\ out_1)(\nu\ query_1)\\ & \quad (\ \textbf{Allocate} \mid \text{port}![] \mid \text{port}![] \mid \text{port}![]\\ & \quad \mid out_1![query_1])\end{aligned}$$

□

As illustrated in the above example, the configuration of a mobile system is at any stage congruent with a configuration of the form $(\nu\ c)(\mathrm{P}_1 \mid ... \mid \mathrm{P}_n)$, where c is a sequence of channel names, and P_1, ..., P_n are syntactic copies of sub-processes which have been substituted during communications. Nevertheless, standard semantics allows to trace neither the origin of those processes, nor the origin of the channels they have declared.

4 Non-standard semantics

A non-standard semantics [23] is a refined semantics, which explicitly specifies the link between channels and the instances of processes which have declared them. It identifies any instance of a process by an unambiguous marker

$$
\begin{array}{rcll}
P & ::= & \text{action}.P & \text{(Action)} \\
 & | & (P \mid P) & \text{(Parallel composition)} \\
 & | & \emptyset & \text{(End of a process)} \\
 & & & \\
\text{action} & ::= & c![x_1, ..., x_n] & \text{(Message)} \\
 & | & c?[x_1, ..., x_n] & \text{(Input guard)} \\
 & | & *c?[x_1, ..., x_n] & \text{(Replication guard)} \\
 & | & (\nu\ x) & \text{(Channel creation)}
\end{array}
$$

where c, x_1, ..., x_n, $x \in \mathit{Channel}$, $n \geqslant 0$. Input guard, replication guard and channel creation are the only name binders, i.e in $c?[x_1, ..., x_n]P$, $*d?[y_1, ..., y_p]Q$ and $(\nu\ x)R$, occurrences of x_1, ..., x_n in P, y_1, ..., y_p in Q and x in R are considered bound. Usual rules about scoping, substitution and α-conversion apply. We denote by $\mathcal{FN}(P)$ the set of free names of P, i.e names which are not under the scope of a binder and by $\mathcal{BN}(P)$ the set of bound names of P.

(a) Syntax

$$
\begin{array}{rcll}
(\nu\ x)P & \equiv & (\nu\ y)P[x \leftarrow y] \quad \text{if } y \notin \mathcal{FN}(P) & (\alpha\text{-conversion}) \\
P \mid Q & \equiv & Q \mid P & \text{(Commutativity)} \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & \text{(Associativity)} \\
P \mid \emptyset & \equiv & P & \text{(End of a process)} \\
(\nu\ x)(\nu\ y)P & \equiv & (\nu\ y)(\nu\ x)P & \text{(Swapping)} \\
((\nu\ x)P) \mid Q & \equiv & (\nu\ x)(P \mid Q) \quad \text{if } x \notin \mathcal{FN}(Q) & \text{(Extrusion)}
\end{array}
$$

where x, $y \in \mathit{Channel}$

(b) Congruence relation

$$c![x_1, ..., x_n]P \mid c?[y_1, ..., y_n]Q \rightarrow P \mid \widetilde{Q} \qquad \text{(communication)}$$

$$c![x_1, ..., x_n]P \mid *c?[y_1, ..., y_n]Q \rightarrow P \mid \widetilde{Q} \mid *c?[y_1, ..., y_n]Q \qquad \text{(res. fetching)}$$

$$\frac{P \rightarrow Q}{(\nu\ x)P \rightarrow (\nu\ x)Q} \qquad \frac{P' \equiv P \quad P \rightarrow Q \quad Q \equiv Q'}{P' \rightarrow Q'} \qquad \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}$$

where c, x, x_1, ..., x_n, y_1, ..., $y_n \in \mathit{Channel}$
and $\widetilde{Q} = Q[y_1 \leftarrow x_1, ..., y_n \leftarrow x_n]$

(c) Reduction relation

Fig. 1. The chemical semantics

in order to distinguish each instance of a recursive process from all others. Then, the origin of channel names is easily traced by identifying each channel name with the marker of the process which has created it. In [14] we propose a non-standard semantics which considers the full π-calculus without non-deterministic choices. We redefine it in order to take them into account efficiently, without adding further reduction rules nor considering useless syntactic components. This allows for an easier analysis by making abstract domains smaller while giving a good intuition on the potential evolutions of the analyzed systems. For that purpose, we restrict the set of computations to those where non-deterministic choices are made as soon as possible. In such computations, no further communication nor resource fetching is performed while there are non-deterministic choices at the top level. This assumption does not change the subset of reachable standard configurations which contain no non-deterministic choice at top level.

Let Lbl be an infinite set of labels, we denote by $\mathcal{M}$ the set of all binary trees the leaves of which are not labelled (ε) and the nodes of which are labelled with a pair (i, j) where both i and j are in Lbl. The tree, having a node labelled a, a left sibling t_1 and a right sibling t_2 is denoted by $N(a, t_1, t_2)$. We use $\mathcal{M}$ as a set of markers and denote by Σ the set $Lbl \times Lbl$. Σ is used in labelling non-standard transitions. We consider a closed mobile system $\mathcal{S}$ in the π-calculus and assume without any loss of generality that two channel binders of $\mathcal{S}$ are never used on the same channel name. We locate syntactic components of $\mathcal{S}$ by marking each sign ? or ! occurring in $\mathcal{S}$ with distinct labels in Lbl. A non-standard configuration is a set of thread instances, where a thread instance is a triplet composed with a syntactic component, a marker and an environment. The syntactic component is a copy of a sub-process of $\mathcal{S}$, the marker is calculated at the creation of the thread and the environment specifies the semantic value of each free name in the syntactic component. Thread instances are created at the beginning of the system computation and during execution. In both cases, several threads are spawned, corresponding to a set of syntactic components, in accordance to which non-deterministic choices are made. Applying the function $Agent$, defined as follows, to either $\mathcal{S}$ for initial threads, or to the continuation of running processes, gives the set of all possibilities for the set of spawned syntactic components.

$$
\begin{aligned}
Agent(\emptyset) &= \{\{\}\} \\
Agent(x!^i[x_1, ..., x_n]P) &= \{\{x!^i[x_1, ..., x_n]P\}\} \\
Agent(y?^i[y_1, ..., y_n]P) &= \{\{y?^i[y_1, ..., y_n]P\}\} \\
Agent(*y?^i[y_1, ..., y_n]P) &= \{\{*y?^i[y_1, ..., y_n]P\}\} \\
Agent(P \mid Q) &= \{A \cup B \mid A \in Agent(P),\ B \in Agent(Q)\} \\
Agent(P + Q) &= Agent(P) \cup Agent(Q) \\
Agent((\nu\ x)P) &= Agent(P)
\end{aligned}
$$

The markers of initial threads are ε, while the markers of new threads are calculated recursively from the marker of the threads whose computation has

led to their creation:

- when an execution does not involve fetching a resource, the marker of the computed thread is just passed to the threads in its continuation;
- when a resource is fetched, the markers of the new threads created from the continuation of the resource are $N((i,j), id_*, id_!)$, where (i,id_*,E_*) is the fetched resource thread and $(j,id_!,E_!)$ the message sender thread.

Environments map each free channel name of syntactic components to a pair (a,b) where a is a bound channel name of $\mathcal{S}$, and b is a marker. Intuitively, a refers to the binder $(\nu\ a)$ which has been used in declaring the channel, and b is the marker of the thread which has declared it. While threads are running, environments are calculated in order to mimic the standard semantics.

We denote by $\mathcal{C}$ the set of all non-standard configurations. Our non-standard semantics is given in Figure 2. The function $\mathcal{C}_0$ gives the set of possible initial configurations, while the relation $\longrightarrow_2$ defines non-standard computation steps. Each non-standard communication steps are labelled with a pair $(i,j) \in \Sigma$, where i is the label of the message receiver and j is the label of the message sender. There is a *bisimulation* between standard and non-standard semantics, provided that we restrict the set of standard computations to those where all non-deterministic choices are always made before communications and resource fetching. The proof relies on the fact that non-standard computations cannot yield conflicts between the markers of the threads.

5 Abstraction

The set of all possible non-standard configurations a system may take during a finite computation sequence is given by its collecting semantics [7] and can be expressed as the least fixpoint of the following $\cup$-complete endomorphism $\mathbb{F}$ on the complete lattice $\wp(\Sigma^* \times \mathcal{C})$:

$$\mathbb{F}(X) = \{(\varepsilon, C) \mid C \in \mathcal{C}_0(\mathcal{S})\} \cup \{(u.\lambda, C) \mid \exists (u, C') \in X,\ C' \xrightarrow{\lambda}_2 C\}$$

We use abstract interpretation [8] to design an abstract domain in which a decidable description of $\mathit{Coll}(\mathcal{S})$ will be computed. Our abstract domain is the reduced product of two domains: the first one describes the control-flow of $\mathcal{S}$; the second one counts occurrences of its processes. Since the π-calculus is a communication-based formalism, any further analysis requires a good approximation of the communication topology. Several analyses [2,22,23,14] have already been proposed. For the sake of simplicity, we use a naive uniform analysis to abstract the control-flow of systems, but it could be enriched by using the non-uniform analysis proposed in [14]. We introduce the set $\mathcal{BN}(\mathcal{S})^2$ as the set of all possible interactions between agents of the system $\mathcal{S}$; intuitively the pair (x,y) denotes that the channel name x may be bound to a channel created by the binder $(\nu\ y)$. Our first abstract domain is then the complete

$$\mathcal{C}_0(\mathcal{S}) = \{\{(p, \varepsilon, E_p) \mid p \in Cont)\} \mid Cont \in Agent(\mathcal{S})\}$$
$$\text{where } E_p = \begin{cases} \mathcal{FN}(p) & \to \mathcal{BN}(\mathcal{S}) \times \mathcal{M} \\ x & \mapsto (x, \varepsilon) \end{cases}$$

(a) Set of initial non-standard configurations

If C is a non-standard configuration,
if there are λ, μ in C,
with $\lambda{=}(y?^i[y_1, ..., y_n]P, id_?, E_?)$ and $\mu{=}(x!^j[x_1, ..., x_n]Q, id_!, E_!)$
such that $E_?(y) = E_!(x)$,
if $Cont_P$ is in $Agent(P)$ and $Cont_Q$ is in $Agent(Q)$,

then $C \xrightarrow{(i,j)}_2 C'$

where $C' = (C \setminus \{\lambda, \mu\}) \cup (f_?(Cont_P)) \cup (f_!(Cont_Q))$,

$$f_? : Ag \mapsto \left(Ag, id_?, \begin{cases} z \mapsto E_?(z) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{FN}(y?^i[y_1, ..., y_n]P) \\ y_k \mapsto E_!(x_k) & \text{if } y_k \in \mathcal{FN}(Ag) \\ z \mapsto (z, id_?) & \text{if } \begin{cases} z \in \mathcal{FN}(Ag) \cap \mathcal{BN}(y?^i[y_1, ..., y_n]P) \\ z \notin \{y_k | k \in [|1; n|]\} \end{cases} \end{cases} \right)$$

$$\text{and } f_! : Ag \mapsto \left(Ag, id_!, \begin{cases} z \mapsto E_!(z) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{FN}(x!^j[x_1, ..., x_n]Q) \\ z \mapsto (z, id_!) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{BN}(x!^j[x_1, ..., x_n]Q) \end{cases} \right)$$

(b) Non-standard communication

If C is a non-standard configuration,
if there are λ, μ in C,
with $\lambda{=}(*y?^i[y_1, ..., y_n]P, id_?, E_?)$ and $\mu = (x!^j[x_1, ..., x_n]Q, id_!, E_!)$
such that $E_?(y) = E_!(x)$,
if $Cont_P$ is in $Agent(P)$ and $Cont_Q$ is in $Agent(Q)$,

then $C \xrightarrow{(i,j)}_2 C'$

where $C' = (C \setminus \{\mu\}) \cup (f_?(Cont_P)) \cup (f_!(Cont_Q))$,
$id_* = N((i, j), id_?, id_!)$,

$$f_? : Ag \mapsto \left(Ag, id_*, \begin{cases} z \mapsto E_?(z) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{FN}(y?^i[y_1, ..., y_n]P) \\ y_k \mapsto E_!(x_k) & \text{if } y_k \in \mathcal{FN}(Ag) \\ z \mapsto (z, id_*) & \text{if } \begin{cases} z \in \mathcal{FN}(Ag) \cap \mathcal{BN}(y?^i[y_1, ..., y_n]P) \\ z \notin \{y_k | k \in [|1; n|]\} \end{cases} \end{cases} \right)$$

$$\text{and } f_! : Ag \mapsto \left(Ag, id_!, \begin{cases} z \mapsto E_!(z) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{FN}(x!^j[x_1, ..., x_n]Q) \\ z \mapsto (z, id_!) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{BN}(x!^j[x_1, ..., x_n]Q) \end{cases} \right)$$

(c) Non-standard resource fetching

Fig. 2. Non-standard semantics

lattice $\wp(\mathcal{BN}(\mathcal{S})^2)$ related to our concrete domain via a Galois connection $(\alpha_{\rm com},\gamma_{\rm com})$ defined as follows:

$$\alpha_{\rm com}(A) = \left\{ (x,y) \;\middle|\; \begin{array}{l} \exists (u,C) \in A,\ \exists (P,id_1,E) \in C, \exists id_2 \in \mathcal{M} \\ \text{such that } x \in \mathcal{FN}(P) \text{ and } E(x) = (y,id_2) \end{array} \right\}$$

$$\gamma_{\rm com}(A^\sharp) = \left\{ (u,C) \;\middle|\; \begin{array}{c} \forall (P,id_1,E) \in C,\ \forall x \in \mathcal{FN}(P),\ \forall y \in \mathcal{BN}(\mathcal{S}) \\ [\exists id_2,\ E(x) = (y,id_2)] \Longrightarrow (x,y) \in A^\sharp \end{array} \right\}$$

The second domain counts both the number of occurrences of processes and the number of performed transitions inside computations. We denote by Π the set of all sub-processes of $\mathcal{S}$. Let $\mathcal{V}$ be the set $\Pi + \Sigma$. We consider $\wp(\mathbb{N}^{\mathcal{V}})$, the complete lattice of the sets of natural number functions defined on $\mathcal{V}$. $\wp(\mathbb{N}^{\mathcal{V}})$ is related to our concrete domain via a Galois connection $(\alpha_{\mathbb{N}^{\mathcal{V}}}, \gamma_{\mathbb{N}^{\mathcal{V}}})$, defined as follows:

$$\alpha_{\mathbb{N}^{\mathcal{V}}}(A) = \left\{ \left\{ \begin{array}{ll} \mathcal{V} & \to \mathbb{N} \\ v \in \Pi & \mapsto \mathrm{Card}(\{(P,id,E) \in C \mid v = P\}) \\ \lambda \in \Sigma & \mapsto |u|_\lambda \end{array} \right. \;\middle|\; (u,C) \in A \right\}$$

$$\gamma_{\mathbb{N}^{\mathcal{V}}}(A^\sharp) = \left\{ (u,C) \;\middle|\; \begin{array}{l} \exists f \in A^\sharp, \forall \lambda \in \Sigma,\ |u|_\lambda = f(\lambda), \\ \forall v \in \Pi,\ \mathrm{Card}(\{(P,id,E) \in C \mid v = P\}) = f(v) \end{array} \right\}$$

The complete lattice, $(\mathcal{N}_{\mathcal{V}}, \sqsubseteq_{\mathcal{N}_{\mathcal{V}}}, \sqcup_{\mathcal{N}_{\mathcal{V}}}, \bot_{\mathcal{N}_{\mathcal{V}}}, \sqcap_{\mathcal{N}_{\mathcal{V}}}, \top_{\mathcal{N}_{\mathcal{V}}})$, left as a parameter of our abstraction, is related to $\wp(\mathbb{N}^{\mathcal{V}})$ by a Galois connection $(\alpha_{\mathcal{N}_{\mathcal{V}}}, \gamma_{\mathcal{N}_{\mathcal{V}}})$ which satisfies the condition: $\gamma_{\mathcal{N}_{\mathcal{V}}}(\bot_{\mathcal{N}_{\mathcal{V}}}) = \emptyset$. We require three abstract primitives which satisfy the following soundness hypotheses:

- *required* : $\wp(\Pi) \times \mathcal{N}_{\mathcal{V}} \to \mathcal{N}_{\mathcal{V}}$
$$\gamma_{\mathcal{N}_{\mathcal{V}}}(v^\sharp) \cap \{f \mid f(p) \geqslant 1,\ \forall p \in A\} \subseteq \gamma_{\mathcal{N}_{\mathcal{V}}}(\mathit{required}(A, v^\sharp))$$
- *trans* : $\mathcal{N}_{\mathcal{V}} \times \mathbb{N}^{\mathcal{V}} \to \mathcal{N}_{\mathcal{V}}$
$$\{x \mapsto f(x) + g(x) \mid g \in \gamma_{\mathcal{N}_{\mathcal{V}}}(v^\sharp)\} \subseteq \gamma_{\mathcal{N}_{\mathcal{V}}}(\mathit{trans}(v^\sharp, f))$$
- ρ : $\mathcal{N}_{\mathcal{V}} \to \mathcal{N}_{\mathcal{V}}$
$$\gamma_{\mathcal{N}_{\mathcal{V}}}(v^\sharp) \subseteq \gamma_{\mathcal{N}_{\mathcal{V}}}(\rho(v^\sharp))$$

Roughly speaking, ρ is a reduction, it maps each abstract value to another one which represents the same set but in which properties are easier to establish. Particularly, it will be used for proving that an abstract value represents the empty set. At each abstract computation step, *required* is used to extract from the abstract value the representation of configurations which simultaneously contain all the processes required by the computation step and *trans* is used to calculate the representation of the result of the computation step by taking into account newly spawned and destroyed processes.

We define our abstract domain $(\mathcal{C}^\sharp, \sqsubseteq, \sqcup, \bot, \sqcap, \top)$ as the complete lattice $(\wp(\mathcal{BN}(\mathcal{S})^2) \times \mathcal{N}_{\mathcal{V}})$, where $\sqsubseteq$, $\sqcup$, $\bot$, $\sqcap$ and $\top$ are defined pairwise. $\mathcal{C}^\sharp$ is related to $\wp(\Sigma^* \times \mathcal{C})$ by a Galois connection (α, γ), where $\alpha(A) = \alpha_{\rm com}(A), [\alpha_{\mathcal{N}_{\mathcal{V}}} \circ \alpha_{\mathbb{N}^{\mathcal{V}}}](A))$ and $\gamma(A,B) = \gamma_{\rm com}(A) \cap [\gamma_{\mathbb{N}^{\mathcal{V}}} \circ \gamma_{\mathcal{N}_{\mathcal{V}}}](B)$. Our abstract semantics is

defined by a transition relation $\rightsquigarrow$ on our abstract domain $\mathcal{C}^\sharp$, given in Figure 3. Soundness hypotheses on abstract primitives ensure the soundness of our abstract transition:

Proposition 5.1 *If* $(u, C) \in \gamma(C^\sharp)$ *and* $C \xrightarrow{\lambda}_2 \overline{C}$, *then there exists* $\overline{C}^\sharp$ *such that* $C^\sharp \overset{\lambda}{\rightsquigarrow} \overline{C}^\sharp$ *and* $(u.\lambda, \overline{C}) \in \gamma(\overline{C}^\sharp)$.

As a consequence, the abstract counterpart $\mathbb{F}^\sharp$ of $\mathbb{F}$, defined by

$$\mathbb{F}^\sharp(C^\sharp) = (\alpha(\{(\varepsilon, C) \mid C \in \mathcal{C}_0(\mathcal{S})\})) \sqcup C^\sharp \sqcup \left(\bigsqcup\{\overline{C^\sharp} \mid \exists \lambda \in \Sigma,\ C^\sharp \overset{\lambda}{\rightsquigarrow} \overline{C^\sharp}\}\right)$$

satisfies the soundness condition $\mathbb{F}(C) \subseteq \gamma(\mathbb{F}^\sharp(\alpha(C)))$. Using Kleene's theorem, we obtain the soundness of our analysis:

Theorem 5.2 $lfp_\emptyset \mathbb{F} \subseteq \bigcup_{n \in \mathbb{N}} [\gamma \circ \mathbb{F}^{\sharp\, n}](\bot)$

Following [6,7], we compute a sound approximation of our abstract semantics by using a widening operator $\nabla \ : \ \mathcal{C}^\sharp \times \mathcal{C}^\sharp \to \mathcal{C}^\sharp$ which satisfies the following properties:

- $\forall C_1^\sharp,\ C_2^\sharp \in \mathcal{C}^\sharp,\ C_1^\sharp \sqcup C_2^\sharp \sqsubseteq C_1^\sharp \nabla C_2^\sharp$
- $(C_n^\sharp) \in (\mathcal{C}^\sharp)^{\mathbb{N}}$, the sequence (C_n^∇) defined as

$$\begin{cases} C_0^\nabla = C_0^\sharp \\ C_{n+1}^\nabla = C_n^\nabla \nabla C_{n+1}^\sharp \end{cases}$$

is ultimately stationary.

The abstract iteration of $\mathbb{F}^\sharp$ is then defined as follows:

$$\begin{cases} \mathbb{F}_0^\nabla = \bot \\ \mathbb{F}_{n+1}^\nabla = \begin{cases} \mathbb{F}_n^\nabla & \text{if } \mathbb{F}^\sharp(\mathbb{F}_n^\nabla) \sqsubseteq \mathbb{F}_n^\nabla \\ \mathbb{F}_n^\nabla \nabla \mathbb{F}^\sharp(\mathbb{F}_n^\nabla) & \text{else} \end{cases} \end{cases}$$

Theorem 5.3 ***Abstract iteration[10,11]*** *Abstract iteration* $(\mathbb{F}_n^\nabla)$ *is ultimately stationary and its limit* $\mathbb{F}^\nabla$ *satisfies* $Coll(\mathcal{S}) \subseteq \gamma(\rho(\mathbb{F}^\nabla))$.

6 Detecting exhaustion of resources and mutual exclusion

We only need to define an abstract domain to approximate set of tuples of natural numbers, in which abstract primitives can be precisely and efficiently implemented. We reject the use of usual numerical domains: we are unlikely to design a precise primitive *required* in non-relational domain, without using an exponential partitioning; we think that the domain of linear inequalities among

Let $(c^\sharp, v^\sharp) \in \mathcal{C}^\sharp$, $u \in \mathcal{BN}(\mathcal{S})$, $y?^i[y_1, ..., y_n]P$ and $x!^j[x_1, ..., x_n]Q$ two subprocesses, $\text{Cont}_? \in \mathit{Agent}(P)$, $\text{Cont}_! \in \mathit{Agent}(Q)$, such that:

- $(y, u) \in c^\sharp$
- $(x, u) \in c^\sharp$
- $V \triangleq \rho(\mathit{required}(\{y?^i[y_1, ..., y_n]P; x!^j[x_1, ..., x_n]Q\}, v^\sharp)) \neq \bot_{\mathcal{N}_\mathcal{V}}$

then $(c^\sharp, v^\sharp) \overset{(i,j)}{\rightsquigarrow} (c'^\sharp, v'^\sharp)$, where

- $c'^\sharp = c^\sharp \cup \{(y_k, t) \mid k \in [|1; n|],\ t \in \mathcal{BN}(\mathcal{S}) \text{ and } (x_k, t) \in c^\sharp\}$
  $\cup \{(x, x) \mid \exists p \in \text{Cont}_?,\ x \in (\mathcal{BN}(P) \cap \mathcal{FN}(p)) \setminus \{y_k \mid k \in [|1; n|]\}\}$
  $\cup \{(x, x) \mid \exists q \in \text{Cont}_!,\ x \in \mathcal{BN}(Q) \cap \mathcal{FN}(q)\}$
- $v'^\sharp = \mathit{trans}(V, \delta)$

with

$$\forall x \in \Pi,\ \delta(x) = \begin{cases} -1 & \text{if } x \in \{y?^i[y_1, ..., y_n]P, x!^j[x_1, ..., x_n]Q\} \setminus (\mathit{cont}_? \cup \mathit{cont}_!) \\ +1 & \text{if } x \in (\mathit{cont}_? \cup \mathit{cont}_!) \setminus \{y?^i[y_1, ..., y_n]P, x!^j[x_1, ..., x_n]Q\} \\ 0 & \text{otherwise} \end{cases}$$

$$\forall \lambda \in \Sigma,\ \delta(\lambda) = \begin{cases} +1 & \text{if } \lambda = (i, j) \\ 0 & \text{otherwise} \end{cases}$$

(a) Abstract communication

Let $(c^\sharp, v^\sharp) \in \mathcal{C}^\sharp$, $u \in \mathcal{BN}(\mathcal{S})$, $*y?^i[y_1, ..., y_n]P$ and $x!^j[x_1, ..., x_n]Q$ two subprocesses, $\text{Cont}_* \in \mathit{Agent}(P)$, $\text{Cont}_! \in \mathit{Agent}(Q)$, such that:

- $(y, u) \in c^\sharp$
- $(x, u) \in c^\sharp$
- $V \triangleq \rho(\mathit{required}(\{*y?^i[y_1, ..., y_n]P; x!^j[x_1, ..., x_n]Q\}, v^\sharp)) \neq \bot_{\mathcal{N}_\mathcal{V}}$

then $(c^\sharp, v^\sharp) \overset{(i,j)}{\rightsquigarrow} (c'^\sharp, v'^\sharp)$, where

- $c'^\sharp = c^\sharp \cup \{(y_k, t) \mid k \in [|1; n|],\ t \in \mathcal{BN}(\mathcal{S}) \text{ and } (x_k, t) \in c^\sharp\}$
  $\cup \{(x, x) \mid \exists p \in \text{Cont}_*,\ x \in (\mathcal{BN}(P) \cap \mathcal{FN}(p)) \setminus \{y_k \mid k \in [|1; n|]\}\}$
  $\cup \{(x, x) \mid \exists q \in \text{Cont}_!,\ x \in \mathcal{BN}(Q) \cap \mathcal{FN}(q)\}$
- $v'^\sharp = \mathit{trans}(V, \delta)$

$$\text{with } \forall x \in \Pi,\ \delta(x) = \begin{cases} -1 & \text{if } x \in \{x!^j[x_1, ..., x_n]Q\} \setminus \mathit{cont}_* \\ +1 & \text{if } x \in (\mathit{cont}_* \cup \mathit{cont}_!) \setminus \{x!^j[x_1, ..., x_n]Q\} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{and } \forall \lambda \in \Sigma,\ \delta(\lambda) = \begin{cases} +1 & \text{if } \lambda = (i, j) \\ 0 & \text{otherwise} \end{cases}$$

(b) Abstract resource fetching

Fig. 3. Abstract transition relation

a finite set of variables [12] is too expensive because we deal with too many variables. We propose the use of a product of two domains. The first domain is based on the use of the interval lattice and is used for expressing properties of interest. This domain can represent all the information we need to express non-exhaustion of resources, but it cannot calculate them precisely without being refined. The second domain is based on the use of linear equalities between variables [17] and is used for expressing more complex properties, such as mutual exclusion for instance, which allows for more precise calculations in the first domain. The power of our analysis directly follows from an unexpensive algorithm, straightforwardly adapted from Linear Constraint Programming, to calculate an approximated reduction between these two domains.

The complete lattice $(\mathcal{I}_\mathcal{V}, \sqsubseteq_{\mathcal{I}_\mathcal{V}}, \sqcup_{\mathcal{I}_\mathcal{V}}, \bot_{\mathcal{I}_\mathcal{V}}, \sqcap_{\mathcal{I}_\mathcal{V}}, \top_{\mathcal{I}_\mathcal{V}})$ is the functional domain of the natural numbers intervals, where lattice operations are defined pointwise. A family $(\nabla^n_{\mathcal{I}_\mathcal{V}})$ of widening operators on $\mathcal{I}_\mathcal{V}$ is defined as follows:

$$[f \nabla^n_{\mathcal{I}_\mathcal{V}} g](x) = f(x) \nabla^n g(x)$$

$$\text{where } \begin{cases} [|a;b|] \ \nabla^n \ [|c;d|] = [|min\{a;c\};\infty|[\text{ if } d > max\{b;n\} \\ I \quad \nabla^n \quad J \quad = \quad I \cup J \qquad \text{otherwise} \end{cases}$$

The complete lattice $(\mathcal{K}_\mathcal{V}, \sqsubseteq_{\mathcal{K}_\mathcal{V}}, \cup_{\mathcal{K}_\mathcal{V}}, \top_{\mathcal{K}_\mathcal{V}}, \cap_{\mathcal{K}_\mathcal{V}}, \bot_{\mathcal{K}_\mathcal{V}})$ of linear equality systems between the finite set of variables $\mathcal{V}$ is described with its lattice operations in [17]. This domain uses Gauss reduction in order to normalize systems. Moreover, since there are no infinite increasing chain [17], we can choose $\cup_{\mathcal{K}_\mathcal{V}}$ as a widening operator. Our numerical domain is then the product $\mathcal{I}_\mathcal{V} \times \mathcal{K}_\mathcal{V}$. Generic primitives are expressed as follows:

- $\mathit{required}(P,(i,s)) = (i',s)$ where $\begin{cases} i'(x) = i(x) \cap [|1;+\infty|[& \forall x \in P \\ i'(x) = i(x) & \forall x \in \Pi \setminus P \end{cases}$
- $\mathit{trans}\left(\left(i, \left\{\sum_{v\in\mathcal{V}} a^k_v v = b_k, \ \forall k \in [|1;m|]\right\}\right), f\right) = (i',s')$

$$\text{where } \begin{cases} i'(x) & = \{k + f(x) | \forall k \in i(x)\} \cap [|0;+\infty|[\\ s' & = \left\{\sum_{v\in\mathcal{V}} a^k_v (v - f(v)) = b_k, \ \forall k \in [|1;m|]\right. \end{cases}$$

We now present a reduction [9] ρ between $\mathcal{I}_\mathcal{V}$ and $\mathcal{K}_\mathcal{V}$. A reduction consists in taking into account linear constraints in order to narrow the domain of interval variables. For instance, the system of constraints $\{x + y = 12, \ x \in [|3;15|], \ y \in [|4;19|]$ can be reduced to the system $\{x+y = 12, \ x \in [|3;8|], \ y \in [|4;9|]$. Linear constraints are likely to be combined, via Gauss reduction, in order to give new linear constraints which will allow for further reductions. Therefore, generating the whole set of such combinations is likely to require an exponential time of execution.

We propose a two-step-polynomial algorithm for solving this problem. The first step aims at narrowing infinite intervals into finite ones. It uses Gauss

reduction to obtain a positive representation of systems of linear equalities, that is to say an equivalent system of equations such that if a variable occurs with a strictly negative coefficient in an equation, then this variable occurs with a negative coefficient in each equation. Positive representations contain only a few undefined forms, which allows to narrow infinite intervals into finite ones, with a worst-case in $\mathcal{O}(n^3)$. The second step is inspired by [5]: it consists in obtaining a triangular system of constraints of the form $a_1.x_1 + ... + a_n.x_n \in I$ where I is an interval. This system is then used for propagating unidirectionnally intervals from non-diagonal to diagonal variables. The result is a good reduction with a worst-case in $\mathcal{O}(n^4)$.

We now propose some examples of mobile systems analyzed with our prototype. For the sake of brevity, we have selected for each example a subset of significant constraints captured by our analysis; full results are available in [13]. In these constraints, the number of occurrences of a process $c?^i[x_1, ..., x_n]P$ or $c!^i[x_1, ..., x_n]P$ is denoted by $\sharp(i)$, while $\underline{\sharp}(i, j)$ denotes the number of times a communication reduction labelled with (i, j) is used.

Example 6.1 Our first example is the *ftp* protocol proposed in Example 3.1:

$$\mathcal{S} := (\nu\ \text{port})(\textbf{Allocate} \mid \text{port}!^5[] \mid \text{port}!^6[] \mid \text{port}!^7[])$$

where

$$\begin{aligned}\textbf{Allocate} := {} & *\text{port}?^0[](\nu\ in)(\nu\ out)(\nu\ query)\\ & (in!^1[query]\\ & \mid in?^2[response].(out!^3[response] \mid \text{port}!^4[]))\end{aligned}$$

$$\begin{cases}\sharp(0) = 1,\ \sharp(i) \in [|0;3|],\ \forall i \in \{1;2;4\}\\ \sharp(3) \in [|0;\infty|[\\ \sharp(i) \in [|0;1|],\ \forall i \in [|5;6;7|]\\ \sharp(1) + \sharp(4) + \sharp(5) + \sharp(6) + \sharp(7) = 3\\ \sharp(3) = \underline{\pi}(2,1)\end{cases}$$

Our analysis has proved that only three physical channels are required to simulate this protocol. □

Example 6.2 We now propose an example of mutual exclusion:

$$\mathcal{S} := \left(\begin{array}{l}(\nu\ \text{a})(\nu\ \text{b})(\nu\ \text{c})(\nu\ \text{d})\\ \qquad (*\text{a}?^0[x](x!^1[\text{a}] + (\text{c}?^2[]\text{d}!^3[]))\\ \qquad |*\text{b}?^4[x](x!^5[\text{b}] + (\text{c}!^6[]))\\ \qquad |\text{a}!^7[\text{b}])\end{array}\right)$$

$$\begin{cases}\sharp(i) \in [|0;1|],\ \forall i \in \{1;2;5;6;7\}\\ \sharp(3) = 0\\ \sharp(1) + \sharp(2) + \sharp(5) + \sharp(6) + \sharp(7) = 1\end{cases}$$

Our analysis has proved that the sub-process $(\text{d}!^3[])$ is unreachable by detecting

a mutual exclusion between the sub-processes (c?^{2}[]d!^{3}[]) and (c!^{6}[]). □

7 Detecting deadlocks

We use both our abstraction of the collecting semantics and dynamic partitioning tools [3] to solve the problem of detecting sets of non-standard configurations such that no infinite sequence of computations can pass by them. For that purpose, we analyze the trace semantics of the mobile system $\mathcal{S}$, which is defined as a transition system $(G,\ A \subseteq G \times G)$ where $G = \mathit{Coll}(\mathcal{S})$ is the set of states and $A = \{(u, C), \lambda, (u.\lambda, C') \mid (u, C) \in \mathit{Coll}(\mathcal{S}),\ C \xrightarrow{\lambda}_2 C'\}$ is the set of transitions. This is also the least fixpoint of an $\cup$-complete endomorphism $\mathbb{F}$ on the complete lattice $(\mathcal{T}, \sqsubset_{\mathcal{T}}, \cup_{\mathcal{T}}, \bot_{\mathcal{T}}, \cap_{\mathcal{T}}, \top_{\mathcal{T}})$ of transition systems on the alphabet Σ, the set of states of which is a subset of the set $(\Sigma^* \times \mathcal{C})$. Lattice operations are the usual set operations, while $\mathbb{F}((G, A))$ is the transition system (G', A') defined as follows:

$$\begin{cases} G' = \{(\varepsilon, C) \mid C \in \mathcal{C}_0(\mathcal{S})\} \cup \{(u.\lambda, C) \mid \exists (u, C') \in G,\ C' \xrightarrow{\lambda}_2 C\} \\ A' = \{((u, C), \lambda, (u.\lambda, C')) \mid (u, C) \in G,\ C \xrightarrow{\lambda}_2 C'\} \end{cases}$$

We abstract the trace semantics to compute a finite approximation in a finite time. We approximate infinite transition systems by finitely partitioning their set of states. An abstract transition system is then defined as a transition system on a finite set of states, and a function mapping each state to the set of concrete states it represents. Let $\mathbb{P}$ be a finite subset of the lattice $\wp(\mathbb{N}^{\mathcal{V}})$, such that $\emptyset \notin \mathbb{P}$, $\bigcup \mathbb{P} = \mathbb{N}^{\mathcal{V}}$ and $\forall a, a' \in \mathbb{P},\ a \neq a' \implies a \cap a' = \emptyset$. We introduce our abstract domain $\mathcal{T}^\sharp$ as the set of all pairs $((G^\sharp, A^\sharp), f)$ such that $(G^\sharp, A^\sharp)$ is a transition system on the alphabet Σ, where $G^\sharp$ is the quotient of $\mathbb{P}$ by an equivalence relation and f is a function mapping each element of $G^\sharp$ to an abstract element of $\mathcal{C}^\sharp$. $\mathcal{T}^\sharp$ is partially pre-ordered by the relation $\sqsubseteq_{\mathcal{T}^\sharp}$ where $((\mathbb{P}_{/\sim_1}, A_1^\sharp), f_1) \sqsubseteq_{\mathcal{T}^\sharp} ((\mathbb{P}_{/\sim_2}, A_2^\sharp), f_2)$ if and only:

$$\begin{cases} \forall q^\sharp, q'^\sharp \in \mathbb{P},\ q^\sharp \sim_1 q'^\sharp \implies q^\sharp \sim_2 q'^\sharp \\ \forall q^\sharp \in \mathbb{P},\ f_1([q^\sharp]_{\sim_1}) \sqsubseteq f_2([q^\sharp]_{\sim_2}) \\ \forall q^\sharp, q'^\sharp \in \mathbb{P},\ \forall \lambda \in \Sigma,\ ([q^\sharp]_{\sim_1}, \lambda, [q'^\sharp]_{\sim_1}) \in {A_1}^\sharp \implies ([q^\sharp]_{\sim_2}, \lambda, [q'^\sharp]_{\sim_2}) \in {A_2}^\sharp \end{cases}$$

Since there is no canonical choice for the equivalence relation, we are unlikely to define an abstraction function. So, we use a relaxed version of abstract interpretation [10], which only relies on the use of a concretization function

$\gamma_{\mathcal{T}}$ which relates the abstract domain to the concrete one, defined as follows:

$$\gamma_{\mathcal{T}}((\mathbb{P}_{/\sim}, A^\sharp), f) = (G, A)$$

$$\text{where } \begin{cases} G = \bigcup\limits_{q^\sharp \in \mathbb{P}} [(\gamma(f([q^\sharp]_\sim))) \cap (\gamma_{\mathbb{N}^\nu}(q^\sharp))] \\ A = \left\{ (q, \lambda, q') \;\middle|\; \exists q^\sharp,\ q'^\sharp \in \mathbb{P},\ \begin{cases} q \in (\gamma(f([q^\sharp]_\sim))) \cap (\gamma_{\mathbb{N}^\nu}(q^\sharp)) \\ q' \in (\gamma(f([q'^\sharp]_\sim))) \cap (\gamma_{\mathbb{N}^\nu}(q'^\sharp)) \\ ([q^\sharp]_\sim, \lambda, [q'^\sharp]_\sim) \in A^\sharp \\ q \xrightarrow{\lambda}_2 q' \end{cases} \right\} \end{cases}$$

We give in Figure 4 the definition of both an abstract counterpart $\mathbb{F}^\sharp$ of $\mathbb{F}$ and an accelerator of convergence $\mathbb{G}^\sharp$. Intuitively, $\mathbb{G}^\sharp$ merges the states of the abstract transition system, as soon as we are unable to prove that no infinite derivation can pass by a state they represent. $\mathbb{G}^\sharp$ uses a generic primitive relation, denoted by *finite* $\in \wp(\mathcal{T}^\sharp \times \mathbb{P})$, such that for all $t^\sharp = ((\mathbb{P}_{/\sim}, A^\sharp), f)$ in $\mathcal{T}^\sharp$, for all $q_0^\sharp$ in $\mathbb{P}$, if $(t^\sharp, q_0^\sharp)$ in *finite* then all derivations in transition system $\gamma_{\mathcal{T}}(t^\sharp)$ passing by a configuration c_0 in $\gamma(f([q_0^\sharp]_\sim)) \cap \gamma_{\mathbb{N}^\nu}(q_0{}^\sharp)$ are finite. $\mathbb{F}^\sharp$ and $\mathbb{G}^\sharp$ satisfy the soundness property: for all $t^\sharp$ in $\mathcal{T}^\sharp$, $[\mathbb{F} \circ \gamma_{\mathcal{T}}](t^\sharp) \sqsubseteq_{\mathcal{T}} [\gamma_{\mathcal{T}} \circ \mathbb{F}^\sharp \circ \mathbb{G}^\sharp](t^\sharp)$. Furthermore, the sequence $[\mathbb{F}^\sharp \circ \mathbb{G}^\sharp]^n((\mathbb{P}, \emptyset), \emptyset)$ is ultimately stationary and, thanks to Theorem 4.1.1.0.2 in [6], its limit $l^\sharp$ satisfies $lfp_{\perp_{\mathcal{T}}} \mathbb{F} \sqsubseteq \gamma_{\mathcal{T}}(l^\sharp)$.

Theorem 7.1 *Let $((\mathbb{P}_{/\sim}, A), f) = l^\sharp$. For all q in $\mathbb{P}$, for all C in $\gamma(f([q^\sharp]_\sim)) \cap (\gamma_{\mathbb{N}^\nu}(q^\sharp))$, there is no infinite computation sequence in the system $\mathcal{S}$ which passes by the state C if $(l^\sharp,\ q) \in$ finite.*

Our last task is to instantiate $\mathbb{P}$ and the primitive *finite*. A good choice for $\mathbb{P}$ consists in partitioning the set $\mathbb{N}^{\mathcal{V}}$ in accordance to the values of the variables of $\mathcal{V}$ which have a bounded behavior. For that purpose, we consider $v \subseteq V$ and a family (M_x) in $\mathbb{N}^v$, such that $\forall f \in \alpha_{\mathbb{N}^\nu}(Coll(\mathcal{S}))$, $f(x) \in [|0; M_x|]$. We take $\mathbb{P} = \mathbb{N}^{\mathcal{V}}{}_{/\sim}$ where $\sim$ is defined by: $g \sim h$ if and only if $(\forall x \in v,\ g(x) \in [|0; M_x|]$ *and* $g(x) = h(x))$ *or* $(\exists x_1, x_2 \in v, g(x_1) > 1$ *and* $h(x_2) > 1)$. Both v and (M_x) are given by the analysis presented in Section 5. The primitive *finite* is given by the following algorithm. Let $(t^\sharp, f) \in \mathcal{T}^\sharp$ and $q_0^\sharp \in \mathbb{P}$, we introduce *Available* $\subseteq \Sigma$ defined by $\lambda \in$ *Available* if and only if there is a derivation in the transition system $t^\sharp$ stemming from $q_0^\sharp$ and containing a transition labelled with λ. The following soundness proposition is valid:

Proposition 7.2 *$\forall \lambda \in \Sigma$, if there is a derivation in the transition system $\gamma_{\mathcal{T}}(t^\sharp, f)$ which stems from a configuration in $\gamma(f([q_0^\sharp]_\sim)) \cap (\gamma_{\mathbb{N}^\nu}(q_0^\sharp))$ and which contains a transition labelled with λ, then $\lambda \in$ Available.*

We denote by $\mapsto_2$ the subset of $\longrightarrow_2$ which only contains the computation steps labelled with an elements of *Available*. Following [16], we try to build a well-founded set in which we will interpret $(\mathcal{C}, \mapsto_2)$ via a morphism. Success in

$$\mathbb{F}^\sharp_{\mathcal{P}}((\mathbb{P}_{/\sim}, A), f) = ((\mathbb{P}_{/\sim}, A'), f') \text{ where}$$

$$\begin{cases} f'(x) = f(x)\nabla h(x) \\ h(x) = \bigsqcup \left\{ (c^\sharp, v^\sharp \sqcap_{\mathcal{N}_\mathcal{V}} \alpha_{\mathcal{N}_\mathcal{V}}(u')) \;\middle|\; \begin{array}{l} u' \in x,\ v^\sharp \sqcap_{\mathcal{N}_\mathcal{V}} \alpha_{\mathcal{N}_\mathcal{V}}(u') \neq \bot_{\mathcal{N}_\mathcal{V}}, \\ \begin{cases} (c^\sharp, v^\sharp) = \alpha(\{(\varepsilon, C) \mid C \in \mathcal{C}_0(\mathcal{S})\}) \\ \text{or } \exists u \in \mathbb{P},\ \exists \lambda \in \Sigma \text{ such that} \\ \quad f([u]_\sim) \sqcap (\top_{\text{com}}, \alpha_{\mathcal{N}_\mathcal{V}}(u)) \overset{\lambda}{\rightsquigarrow} (c^\sharp, v^\sharp) \end{cases} \end{array} \right\} \\ A' = A \cup \left\{ ([u]_\sim, \lambda, [u']_\sim) \;\middle|\; \begin{array}{l} \exists (c^\sharp, v^\sharp) \in \mathcal{C}^\sharp, \\ f([u]_\sim) \sqcap (\top_{\text{com}}, \alpha_{\mathcal{N}_\mathcal{V}}(u)) \overset{\lambda}{\rightsquigarrow} (c^\sharp, v^\sharp) \\ v^\sharp \sqcap_{\mathcal{N}_\mathcal{V}} \alpha_{\mathcal{N}_\mathcal{V}}(u') \neq \bot_{\mathcal{N}_\mathcal{V}} \end{array} \right\} \end{cases}$$

(a) counterpart function

$$\mathbb{G}^\sharp_{\mathcal{P}}((\mathbb{P}_{/\sim}, A), f) = ((\mathbb{P}_{/\sim'}, A'), f') \text{ where}$$

$$\begin{cases} a \sim' b \iff a \sim b \text{ or } \{(((\mathbb{P}_{/\sim}, A), f), a); (((\mathbb{P}_{/\sim}, A), f), b)\} \cap \mathit{finite} = \emptyset \\ f'([a]_{\sim'}) = \bigsqcup_{x \in [a]_{\sim'}} f([x]_\sim) \\ A' = \{([a_0]_{\sim'}, \lambda, [a]_{\sim'}) \mid ([a_0]_\sim, \lambda, [a]_\sim) \in A\} \end{cases}$$

(b) quotient function

Fig. 4. Abstract trace semantics

doing this will prove that any derivation in the system $\gamma_\mathcal{T}(t^\sharp, f)$ which passes by a configuration in $\gamma(f([q_0^\sharp]_\sim)) \cap (\gamma_{\mathbb{N}^\mathcal{V}}(q_0^\sharp))$ is finite.

We introduce the relation $\curvearrowright$ between the subprocesses of $\mathcal{S}$ such that $\forall p,\ q \in \Pi,\ p \curvearrowright q$ if and only if at least one of the following conditions is satisfied:

- p has the particular form $x!^j[]Q$, $q \in \bigcup \mathit{Agent}(Q)$, and there exists $i \in \mathit{Lbl}$ such that $(i, j) \in \mathit{Available}$;
- p has the particular form $x?^i[]P$, $q \in \bigcup \mathit{Agent}(P)$, and there exists $j \in \mathit{Lbl}$ such that $(i, j) \in \mathit{Available}$;
- p has the particular form $x!^j[]Q$, and there exists a process of the particular form $*y?^i[]P$ such that $q \in \bigcup \mathit{Agent}(P)$ and $(i, j) \in \mathit{Available}$.

We define the relation *finite* by: $((t^\sharp, f), q_0^\sharp) \in \mathit{finite} \iff (\Pi, \curvearrowright)$ is acyclic. Soundness hypotheses are satisfied since if $(\Pi, \curvearrowright)$ is acyclic, then $(\Pi, \curvearrowright^+)$ is a well founded order and the multiset extension [16] of $\curvearrowright^+$ gives a well founded order $\curvearrowright^+_{Mul}$ on $\mathbb{N}^\Pi$. The function $\Phi : (\mathcal{C}, \mapsto_2) \to (\mathbb{N}^\Pi, \curvearrowright^+_{Mul})$ defined by $\Phi(C)(p) = \mathit{Card}(\{(P, id, E) \in C \mid P = p\})$ is then a morphism.

Example 7.3 We propose to analyze the following mobile system:

$$\mathcal{S} := (\nu\ \text{push})(\nu\ \text{pop}) \\ ((*\text{push}?^1[](\text{pop}!^2[] \mid \text{push}!^3[])) \mid *\ \text{pop}?^4[] \mid *\ \text{push}?^5[] \mid \text{push}!^6[])$$

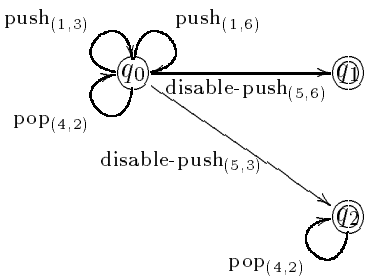
This system describes the behavior of a stack. The height of the stack is symbolized by the number of processes pop$!^2[]$ that occurs at the top level. Push (resp. pop) operations are symbolized by communication with the resource $*\text{push}?^1[]$ (resp. $*\text{pop}?^4[]$). Finally, a communication with the resource $*\text{push}?^5[]$ symbolizes that further push operations are no longer allowed.

Occurrence counting analysis gives the following properties:

$$\begin{cases} \pi(1) = 1,\ \pi(2) \in [|0; +\infty|[,\ \pi(3) \in [|0; 1|], \\ \pi(4) = 1,\ \pi(5) = 1,\ \pi(6) \in [|0; 1|], \\ \underline{\pi}(1,6) \in [|0; 1|],\ \underline{\pi}(5,3) \in [|0; 1|],\ \underline{\pi}(5,6) \in [|0; 1|], \\ \underline{\pi}(1,3) \in [|0; \infty|[,\ \underline{\pi}(4,2) \in [|0; \infty|[. \end{cases}$$

We denote by $v \subseteq \mathcal{V}$ the set of variables $\{\text{push}!^3[]; \text{push}!^6[]; (1,6); (5,3); (5,6)\}$ and we take $\mathbb{P} = \mathbb{N}^{\mathcal{V}}/_{\sim}$ where $g \sim h$ if and only if $(\forall x \in v,\ g(x) \in [|0; 1|]$ *and* $g(x) = h(x))$ *or* $(\exists x_1, x_2 \in v,\ g(x_1) > 1$ *and* $h(x_2) > 1)$.

Our deadlock analysis then gives the abstract transition system $(t^\sharp, f)$ where $t^\sharp$ is given as follows:



where

$$q_0 = \left\{ f \;\middle|\; \begin{array}{l} f(\text{push}!^6[]) + f(\text{push}!^3[]) = 1 \\ f(\text{push}!^3[]) = f((1,6)) \in [|0; 1|] \\ f(x) = 0,\ \forall x \in \{(5,3); (5,6)\} \end{array} \right\}$$

$$q_1 = \left\{ f \;\middle|\; \begin{array}{l} f((5,6)) = 1 \\ f(x) = 0,\ \forall x \in v \setminus \{(5,6)\} \end{array} \right\}$$

$$q_2 = \left\{ f \;\middle|\; \begin{array}{l} f((1,6)) = f((5,3)) = 1 \\ f(x) = 0,\ \forall x \in v \setminus \{(1,6); (5,3)\} \end{array} \right\}$$

By applying our primitive relation *finite*, we prove that no infinite computation sequence can pass by a configuration in $\gamma_{\mathbb{N}}\nu(q_1) \cup \gamma_{\mathbb{N}}\nu(q_2)$, which means that executions of our system are bound to terminate as soon as a communication $(5,3)$ or a communication $(5,6)$ is performed. □

8 Conclusion

We have designed a powerful framework to prove properties on the potential behavior of a mobile system. Our analysis allows to detect, in polynomial time on the number of sub-processes of the mobile system, mutual exclusion and non-exhaustion of resources. Our analysis has succeeded in analyzing very quickly a few nested concrete systems, featuring unbounded communication topologies, with the expected level of accuracy. Deadlock detection is a much more difficult problem and our proposed approach is likely to require exponential time. However it gives interesting results on unbounded systems, which are out of reach of model checking methods. We are likely to refine our initial partitioning in order to get a quicker analysis.

This framework is likely to be enriched by using [14] in order to detect non-

uniform confidentiality properties between processes, and to allow the analysis of mobile systems in hostile context. This will lead to a very powerfull modular analysis for mobile systems. To scale up, for large nested mobile systems, the analysis must be more approximate to ensure short execution time.

Acknowledgement

We deeply thank anonymous referees for their significant comments on an early version. We wish also thank Patrick and Radhia Cousot, Arnaud Venet, Jorge Pinto and Antoine Miné, for their comments and discussions.

References

[1] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[2] C. Bodei, P. Degano, F. Nielson, and H.R Nielson. Control flow analysis for the π-calculus. In *Proceedings of CONCUR'98*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

[3] F. Bourdoncle. Abstract interpretation by dynamic partitioning. *Journal of Functional Programming*, 2(4), 1992.

[4] L. Cardelli and A. D. Gordon. Mobile ambients. In *Foundations of Software Science and Computational Structures*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.

[5] C.K. Chiu and J.H.M. Lee. Interval linear constraint solving using the preconditioned interval gaus-seidel method. In *Proceedings of the Twelfth International Conference on Logic Programming*, Logic Programming, pages 17–32. The MIT Press, 1995.

[6] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes.* PhD thesis, Université Scientifique et Médicale de Grenoble, 1978.

[7] P. Cousot. Semantic foundations of program analysis. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 10, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, 1981.

[8] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, U.S.A., 1977.

[9] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the Sixth Conference on Principles of Programming Languages POPL'79*. ACM Press, 1979.

[10] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of logic and computation*, 2(4):511–547, August 1992.

[11] P. Cousot and R. Cousot. Comparing the Galois connection and widening-narrowing approaches to abstract interpretation. In *Programming Language Implementation and Logic Programming, Proceedings of the Fourth International Symposium, PLILP'92*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295. Springer-Verlag, 1992.

[12] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the Fifth Conference on Principles of Programming Languages*. ACM Press, 1978.

[13] J. Feret. Conception de *π-sa* : un analyseur statique générique pour le π-calcul. Mémoire de dea, SPP, september 1999. Electronically available at `http://www.di.ens.fr/~feret/dea/dea.ps`.

[14] J. Feret. Confidentiality analysis for mobiles systems. In *Seventh International Static Analysis Symposium (SAS'00)*, volume 1824 of *LNCS*. Springer-Verlag, 2000.

[15] R. R. Hansen, J. G. Jensen, F. Nielson, and H. R. Nielson. Abstract interpretation of mobile ambients. In *Proc. SAS'99*, number 1694 in Lecture Notes in Computer Science, pages 134–148. Springer-Verlag, 1999.

[16] Jean-Pierre Jouannaud. Rewrite proofs and computations. In Helmut Schwichtenberg, editor, *Proof and Computation*, volume 139 of *Computer and Systems Sciences*, pages 173–218. Springer-Verlag, 1995.

[17] M. Karr. Affine relationships among variables of a program. *Acta Informatica*, pages 133–151, 1976.

[18] R. Milner. The polyadic π-calculus: a tutorial. In *Proceedings of the International Summer School on Logic and Algebra of Specification*. Springer Verlag, 1991.

[19] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100:1–77, 1992.

[20] H. Riis Nielson and F. Nielson. Shape analysis for mobile ambients. In *Proc. POPL'00*, pages 142–154. ACM Press, 2000.

[21] D. N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, Edinburgh University, 1995.

[22] A. Venet. Abstract interpretation of the π-calculus. In *Proc. of the Fifth LOMAPS Workshop on Analysis and Verification of High-Level Concurrent Languages*, volume 1192 of *Lecture Notes in Computer Science*, pages 51–75. Springer-Verlag, 1996.

[23] A. Venet. Automatic determination of communication topologies in mobile systems. In *Proceedings of the Fifth International Static Analysis Symposium SAS'98*, volume 1503 of *Lecture Notes in Computer Science*, pages 152–167. Springer-Verlag, 1998.

Recent BRICS Notes Series Publications

NS-00-3 **Patrick Cousot, Eric Goubault, Jeremy Gunawardena, Maurice Herlihy, Martin Raussen, and Vladimiro Sassone, editors.** ***Preliminary Proceedings of the Workshop on Geometry and Topology in Concurrency Theory, GETCO '00,*** **(State College, USA, August 21, 2000), August 2000. vi+116 pp.**

NS-00-2 **Luca Aceto and Björn Victor, editors.** ***Preliminary Proceedings of the 7th International Workshop on Expressiveness in Concurrency, EXPRESS '00,*** **(State College, USA, August 21, 2000), August 2000. vi+130 pp.**

NS-00-1 **Bernd Gärtner.** ***Randomization and Abstraction — Useful Tools for Optimization*****. February 2000. 106 pp.**

NS-99-3 **Peter D. Mosses and David A. Watt, editors.** ***Proceedings of the Second International Workshop on Action Semantics, AS '99,*** **(Amsterdam, The Netherlands, March 21, 1999), May 1999. iv+172 pp.**

NS-99-2 **Hans Hüttel, Josva Kleist, Uwe Nestmann, and António Ravara, editors.** ***Proceedings of the Workshop on Semantics of Objects As Processes, SOAP '99,*** **(Lisbon, Portugal, June 15, 1999), May 1999. iv+64 pp.**

NS-99-1 **Olivier Danvy, editor.** ***ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation, PEPM '99,*** **(San Antonio, Texas, USA, January 22–23, 1999), January 1999.**

NS-98-8 **Olivier Danvy and Peter Dybjer, editors.** ***Proceedings of the 1998 APPSEM Workshop on Normalization by Evaluation, NBE '98 Proceedings,*** **(Gothenburg, Sweden, May 8–9, 1998), December 1998.**

NS-98-7 **John Power.** ***2-Categories*****. August 1998. 18 pp.**

NS-98-6 **Carsten Butz, Ulrich Kohlenbach, Søren Riis, and Glynn Winskel, editors.** ***Abstracts of the Workshop on Proof Theory and Complexity, PTAC '98,*** **(Aarhus, Denmark, August 3–7, 1998), July 1998. vi+16 pp.**

NS-98-5 **Hans Hüttel and Uwe Nestmann, editors.** ***Proceedings of the Workshop on Semantics of Objects as Processes, SOAP '98,*** **(Aalborg, Denmark, July 18, 1998), June 1998. 50 pp.**