Schedulers as Abstract Interpretations of Higher-Dimensional Automata

Eric Goubault*

École Normale Supérieure

Abstract

We define here a unified formal treatment for scheduling problems arising in different areas of computer science (implementation of concurrent languages, protocols for distributed systems, pipeline management, concurrent databases...) through abstract interpretation of Higher-Dimensional Automata (HDA) semantics. We give practical polynomial time algorithms for testing/inferring schedulers at the end of the article.

1 Introduction and motivation

The use of schedulers is somewhat pervasive to many branches of computer science. We mention below a few application areas, the properties that schedulers are to verify and give some references to the theoretical work done in these different fields.

1.1 Safety and efficiency of the implementation of concurrent languages

A real parallel machine has but a limited number of resources. It has limited memory, limited number of processing units and many constraints on the way it can use them. The idealistic view of true concurrency semantics, assuming an infinite number of processors for instance, is therefore misleading when it comes to runtime behaviour of programs. It may happen that to badly schedule spawning operations may deadlock (just delay in practise) a process that would need to synchronize with another (not yet executed) process. It may happen as well that some shared resources of the machine have to be used in **mutual exclusion**. The safety (respectively efficiency) properties that schedulers must verify are mainly choosing behaviours that will not lead to deadlocks (respectively not delay too much the execution of some process) and guaranteeing mutual exclusion of some resources. This last property could well be implemented by standard techniques (Peterson's algorithm for shared-variables or hardware test-and-set like operations) independently of programs, but this would be at the expense of efficiency. Let us take an example from [14] and [27]. Many modern CPUs like SPARCs or MIPS pipeline instructions. Of course, their functional units, registers or bus are all used in mutual exclusion. Unfortunately the pipelined instructions overlap in time as they use more than one clock cycle and some of them cannot be executed (otherwise "**structural hazards**" occur) within a certain number of cycles after some others. We do not want to use the pipeline in mutual exclusion since we would have to empty it after every instruction. The problem addressed in [27] is to verify that schedulers for a single process ensure that structural hazards will not occur (safety). In a concurrent framework, if there are more processes than processors, we can address the new problem of finding a way to interleave actions from different processes executed on the same processor, that verify the constraints while using the pipeline at the best of its capabilities (this is a view formalised in [2]). Some processors (like INTEL's Pentium) are even more complex to deal with since some resources may be used by at most two processes in parallel but not three[1]. A similar example at a more macroscopic level is given by an I/O buffer shared by two or more processes. The main contribution of this article in this area is to provide a way to get the best scheduler (or an approximation of it).

An inverse problem (where we want to unconstrain the ordering of actions) can be found in the very important **parallelisation** literature. Given a sequential program p, can we decide which parts of it can be executed in parallel? Most approaches up to now are based on program transformation [19, 24, 29]. We propose here a framework based on the operational semantics of p that enables us to derive a scheduler choosing dynamically the intructions to spawn.

1.2 Protocols in distributed/concurrent systems

In order to have well behaved distributed systems, one very often has to make local processors agree on some criterion, like elect a leading one or organise the flow of information to guarantee the coherence of the global state of the system (by local rules only) like the consensus, set or renaming **agreement tasks** [15]. This is done by defining protocols.

*LIENS, École Normale Supérieure, 45 rue d'Ulm, 75230 Paris Cedex 05, FRANCE, email:goubault@dmi.ens.fr

[1] it has two integer arithmetic units.

An example of such a situation is given by a parallel machine whose different units communicate by asynchronous messages along channels which have a given topology (let us say a ring topology for instance). Now, a protocol for guaranteeing a global knowledge of some fact must serialize all message passing primitives according to the communication topology (in our example of the ring topology, messages are to be waited for from say the left neighbour before passing them to the right neighbour). In [15] some of these problems are addressed in a static manner (the topology is fixed once and for all). We propose here to use the dynamic semantics to deal with changing topologies as well[2].

Another example can be found in concurrent database systems [32]. To ensure the consistency of the database the processes have to lock some entries and then unlock them after some of their transactions have been executed. Protocols define the way processes lock and unlock items. A good instance of this is the **two-phase protocol**, see section 2.1. Our contribution in this area is to give a general definition of serializability, carrying on the work presented in [11], and give a practical test for protocols based on the semantics of the processes and not only on static or syntactic ground. Very recently, Jeremy Gunawardena [13] has given a very clear explanation about why serializability has something to do with **homotopy**. We subscribe to this point of view and give here a formalization of these notions using **higher-dimensional automata**.

2 A geometric approach

Interestingly enough, a graph-based criterion is known for serializability [32]. In more general protocols for "decision problems" recent results [15] use combinatorial algebraic topology on static representations of protocols. We will show here that we can use more general tools from algebraic topology directly on the dynamic semantics of the systems studied to extract the information about serializability and schedulers. We will develop in particular a **homotopy theory** of oriented paths (next section). Let us explain the intuition about this.

2.1 The two-phase protocol

A concurrent database is composed of a set of shared objects, or *items*, and a set of processes accessing these items $T_1, \ldots, T_n$, or *transactions*. The transactions can be executed in parallel, and one can think of a good example (of economic interest too !) as being a reservation system of an airline. The items are seats in the planes and the transactions are individual queries from customers, made in parallel since there may be many different selling points. The basic property we want to insure is that no seat is sold twice (at the same time) to different customers. In the shared memory paradigm the well-known method for attacking this is to put locks ([10]) on shared variables. In Dijkstra's formalism, for an item a, Pa is the action of locking a and Va is the action of relinquishing the lock on a. As long as we are only interested in the policy of acquiring items and not in their actual values, we can abstract the transactions in such a way that they are written as strings of Px, Vx, x ranging over the shared objects.

[2] In many languages, like CM-Fortran with the CMMD library, or CML [28], channels are defined during the execution of the program. They are not physical but they are logical channels.

Figure 1: A process graph for two transactions accessing the same shared item.

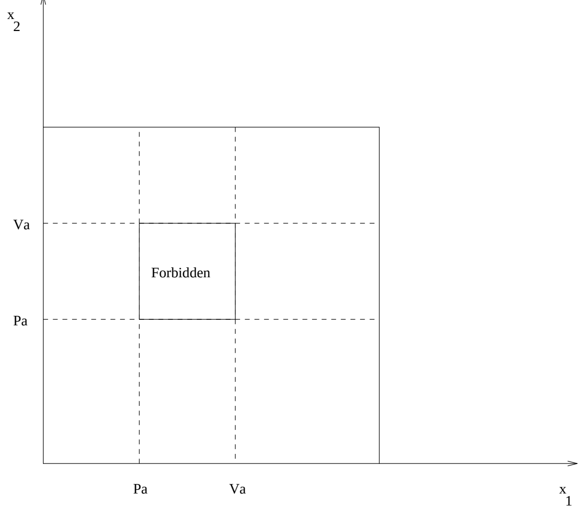
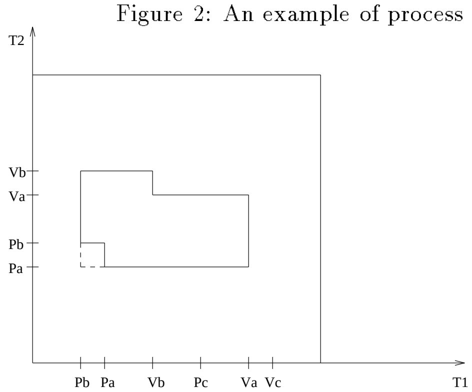


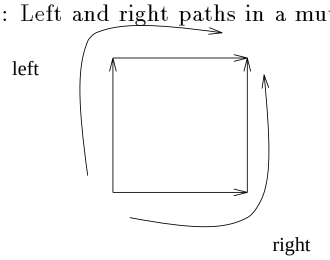
Figure 2: An example of process graph.



As an example, consider $T_1 = PaVa$, $T_2 = PaVa$. There is an old way to represent these transactions due to Dijkstra again, known as *process graphs*. We will see that it has much to do with the HDA approach. The idea is to associate to each transaction a "local time" which geometrically is one coordinate in an euclidean space. Supposing that all processes can individually terminate, we may normalise this local time for it to be in range $[0, 1]$. An purely asynchronous execution of n transactions is now any path from $(0, \ldots, 0)$ to $(1, \ldots, 1)$ in the n-cube $[0, 1]^n$ where the local times, i.e. the coordinates always increase. But the executions are constrained by the fact that shared objects are accessed in mutual exclusion. In Figure 1 we have pictured the central square in $[0, 1]$ which is forbidden: a valid path of execution cannot enter it since it is precisely the region in which both transactions access the same object a. The more complex example borrowed from [13], $T_1 = PbPaVbPcVaVc$, $T_2 = PaPbVaVb$ is pictured in Figure 2. Using these geometric representations, we have two main questions, (i) can the system of transactions deadlock? (ii) is the system of transactions correct in some sense?

As for question (i), the answer is geometrically clear (see

Figure 3: Left and right paths in a mutual exclusion.



[6]). The only way a path coming from $(0,\ldots,0)$ may be stopped before reaching $(1,\ldots,1)$ is by "meeting" a corner like the dashed one ($PaPb$, $PbPa$) in Figure 2. As soon as a path goes into the small dashed rectangle, it cannot reach $(1,\ldots,1)$. Formally, this question relates to a connectedness result. We will not discuss this in the following but rather concentrate on question (ii). Question (ii) is not immediate since we first have to define what the correctness condition is. In the airline reservation system example we have only demanded that no seat be sold twice. This means that some parts of the transactions may be done in parallel, but that the execution must be sufficiently constrained so that the resulting reservations are the same as some sequential treatment of the queries of the customers. In database theory this correctness criterion is known as "serializability". It has a basic "geometric" formalisation in [32] in the form of a topological condition on the "graph of transactions". We show now, following J. Gunawardena ([13]) that it is even more directly of a geometric nature, and that the serialization property can be read on the process graph.

This may seem strange since the correctness criterion seems essentially given as a condition on states of the system. Do not forget though that we assumed the representation of transactions does not depend on the actual values of items. Surely, some arithmetical operation involved in the booking process may commute with other operations for some values of the items, but we have to think it would be very strange that they commute for all values. The condition now is then only on paths of executions. If you look at the forbidden square, or mutual exclusion in Figure 1 reproduced in Figure 3, the values of the items at its top right depend on the way we have reached this point. Going on the left of the hole may give different results than going on the right of the hole: just think at the following example. The initial value of a is 0. The arithmetic operations involved for the processes when the lock on a has been acquired is $a := a+1$ for T_1 and $a := 2*a$ for T_2. Going on the left means doing $a := 2*a$ before $a := a+1$, result is $a = 1$. Going on the right means doing $a := a+1$ before $a := 2*a$, result is $a = 2$. Instead of looking at the holes, look at the filled parts of the drawing 1. It becomes obvious now that all paths below (or at the right hand side of) the hole are serializable to the right boundary of the square, and that all paths above (or at the left hand side of) the hole are serializable to the left boundary of the square. Holes appear to be the elements to discover. They are the obstructions to the "continuous" deformation of paths (homotopy), which is the "infinitesimal" serialization equivalence. Here, the system is serializable since any path can be deformed onto one of the interleavings $T_1;T_2$ or $T_2;T_1$, i.e. any path gives the same result as a serial execution of the transactions.

Let us examine another process graph we may have (see

Figure 4: A process graph with two mutual exclusions.

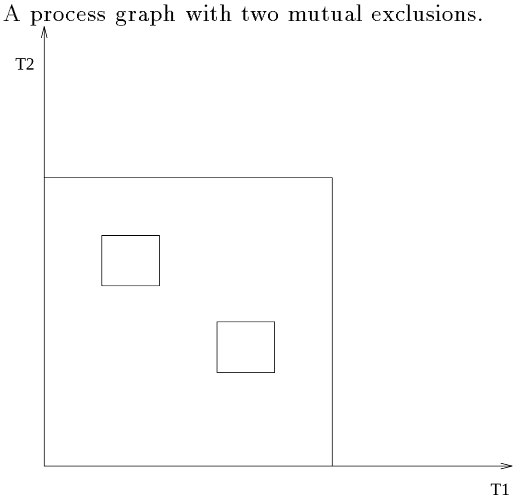


Figure 4). Here, the paths "in between" the two holes cannot be deformed onto one of the interleavings, hence they are not serializable.

The aim of protocols for concurrent databases is to provide us with a uniform[3] way of insuring the consistency of a database. A good example is the two-phase protocol. Every transaction must acquire all locks of all items they will compute on (first phase), compute, and at the end they must release all their locks (second phase). For instance $T_1 = PaPbPcVaVbVc$ verifies the two-phase protocol (is "two-phase locked") whereas $T_2 = PaVaPbVb$ does not. It can be proven by combinatorial means that it makes all systems of transactions serializable (or in short, it is serializable). However it does not prevent a system from deadlocking. Geometrically, the proof that it is serializable has been given in [13], and is much more illuminating than the combinatorial one of, say, [32]. Basically, it is proven that the n-cubes forbidden by the two-phase protocol form a unique hole in the "center" of $[0,1]^n$. It is then easy, using a "radial" homotopy to deform all paths of execution onto one of the interleavings, and then prove the serializability.

There are a few matters of insatisfaction in this proof though. First, it uses continuous methods. They are elegant but induce a few complications, like knowing that the paths correspond to real ones. Second, it uses a standard theory of homotopy which authorises reversal of time. Here, we really need a homotopy theory for "oriented" paths as already shown in [26]. In the following we sketch such a theory, in a discrete framework using Higher-Dimensional Automata. The theory will generalise also for higher-dimensional mutual exclusion problems.

2.2 Protocols for distributed systems

Here, we want to deal with general problems that one may encounter while programming distributed systems. The case of concurrent databases can be considered as a particular case. More generally, we are concerned with the following type of problems.

Given a number of hypotheses on the distributed system,

[3] i.e. independently of what process we want to program.

like a topology of the communication network, a specification of the way messages are sent and received (asynchronously, synchronously, with bounded buffers, with no loss...), or in case of a shared-memory system, a number of assumptions like sequential/concurrent read/write...;

Given a specification of what we want to program (as a set of distributed processes) on that system in the form of conditions on the input values accepted by this set of processes and conditions on the output values that this set of processes should compute;

Given a number of requirements on the execution of this program, like being as most efficient as it can be, or (it may be seen as a limit case of the previous requirement) being robust enough to compute a good part of the output specification even if some processors fail;

The questions are: "Does such an algorithm exist on such machines ?" and if the answer is positive, "Can we derive it from the specification of the problem ?".

All this is formalised under the name of *decision tasks*. Let us first give a few examples, following the presentation of [16] and [15]. The consensus task (abstraction of the commitment problem in concurrent database theory where transactions have to agree on a common value or abort) is a decision task in which N asynchronous processes begin with arbitrary input values in some set S and must agree at the end on some common value taken from S. The k-set agreement task asks for arbitrary input values (in some set S) but no more than k output values (in S as well). This can be seen as a partial consensus among the processes.

Now, an algorithm may be constrained in the following way. Call an execution of a program on a distributed system t-faulty if at most t processes in the program fail. Then an algorithm is t-resilient if it solves a decision task in every t-faulty execution. An algorithm is wait-free if it is $(n-1)$-resilient (n is the number of processes we have). It is proven for instance in [16] that in a shared-memory model with single reader/single writer registers providing atomic read and write operations, k-set agreement requires at least $\lfloor f/k \rfloor + 1$ rounds where f is the number of processes that can fail. This is done in a very nice geometric framework, and general tests are given for solving t-resilient problems. Not only impossibility results can be given but also constructive means for finding algorithms derive from this work (see for instance [17]).

The geometric framework is different from ours, but hints about their relationships will be given in appendix C. Herlihy's framework is based on a representation of the input and output specifications in the form of input and output simplicial complexes, [20].

A simplex is associated to the states of processes in the following manner. Vertices v are pairs $(val(v), id(v))$ of local values of the process having identifier $id(v)$, We can draw an edge between v and v' if and only if v and v' are compatible with the specification we have of the state of the system. It means in all cases that v and v' must have distinct process identifiers. Higher-dimensional simplexes include states $v_1, \ldots, v_n$ if and only if these states are compatible with the specification (input or output one). Again, all the $id(v_i)$ are distinct. As an example, we give the specification complexes for the consensus task with $S = \{0, 1\}$, called the binary consensus task. The input complex has simplexes of the form $((P_0, b_0), \ldots, (P_n, b_n))$ with $0 \leq n \leq N$ and $b_i \in \{0, 1\}$ since all processes can take whatever boolean value they want. It is homeomorphic to a n-sphere, The output complex has simplexes of the form $((P_0, 0), \ldots, (P_n, 0))$ or $((P_0, 1), \ldots, (P_n, 1))$ since all processes should agree on some common boolean value. This complex has exactly N connected components. Then, an algorithm for solving such decision tasks can be given in geometric terms, relating the geometry of the input and output complexes.

2.3 Our geometric approach

Here we are more interested with the analysis of programs and of architectures of machines. This elaborates on the case of concurrent databases in the sense that we need a real classification of all possible orderings of actions (i.e. of all schedulers) and not only a proof that all schedulers are "equivalent" to one of the interleavings of the transactions.

We first have to describe the semantic model we use from which all that information can be extracted. It is a very simple geometric model for true concurrency, based on the ideas by Vaughan Pratt and Rob van Glabbeek [26, 33] and formalised in different ways in [11], [12]. To explain this, we can start off with ordinary transition systems. A transition system is a structure $(S, i, L, Tran)$ where S is a set of states, i is the initial state, L is the set of labels and $Tran \subseteq S \times L \times S$ is the transition relation. We need to distinguish between true concurrency and non-determinism before being able to discuss about scheduling properties since the latter may come from a necessary mutual exclusion and the former would then describe an unsound behaviour. A possible answer is to decorate the transition systems with some relation prescribing the independence of some actions (or transitions). This can be done in more than one manner; just to mention a few: asynchronous transition systems [4, 30], concurrent automata [31] and transition systems with independence [34]. We comment on the former only, since exhaustivity would be too space consuming.

Asynchronous transition systems are equipped with an irreflexive symmetric binary independence relation I on actions verifying a few conditions. The most important is that independence of actions means confluence of the transition relation for the actions involved. This decoration on ordinary transition systems (the independence relation I) is enough to make the distinction between non-determinism and true concurrency. There is a slight problem though. The level of parallelism is not defined in a very precise manner since the independence relation is only a binary one. It is necessary in particular in the example of the Pentium processor since we need to distinguish between two concurrent calls to the arithmetic units (truly parallel) and three (in which case one of these has to be interleaved with some other one). Of course, a straightforward generalization would be to replace the binary relation I by an n-ary relation. This could be done (though we do not have any pointers in the literature) but this is then awkward to use when one wants to effectively manipulate things (since the independence relation is of a somewhat different nature than transitions everything becomes heavy work) and when one wants to constrain things to an arbitrarily chosen number of processors.

This can be tackled if we get back to our geometric intuition. Things have been made overly unnatural by adding an object (the independence relation) which is not of the same nature as transitions and states. Just think of aIb as an abstraction of all possible asynchronous executions of a

Figure 5: Non-determinism (i) versus overlap in time (ii) abstracted by a transition of dimension 2 (iii).

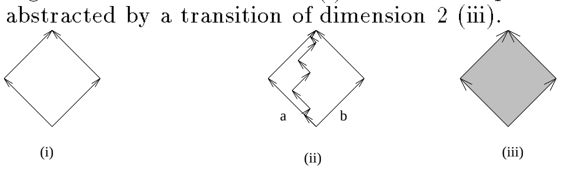
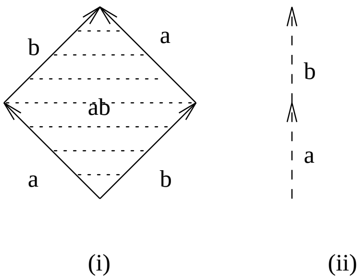


Figure 6: A HDA (i) and its set of paths seen as schedulers (ii).

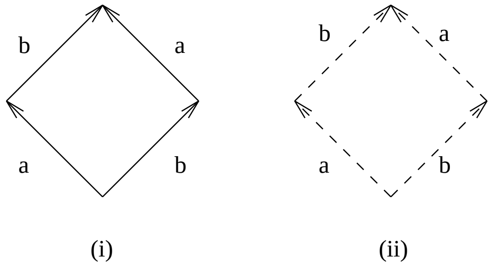


and b. As in [26], this can be pictured as the filled-in square of the right-hand side of Figure 5, distinguishing it with the interleaving at the left-hand side of the same Figure. Notice that geometrically, the interior of the square consists of the union of all paths where executions of a and b overlap "in time" (middle picture of Figure 5). As a direct generalization, asynchronous execution of n transitions give rise to hypercubes of dimension n, called n-transitions (ordinary transitions are 1-transitions, states are 0-transitions). This is very close to Dijkstra's process graphs, but in a discrete framework. Interestingly enough, all this has a very neat algebraic formulation (see next section).

Let us look now at the geometric counterpart of schedulers or protocols in this setting. Suppose we have a real machine with only a finite number n of processors on which we want to implement a semantics given by HDA. What should we consider as a **valid implementation** ?

We first look at an instructive example for $n = 1$. Suppose the semantics of a program P is given by the truly concurrent execution of a and b pictured as the 2-transition in (i) of Figure 6. Then the valid execution paths are given by (ii) of the same figure. A scheduler can choose statically to do a then b or b then a. a then b is one scheduler and b then a is another. They are essentially the same (this will be defined formally as an equivalence relation between schedulers) since a and b are non-interfering. In a geometrical manner,

Figure 7: A HDA (i) and its set of paths seen as a scheduler (ii)



they are equivalent since one can continuously deform one path onto the other through the 2-transition ab (homotopy). In more well-known terms (Mazurkiewitz trace theory) one can understand ab as a commutation relation between a and b that is, ab is serializable to a then b and serializable to b then a [32].

If P were the mutual exclusion between a and b ((i) of Figure 7) then do we have also two equivalent schedulers on a one-processor machine ? The answer is no: choosing *a priori* to fire a before b is radically different from choosing *a priori* to fire b before a. Suppose for instance that a is the action on a process 1 of accessing a shared resource R and b is the action on a process 2 of accessing R as well. Then we should think of the two processes to be in competition for R and the scheduler does not have to make one wait for the other to access it first if the other was ready to: it is a matter of inefficiency and it transforms the properties of the program (livelocks, deadlocks). Moreover, if we are at an abstract level of the semantics, (where we have folded together some of the states for instance) we cannot be sure that the results of the two paths will be the same. We knew that when we had the 2-transition in Figure 6 because it indicated a non-interfering behaviour, but here we just do not know. This is the abstract point of view implicitly used for studying protocols and concurrent databases (because they should not depend on the particular values of the items). There must then be one and only scheduler whose trace is represented as (ii) in Figure 7. The initial state of (i) is an internal choice the parameters of which the scheduler cannot influence. There, the "hole" between $a;b$ and $b;a$ prevents us from deforming one onto the other. We now formalize this in more abstract geometrical terms.

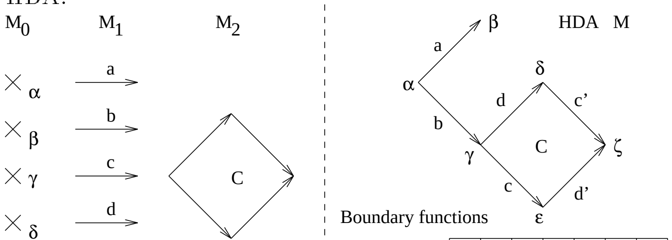
3 Schedulers and homotopy of regular HDA

3.1 Semi-regular and Regular HDA

We present the geometric shapes we are interested in as unions of points, segments, squares, ..., hypercubes, i.e., as collections of n-transitions ($n \in \mathbb{N}$). We glue them together by means of boundary relations (see Figure 8), given by two boundary operators: d^0, the start boundary operator and d^1 the end boundary operator. They generalize the source and target functions for ordinary automata: the boundary of a segment (respectively square...) is composed of two points (respectively of four segments...).

Consider the square,
$$\begin{tikzcd} (0,0) \arrow[r, "a"] \arrow[d, "b"'] \arrow[dr, phantom, "A"] & (0,1) \arrow[d, "b'"] \\ (1,0) \arrow[r, "a'"] & (1,1) \end{tikzcd}$$
This corresponds to the asynchronous execution of actions a and b (a' and b' are copies of transitions of label a and b respectively). The object of dimension 2 "interior of the square" A should certainly have two source boundaries, up to the order on $\{a, b\}$, $d_0^0(A) = a$ and $d_1^0(A) = b$ since from state $(0, 0)$ we can fire a and b. Similarly, it should have two target boundary operators $d_0^1(A) = a'$ and $d_1^1(A) = b'$ since from the parallel execution of a and b (represented by A) we can end first action a (giving "residue" b') or action b (giving "residue" a'). We will see that again when speaking about paths. Notice that with this ordering on vertices, we have, $d^0(d_1^0(A)) = (0, 0) = d^0(d_0^0(A))$ and $d^1(d_1^0(A)) = (1, 0) = d^0(d_0^1(A))$. We can show

Figure 8: Glueing of elementary shapes to get a semi-regular HDA.



		a	b	c	d	c'	d'	C
d_0^0	=	α	α	γ	γ	δ	ε	c
d_1^0	=	-	-	-	-	-	-	d
d_0^1	=	β	γ	ε	δ	ζ	ζ	c'
d_1^1	=	-	-	-	-	-	-	d'

that for any hypercube of dimension n, we can choose an ordering on vertices, squares ... such that the $2*n$ boundary operators verify the commutation rules[4], $d_i^k \circ d_j^l = d_{j-1}^l \circ d_i^k$ for $k=0,1$, $l=0,1$ and $i<j$ ($\circ$ is the ordinary composition of functions). Now we can **glue** these elementary shapes in order to get HDA. This is exemplified in Figure 8. We verify on the example the commutation rule between the source and target boundary operators d^0 and d^1 respectively.

We can then introduce these formally under the name of unlabelled semi-regular HDA. We will not develop the full theory of labelled semi-regular HDA (or higher-dimensional transition systems) in this article due to lack of space.

Definition 1 *An unlabelled semi-regular HDA is a collection of sets M_n ($n \in \mathbb{N}$) together with functions*

$$M_n \underset{d_j^1}{\overset{d_i^0}{\rightrightarrows}} M_{n-1}$$

for all $n \in \mathbb{N}$ and $0 \leq i,j \leq n-1$, such that $d_i^k \circ d_j^l = d_{j-1}^l \circ d_i^k$ ($i<j, k,l=0,1$) and $\forall n,m$ $n \neq m$, $\quad M_n \cap M_m = \emptyset$.

Elements x of M_n ($dim\ x = n$) are called n-transitions (or states if $n=0$).

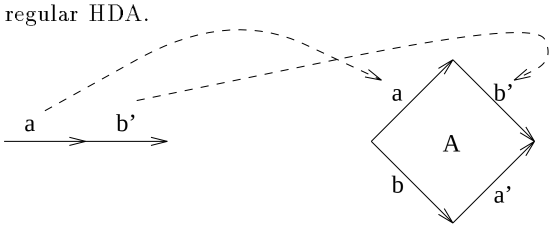
In order to be able to study "natural" constructions on HDA, we define a notion of **morphism** between them. As customary in recent work in concurrency [34], morphisms look like **simulations**. In geometrical terms, morphisms preserve shapes (every n-transition is mapped onto a n-transition) and orientation.

Definition 2 *Let M and N be two semi-regular HDA, and f a family $f_n : M_n \rightarrow N_n$ of functions. f is a morphism of semi-regular HDA if and only if $f_n \circ d_i^0 = d_i^0 \circ f_{n+1}$ and $f_n \circ d_i^1 = d_i^1 \circ f_{n+1}$ for all $n \in \mathbb{N}$.*

This defines the category Υ_{sr} of semi-regular HDA. Now, traces of execution are described as sequences of states and transitions satisfying certain properties. A **path** is to be understood as a sequence of *allocation* of one action at a time on a new processor or *deallocation* of one action at a time (i.e. its execution has ended on a given processor). An example of a path in an automaton M is given in Figure 9 together with its inclusion morphism into M (M simulates all of his paths).

[4] very much alike the ones we have for simplicial complexes. Ideas of many constructions of the article actually come from combinatorial algebraic topology.

Figure 9: A path and its inclusion morphism in a semi-regular HDA.



We can elaborate a bit on the previous definitions to allow **formal sums** of transitions (to speak about multisets of transitions and hence about paths): we choose n-transitions to be elements of modules or vector-spaces. We choose to make the representation of transitions a bit more explicit and do as in [11], add a new index to M: we will have a decomposition of each M_n into $\sum_{p+q=n} M_{p,q}$. This will enable us to have a direct representation of paths (see the appendices). For instance, the HDA M of Figure 8 can be described by $M_{0,0} = (\alpha)$, $M_{1,0} = (a) \oplus (b)$, $M_{1,-1} = (\beta) \oplus (\gamma)$, $M_{2,-1} = (d) \oplus (c)$, $M_{2,-2} = (\delta) \oplus (\epsilon)$, $M_{3,-2} = (c') \oplus (d')$, $M_{3,-3} = (\zeta)$ and $M_{3,-1} = (C)$ (where (x) is the module generated by x and $\oplus$ is the direct sum of modules). The 1-path (b,c,d') can now be conveniently identified with the formal sum $b+c+d'$. In the following, R is a principal commutative domain [18].

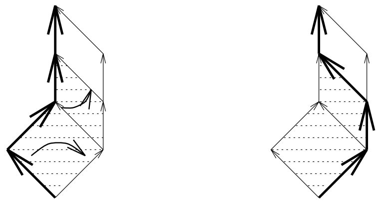
Definition 3 *A regular HDA is a direct decomposition of a free R-module M as $M = \oplus_{p,q \geq 0} M_{p,q}$ together with boundary operators $d_i^0 : M_{p,q} \rightarrow M_{p-1,q}$ ($0 \leq i \leq p+q-1$) and $d_j^1 : M_{p,q} \rightarrow M_{p,q-1}$ ($0 \leq j \leq p+q-1$) such that $d_i^k \circ d_j^l = d_{j-1}^l \circ d_i^k$ (for all $i<j$ and $k=0,1$, $l=0,1$).*

Morphisms of regular HDA are $f : M \rightarrow N$ with $f = (f_{p,q})_{p,q}$, where $f_{p,q} : M_{p,q} \rightarrow N_{p,q}$ are module homomorphisms such that $f_{p,q} \circ d_i^0 = d_i^0 \circ f_{p+1,q}$ and $f_{p,q} \circ d_j^1 = d_j^1 \circ f_{p,q+1}$ for all i,j with $0 \leq i,j \leq p+q$. The category of regular HDA is denoted by Υ_r. We will also consider cyclic regular automata which are regular HDA in which some elements of $M_{p,q}$ and $M_{p',q'}$, $p'+q' = p+q$ may be identified. They form a category Υ_{cr}.

There is a relationship with the HDA defined in [11]. Let $\partial_0(x) = \sum_{i=0,\ldots,n-1} (-1)^i d_i^0(x)$ for $x \in M_n$ (where M is a regular or semi-regular automaton) and similarly, $\partial_1(x) = \sum_{i=0,\ldots,n-1} (-1)^i d_i^1(x)$. Then $\partial_0\partial_0 = \partial_1\partial_1 = \partial_0\partial_1 + \partial_1\partial_0 = 0$ and $(M, \partial_0, \partial_1)$ is a HDA in the sense of [11]. To make things clear, we write $\underline{M}$ for the general HDA derived in this manner from a regular or semi-regular HDA M. For example, HDA M of Figure 8 generates $\underline{M}$ with $\partial_0(C) = c - d$ and $\partial_1(C) = c' - d'$. We see that $\partial_0 \circ \partial_0(C) = \gamma - \gamma = 0$ and $\partial_0 \circ \partial_1(C) = \delta - \epsilon = -\partial_1 \circ \partial_0(C)$.

Paths as sequences of allocation and deallocation are not very easy to use. We prefer to use n-paths where we restrict actions to be of constant dimension n and where we collect

Figure 10: Step by step deformation (curved arrows) of one path onto an other



all possible ways n-transitions can end. There is a sense in which only considering n-paths is equivalent to considering all paths. In particular, 1-paths and paths whose elements are all of dimension one coincide.

Definition 4 *A n-path p is a finite sequence $(p_i)_{i=1,\dots,k}$ of n-transitions such that $\partial_1(p_i) = \partial_0(p_{i+1})$.*

From now on, we choose to have only initial states for 1-paths in $I \subseteq M_{0,0}$.

3.2 Towards formal definitions

Let M be a regular automaton. Let $p = (p_i)_{i=1,\dots,k}$ be a 1-path. Let α and ϵ be respectively its initial and final states.

We wish to define **geometrically** how two paths of dimension one are to be considered equivalent in a scheduler. We have already noticed that they should be considered equivalent if one can deform any of them into the other one. In order to give a definition, let us look at the HDA from a different point of view. We slice paths into actions that occur at a given time: supposing that all paths we are interested in begin in $M_{0,0}$, we say that we are at time i when we look at actions in $M_{i,-i+1}$.

Let p and q be two paths. We say that p and q are **elementary equivalent** at time i if and only if p_i and q_i are two ends of a 2-transition A and $p_j = q_j$ for $j < i-1$ and $j > i$ and if p_{i-1} and q_{i-1} are two beginnings of the same 2-transition A. This corresponds to the idea of continuously deforming one path onto the other (or to use a commutation rule between two transitions) like one can see in Figure 10. We define equivalence to be the reflexive transitive closure of elementary equivalence.

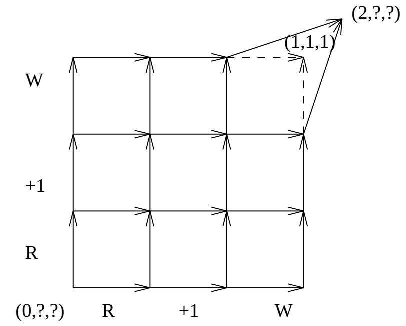
Looking again at Figure 8, we see that path $b+c+d'$ is elementary equivalent to path $b+d+c'$ since we can deform c into d and d' into c' through C. Paths a and b are not elementary equivalent.

The **algebraic definition** of this equivalence is given in appendices A and B. It takes the form of homotopy groups (or modules) $\Pi_n(X)$ for an HDA X and a dimension n that **collect all equivalence classes of n-paths of X.**

3.3 Schedulers

A n-scheduler should basically be able to execute all possible n-paths up to equivalence. This means that a n-scheduler of an automaton D is a choice of a subHDA M of D such that **all n-paths of D are equivalent (homotopic) to a n-path of M.** Let us call scheduler a n-scheduler for all n. At the light of the previous sections, this is formalized as follows,

Figure 11: Conflict in the case of a shared memory parallel machine/concurrent database.



Definition 5 *A n-scheduler is a monomorphism (i.e. a cofibration in the homotopy theory we are considering, see [3]) $s : M \to D$ such that $\Pi_n(s) : \Pi_n(M) \to \Pi_n(D)$ is an isomorphism.*

A scheduler is now a cofibration inducing an isomorphism between all the $\Pi_n(D)$ and the $\Pi_n(M)$. This is known as a **weak equivalence** [3]. A basic property of the homotopy theory we use is that weak equivalence is the same as strong equivalence, i.e. a scheduler is a cofibration such that there exists $s' : D \to M$ with $s \circ s'$ and $s' \circ s$ homotopic to the identity. A scheduler is thus the choice of a retract of D.

Example 1 *In Figure 11 we have pictured the semantics of the program $P_1 \mid P_2$ where $P_1 ::= READA; A := A+1; WRITEA$ and $P_2 ::= READA; A := A+1; WRITEA$. In the figure we have abbreviated $READA$, $A := A+1$ and $WRITEA$ by R, $+1$ and W respectively. The shapes (deformed squares) are all filled in, indicating concurrency. The states are given by the value of A. To make the picture easy to read we have chosen to unfold the central squares (but not all of them), using the value of A read by P_1 (second component of the triple) and the value of A read by P_2 (third component of the triple). The picture thus contains ten squares, the two at the top right corner show the interference while writing the computed value into the shared variable A. From $A = 0$ we can have two different results, $A = 1$ or $A = 2$. Now the two-phase protocol added to the two processes will constrain the execution so that all of P_1 (respectively P_2) is executed before all of P_2 (respectively P_1). These are two equivalent 1-schedulers (linked together by the nine upper squares). The protocol is therefore sound in the sense that it is serializable.*

Obviously the algorithmic characterisation of schedulers is given by the weak equivalence condition and not the strong one since we have practical means for computing the homotopy modules (see section 5).

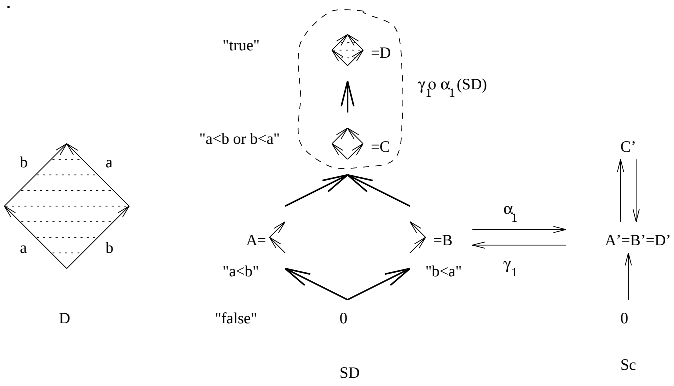
4 Abstract interpretation

4.1 Schedulers as an abstract interpretation

Suppose that the semantics of programs in some language is given by subHDA of a HDA D (containing all traces of

Figure 12: A domain of automata D, a subposet of SD and its abstraction to Sc

.



executions), called domain in [11]. Now, the set of subobjects SD of the semantic domain D ordered by inclusion is a complete lattice[5] as it can be identified to the category consisting of monomorphisms (or inclusion morphisms) into D modulo isomorphisms. Let us make this precise. Let $i : M \hookrightarrow D$ and $j : N \hookrightarrow D$ be two monomorphisms into D. Then $i = j$ in SD if and only if there exists an isomorphism $f : M \to N$ such that $i = f \circ j$. Let Sc be the category whose objects are equivalence classes of elements of $R-Mod/\Pi_n(D)$ modulo isomorphisms (same equivalence relation as above).

Define now $\alpha_n : SD \to Sc$ by $\alpha_n(i : M \hookrightarrow D) = (\Pi_n(i) : \Pi_n(M) \to \Pi_n(D))$. $\alpha_n(x)$ provides us with all n-schedulers of automaton x: $\alpha_n(x)$ returns basically the equivalence class of all retracts of dimension n of x (see Figure 12).

An analogous result to Van Kampen's theorem holds (appendix B). Therefore α_n commutes with (binary) least upper bounds. In case $\Pi_n(D)$ is of **finite type** (implied by D finite for instance), this proves the existence of a right-adjoint $\gamma_n : R-Mod/\Pi_n(D) \to SD$ to α_n by Freyd's special adjoint functors theorem [21]. (α_n, γ_n) is a pair of adjoint functors or a **Galois connection**[6]. We strongly believe that this generalizes to modules $\Pi_n(D)$ of infinite type but we do not have yet a proof of that.

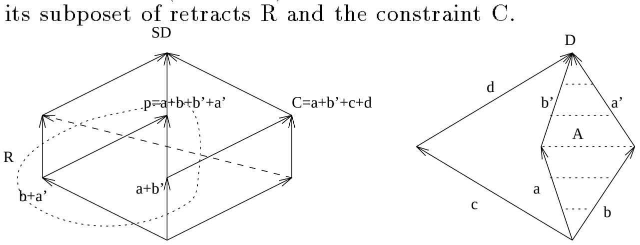
Figure 12 should then be understood as follows. When we have only one processor, A, B and D have exactly the same schedulers, i.e. they have essentially one and only 1-path (D retracts to any of A or B). This means that the best approximation of A and B (by $\gamma_1 \circ \alpha_1$) is D. As shown in the introductory part, only $C' = \alpha_1(C)$ (two paths, i.e. two generators) is different from (non-isomorphic to) $A' = \alpha_1(A)$, $B' = \alpha_1(B)$ and $D' = \alpha_1(D)$ (one path, i.e. one generator). The arrow in Sc going up from A' to C' is the image by α_1 of the inclusion morphism from A to C. Similarly, the arrow going downwards from C' to D' comes from the inclusion morphism of C into D and whose action is to project the hole of C onto 0 (the hole is filled in D).

Notice that all this discussion about schedulers given from the semantics is very much alike the **dependence orderings** that one may find in e.g. Mazurkiewitz trace theory [23] or more recently in concurrent automata [5]. Here we will write "$a < b$" meaning action b must be scheduled just after action a.

[5] this comes from the fact that Υ_{sr} is a complete and co-complete category.

[6] We do not ask to have a poset as an abstract domain, it may be a general category (or a preorder as in [8]). Notice that (as pictured in Figure 12) $\gamma_n \circ \alpha_n$ is an upper closure operator on SD, [7].

Figure 13: SD (simplified), the denotation p of the program, its subposet of retracts R and the constraint C.



In Figure 13 we have pictured a constraint on scheduling C which can be described as $C = c < d \vee a < b'$ as well as a program semantics p. In Figure 12, the constraints are written next to the corresponding elements of SD.

4.2 Verification of protocols

Given a constraint (or protocol) $C \in \mathcal{L}$ and a program p (identified with its semantics in D), can we find a best scheduler for p under constraint C ?

This problem can be expressed in our framework as follows. What is the maximal element of the intersection of the subposet of retracts of p with the left-closed set of elements satisfying the constraint C: $\{y \in SD / y \leq C\}$? or using a geometric image: "can we retract p onto $C \cap p$?". As an example, $a + b'$ is this maximal element in Figure 13. An algorithm is given in next section.

4.3 Inference of a best parallelisation

Here, we are given a sequential program p (identified with a HDA of dimension one) and we want to give a sense to the problem of finding the "best" parallelisation of it. The way we do this is by considering p to be embedded into a domain D specifying all possible actions. Practically, this is done by considering all traces in which all actions of p are put in parallel. This may obviously create some interferences or demonstrate the ability to perform some parts of p in parallel. Now, instead of retracting paths onto p, we wish to extend p as much as we can: we wish to find the greatest subHDA of D that retracts onto p. This makes sense since SD is complete. We derive an algorithm in next section.

5 Algorithmic details

As we do not want to specialise to a particular language or subproblem here we choose to give generic algorithms which may be optimised for some specific areas (see the conclusion). The generic algorithms rely on finding a solution to a central problem **cryptographists** have (finding dependence relations among lines of huge sparse matrices for quadratic sieves for instance). Huge progress is made everyday on finding good algorithms for solving this problem and these are then of direct interest for our abstract interpretation.

For keeping things understandable we use only a fairly well known algorithm.

5.1 Representation of HDA

The first simplification is to work with $R = \mathbb{Z}/2\mathbb{Z}$. This makes coefficients into simple booleans. Now boundary operators can be represented as boolean matrices, and even sparse ones: this means that lines (or line vectors) of the matrices are represented as ordered lists of integers indicating the occurrences of ones. Finite HDA are represented by the matrices of their boundary operators in every dimension. Whenever we have to work on two HDA one included into the other, we mark some of the line vectors of the greater one to indicate that they generate the other HDA as well.

5.2 Representation of program semantics and constraints

We generate the program semantics by **compositional methods** like in [11, 12] and then compute the abstract operators using standard methods from homology theory or preferably here by **SOS-like rules** that generate all possible transitions. The constraints filter the application of the SOS rules: the ones that verify the constraints are then transformed into marked lines. For the inference problem, the domain is generated by applying all valid rules for all instructions (this should be done lazily in a near future) and p is marked.

5.3 Verification

The algorithm can then be described as follows. Are given n, the representation of the HDA $C \cap p$ and p, initial $(n-1)$-transitions I (a line vector) and final $(n-1)$-transitions F (a line vector). The algorithm says if we can n-schedule p under the constraint C.

We suppose that we have already implemented the following functions:

$shift(M : HDA, k : integer)$ which shifts the dimension index of M by k, i.e. $shift(M, k) = N : HDA$ with $N_n = M_{n+k}$.

$quotient(M : HDA; I, F : vector)$ which returns the HDA M' where all states in I and F are replaced by 0.

$tot(M : HDA)$ which returns the matrices M_1 for the boundary operator $\partial_0 - \partial_1$ in dimension one and M_2 in dimension two.

We first program $triangular(U : matrix) = (U' : matrix, P : matrix)$ where U' is a triangular form of both the submatrix of marked lines of U and a triangular form of U, and P is the matrix of change of coordinates. In this way, null lines of U' corresponds to generators (whose expression can be read in P) of $Ker\ U$ and the non null lines of U' give a basis of $Im\ U$. If implemented by (a version of) Gauss method then the worst case complexity of $triangular(U)$ is in $O(ij^2)$ where j is the number of lines in U and i is the maximum number of non-null elements per line in U. If $tot(quotient(M, i, F)) = (M_1, M_2)$ where M is replaced by $shift(M, n-1)$ then the null lines of $triangular(M_1)$ represent the generators of $Ker(\partial_0 - \partial_1)$ in dimension one, i.e. the generators of the set of 1-paths of M (see appendix A). They can be computed in $O(nk_n^2)$ where k_n is the number of n-transitions in M. Similarly, the non-null lines of $triangular(M_2)$ represent the generators of $Im(\partial_0 - \partial_1)$ in dimension two. This can be computed in $O(nk_{n+1}^2)$. The algorithm $n - scheduler(p, C \cap p : HDA; I, F : vector) : boolean$ for verifying if we can n-schedule p under constraint C is described in Figure 14 and runs in $O(n(k_n^2 + k_{n+1}^2))$ where k_n and k_{n+1} are the number of n transitions (resp. $(n+1)$-transitions) in p.

Figure 14: The algorithm $n - scheduler$

1) $p = s(p, n-1)$
2) $(M_1, M_2) = tot(quotient(p, I, F))$
3) $(U_1, P_1) = triangular(M_1)$
4) $(U_2, P_2) = triangular(M_2)$
5) $N = \begin{pmatrix} N_u\text{=lines } q_j \text{ of } P_1 \text{ with } l_j = 0 \text{ in } U_1 \\ N_l\text{=non-null unmarked lines of } U_2 \end{pmatrix}$
6) $(N_1, Q_1) = triangular(N)$
7) $res = (card\{j/n_j = 0\} = card\{unmarked - lines(N_u)\})$

The algorithm works as follows. At lines 1) and 2) are computed the matrices of the boundary operator $\partial_0 - \partial_1$ for the transitions of dimension n and $n+1$ of the pair $(p, I \oplus F)$. The triangulations of lines 3) and 4) are used at line 5) for generating a matrix N whose first part (N_u whose lines which correspond to paths in $C \cap p$ are marked) is composed of generators of n-paths of p and whose second part (N_v) is composed of generators of $(\partial_0 - \partial_1)(p_{n+1})$. The triangulation makes explicit all dependency relations between N_u and N_v, that is show how many n-paths are homotopic in p (see appendix A and B). The last line verifies that all n-paths of p are equivalent to some path of $C \cap p$ by an easy argument on dimensions.

Example 2 *Take as domain and constraint those of the example pictured in Figure 13. For p, take the filled in square A. Then, the matrix representation of p is (where the marked lines are underlined), in dimension 1,* $\begin{pmatrix} \underline{1} & \underline{1} & \underline{0} & \underline{0} \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ \underline{0} & \underline{1} & \underline{0} & \underline{1} \end{pmatrix}$ *and in dimension 2,* $D_2 = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$ *If we run the algorithm 1-scheduler on these data, we first make disappear columns 1 and 4 in the matrix representing the objects of dimension 1 (since they correspond to the initial and final states respectively). Then we find,*

$$U_1 = \begin{pmatrix} \underline{1} & \underline{0} \\ 0 & 1 \\ 0 & 0 \\ \underline{0} & \underline{0} \end{pmatrix} \quad P_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \quad \textit{and } U_2 = D_2.$$

Then, $N = \left(\begin{pmatrix} 0 & 1 & 1 & 0 \\ \underline{1} & \underline{0} & \underline{0} & \underline{1} \end{pmatrix} \\ \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \right)$ *and* $N_1 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ \underline{0} & \underline{0} & \underline{0} & \underline{0} \\ 1 & 1 & 1 & 1 \end{pmatrix}$

This entails $res = (1 = 1)$ *is true, hence* p *can be implemented on a machine with constraint* C.

5.4 Inference

Are given, m (the maximum number of processors available), HDA D and p (of dimension one). p consists of the submatrix of marked lines of the matrix representation of D. The algorithm is an iteration on the following algorithm parameterized by the dimension n (we call Add_n), $for\ n = 1\ to\ m-1$, Add_n. It is basically the same as the one in Figure 14 except the res assignment is replaced by a marking of M_2 which describes the $(n+1)$-transitions to be added to p to get its parallelisation. The lines marked in M_2 are q_j with line vector $r_j \in Q_1$ such that $n_k \in N_1$ is null. This marking is then translated into a marking of p_{n+1} using the matrix of change of coordinates P_2. The complexity is bounded by a function of order $O(n^2 max_{1 \leq i \leq n-1} k_i^2)$.

6 Conclusion and future work

In this article, we have discussed the use of schedulers in different areas in computer science, and shown that a formal treatment involving HDA and abstract interpretation leads to elegant definitions and to algorithms for finding or verifying schedulers. The algorithms we have presented (which are being implemented in C) can most probably be used for quite big programs (about ten thousand transitions). Cryptographists and number theoreticians currently compute only a few dependency relations between the lines of huge sparse matrices - of width and height of about several hundred thousand elements -. This corresponds to testing/infering schedulers only on a subset of paths of a given program and could certainly give **widening** or dual widening operators. Other means of approximating the computation of homotopy groups are under study at the moment, like eliminating columns of the boundary matrices which have a great number of ones (much alike Odlyzko's method, [1]). Wiedemann's method [25] and iterative methods like conjugate gradient algorithms show also good promise for our problem. Another possibility is to work in a specific domain of HDA (like a labelling domain, [11]) for which the triangulation of the defining matrices has been solved. This has not been tested yet. Last but not least, an important point is that, as we have defined an abstract interpretation, we can always compose with other well-known ones, like folding some states together (see [9]), for getting things more tractable.

Acknowledgements Many thanks to Patrick Cousot, Régis Cridlig, Jeremy Gunawardena, Maurice Herlihy. Diagram macros from Paul Taylor.

References

[1] LaMacchia B. A. Solving large sparse linear systems over finite fields. In *Advances in Cryptology*. Springer-Verlag, 1990.

[2] H. Attiya and R. Friedman. A correctness condition for high-performance multiprocessors. In *Proc. of the 24th STOC*. ACM Press, 1992.

[3] H. J. Baues. Algebraic homotopy. In *Cambridge Studies in Advanced Mathematics*, volume 15. Cambridge University Press, 1989.

[4] M. A. Bednarczyk. *Categories of asynchronous systems*. PhD thesis, University of Sussex, 1988.

[5] F. Bracho, M. Droste, and D. Kuske. Representation of computations in concurrent automata by dependence orders. Technical report, Technische Universitat Dresden, 1994.

[6] S. D. Carson and P. F. Reynolds Jr. The geometry of semaphore programs. *ACM Transactions on Programming Languages and Systems*, 9(1):25–53, January 1987.

[7] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. of the 6th POPL*, pages 269–282. ACM Press, 1979.

[8] P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and per analysis of functional languages). In *Proceedings of the 1994 International Conference on Computer Languages*, pages 95–112. IEEE Computer Society Press, May 1994.

[9] R. Cridlig and E. Goubault. Semantics and analyses of linda-based languages. In *Proc. of WSA'93*, number 724. Springer-Verlag, 1993.

[10] E.W. Dijkstra. *Cooperating Sequential Processes*. Academic Press, 1968.

[11] E. Goubault. Domains of higher-dimensional automata. In *Proc. of CONCUR'93*, Hildesheim, August 1993. Springer-Verlag.

[12] E. Goubault and T. P. Jensen. Homology of higher-dimensional automata. In *Proc. of CONCUR'92*, Stonybrook, New York, August 1992. Springer-Verlag.

[13] J. Gunawardena. Homotopy and concurrency. In *Bulletin of the EATCS*, number 54, pages 184–193, October 1994.

[14] E. Harcourt, J. Mauney, and T. Cook. From processor timing specifications to static instruction scheduling. In *Proc. of the Static Analysis Symposium'94*. Springer-Verlag, 1994.

[15] M. Herlihy. A tutorial on algebraic topology and distributed computation. Technical report, presented at UCLA, 1994.

[16] M. Herlihy and N. Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proc. of the 25th STOC*. ACM Press, 1993.

[17] M. Herlihy and N. Shavit. A simple constructive computability theorem for wait-free computation. In *Proceedings of STOC'94*. ACM Press, 1994.

[18] S. Lang. *Algebra*. Addison-Wesley, third edition, 1993.

[19] S.T. Leung and J. Zahorjan. Improving the performance of run-time parallelization. In *ACM Sigplan Symposium on Principles and Practice of Parallel Programming*, May 1993.

[20] S. Mac Lane. Homology. In *Die Grundlehren der Mathematishen Wissenschaften in Einzeldarstellungen*, volume Band 114. Springer Verlag, 1963.

[21] S. Mac Lane. *Categories for the working mathematician*. Springer-Verlag, 1971.

[22] W. S. Massey. A basic course in algebraic topology. In *Graduate Texts in Mathematics*, number 127. Springer-Verlag, 1991.

[23] A. Mazurkiewicz. Basic notions of trace theory. In *Lecture notes for the REX summer school in temporal logic*. Springer-Verlag, 1988.

[24] P. Peterson and D. Padura. Dynamic dependence analysis: a novel method for data dependence evaluation. In *Proceedings of the fifth Workshop on Languages and Compilers for Parallel Computing*, volume 757. Springer-Verlag, August 1992.

[25] C. Pomerance and J. W. Smith. Reduction of huge, sparse matrices over finite fields via created catastrophes. *Experimental Mathematics*, 1(2):89–94, 1992.

[26] V. Pratt. Modeling concurrency with geometry. In *Proc. of the 18th ACM Symposium on Principles of Programming Languages*. ACM Press, 1991.

[27] T. A. Proebsting and C. W. Fraser. Detecting pipeline structural hazards quickly. In *Proc. of the Symposium on Principles of Programming Languages*. ACM Press, 1994.

[28] J.H. Reppy. *Higher-order concurrency*. PhD thesis, Department of Computer Science, Cornell University, 1992.

[29] J. Salz, R. Mirchandaney, and K. Crowley. Run-time parallelization and scheduling of loops. *IEEE Transactions on Computer*, 40(5), May 1991.

[30] M.W. Shields. Concurrent machines. *Computer Journal*, 28, 1985.

[31] A. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64:221–269, 1989.

[32] J. D. Ullman. *Principle of Database Systems*. Pitman, 1982.

[33] R. van Glabbeek. Bisimulation semantics for higher dimensional automata. Technical report, Stanford University, 1991.

[34] G. Winskel and M. Nielsen. *Models for concurrency*, volume 3 of Handbook of Logic in Computer Science, pages 100–200. Oxford University Press, 1994.

A The fundamental group

A.1 The fundamental group of length k

Let $P_1^k(M)$ be the set of all 1-paths of length k from $I \subseteq M_{0,0}$ in the semi-regular HDA M. It generates a sub-R-module of the product module $\underline{M}_{1,0} \times \underline{M}_{2,-1} \times \ldots \times \underline{M}_{k,-k+1}$: the addition and external multiplication are defined on each component of the paths.

Let $p = (p_i)_{1 \leq i \leq k}$ and $q = (q_i)_{1 \leq i \leq k}$ be two elements of $P_1^k(M)$. Then we say that p and q are equivalent or homotopic ($p \sim q$) if and only if p_i and q_i are connected in the complex of modules (see [20]) $((\underline{M}_{j,-i+1})_j, \partial_0)$ or in $((\underline{M}_{i-1,j})_j, \partial_1)$. It corresponds to our geometric definition of section 3.2. We define the **fundamental group** of M for paths of length k to be $\Pi_1^k(M) = P_1^k(M)/\sim$.

Proposition 1 *Let O be the image of $I \times M_{k,-k}$ by u such that $u(x,y) = (x-y) \in \underline{I} \oplus \underline{M}_{k,-k}$. Then, $\Pi_1^k(M) = H_1((\underline{M}, O), \partial_0 - \partial_1)$ where $H_1((\underline{M}, O), \partial_0 - \partial_1)$ is the first relative homology group of the pair ([20]) $N = (\underline{M}, O) = \underline{M}/O$ and boundary operator $\partial_0 - \partial_1$, i.e. is the quotient module $Ker(\partial_0 - \partial_1)_{|N_1}/(\partial_0 - \partial_1)(N_2)$.*

SKETCH OF PROOF. $Ker(\partial_0 - \partial_1)_{|N_1}$ contains the 1-paths of M starting from I and of length k since if $p = (p_1, \ldots, p_k)$ is such a path, $(\partial_0 - \partial_1)(p_1 + \ldots + p_k) = \partial_0(p_1) - \partial_1(p_k)$ which is null in N. Quotienting by $(\partial_0 - \partial_1)(N_2)$ amounts to taking them modulo homotopy. □

A.2 The functor Π_1^k

Let f be a morphism from the semi-regular automaton M to the semi-regular automaton N. Then f induces a morphism $\tilde{f}^k$ from the pair $(\underline{M}, O)$ to the pair $(\underline{N}, O')$ for all k. Then $\tilde{f}^k$ induces $H_1(\tilde{f}^k) : H_1(\underline{M}, O) \to H_1(\underline{N}, O')$. This defines $\Pi_1^k(f) = H_1(\tilde{f}^k)$ and makes Π_1^k into a covariant functor from the category of semi-regular automata to the category $R - Mod$ of R-modules.

All these definitions can be made starting from anywhere, not only $M_{0,0}$. For instance, if we consider initial states in $M_{p,q}$, we define in a similar manner the R-modules $P_1^{p,q,k}(M)$ and $\Pi_1^{p,q,k}(M)$, and the corresponding functors. We can also define submodules of these like $\Pi_1^{\alpha,\beta}(M)$ of paths from a state $\alpha \in M_{p,q}$ to a state $\beta \in M_{p+k-1,q-k+1}$

A.3 The functor Π_1

The real interesting homotopical object is the module of all *finite* paths modulo homotopy. The formal definition is as follows,

Definition 6 *The full fundamental group is*

$$\Pi_1(X) = \bigoplus_{\alpha,\beta \in X_0} \Pi_1^{\alpha,\beta}(X)/\{[f]_{\alpha,\beta} + [g]_{\beta,\gamma} = [f+g]_{\alpha,\gamma}\}$$

The quotient condition means that a sum of two classes of paths that may compose is equated to the class of the sum of the two paths. In this homotopy group, we cannot reverse time, but we can consider collections of "oriented" paths.

B Higher-order homotopy groups

We generalise the definition of the fundamental group. We had two "limiting" $(n-1)$-cubes in between which we could deform any sequence of n-cubes. That was defined with $n = 1$. For $n = 2$ we have a homotopy group of dimension 2 parameterized with two 1-paths p_1 and p_2, having the same initial and final states. Then in order to define a "full" homotopy group, we have to glue together all parameterized homotopy groups. The combinatorics of this glueing operation is much more complex than in dimension one and will not be described here.

The definition we give below is "iterative" in the sense that we know what a 1-path between α and β is (or what a 1-path of length k is) and that the definition of n-paths between $(n-1)$-paths depends on that definition.

Definition 7 *Let $n \geq 2$ and M be an acyclic HDA. Let p_1 and p_2 be two $(n-1)$-paths between two $(n-2)$-paths α and β. We suppose that $p_i = (p_i^1, \ldots, p_i^k)$ and that $p_i^1 \in M_{n-1+s,-s}$. The R-module of n-paths between p_1 and p_2 is the R-module of sequences $x = (x^1, \ldots, x^{k-1})$ with,*

- $x^1 \in \underline{M}_{n+s,-s}$,
- $\partial_0(x^{i+1}) = \partial^1(x^i) mod(p_1^{i+1} - p_2^{i+1})$,
- *module operations are pointwise addition and pointwise external multiplication.*

This R-module is named $P_n^{p_1,p_2}(M)$.

We say that two n-paths $p, q \in P_n^{p_1,p_2}(M)$ are homotopic, and we write $p \sim q$ if and only if $\forall i$, p_i and q_i are homotopic in the complex of modules $((\underline{M}_{j,-i+1})_j, \partial_0)$ or in $((\underline{M}_{n-i+2,j})_j, \partial_1)$ Let $\Pi_n^{p_1,p_2}(M) = P_n^{p_1,p_2}(M)/\sim$.

Proposition 2 $\Pi_n^{p_1,p_2}(M) = H_n((\underline{M}, T), \partial_0 - \partial_1)$ *where T is the image of $p_1 \oplus p_2$ under the map u with $u(x,y) = x - y$.*

Notice that Van Kampen's theorem [22] holds. This is one of the links with ordinary homotopy theory of topological spaces. We recall it for completeness of the paper. Let X_1, X_2 two regular HDA (subHDA of some domain D, [11]). Then $X_1 \cup X_2$ and $X_1 \cap X_2$ are well defined by a cocartesian diagram. Then Van Kampen's theorem asserts that the following diagram in $R-Mod$ is cocartesian as well,

$$\begin{tikzcd}
\Pi_n(X_1 \cap X_2) \arrow[r, "\Pi_n(j_1)"] \arrow[d, "\Pi_n(j_2)"] & \Pi_n(X_1) \arrow[d, "\Pi_n(i_1)"] \\
\Pi_n(X_2) \arrow[r, "\Pi_n(i_2)"] & \Pi_n(X_1 \cup X_2)
\end{tikzcd}$$

C Relationship with Herlihy's and Shavit's work

Here we are interested in the main result of [17] which can be roughly stated as follows: "There is a wait-free protocol for solving a given decision task if and only if its input complex can be continuously stretched and folded to cover its output complex".

To relate this to our framework, we first need to define input and ouput complexes. Let $P_0, \ldots, P_{N-1}$ be N sequential processes which, when run in parallel verify a protocol $\mathcal{P}$ solving a decision task $\mathcal{D}$. For the sake of simplicity, we suppose that the processes P_i are reduced to one action a_i. The HDA representing the program in the shared-memory paradigm is then $(a_0) \otimes \ldots \otimes (a_{N-1})$. The local input values are collected in the initial state $\alpha_0 \otimes \ldots \otimes \alpha_{N-1}$ and the vertices v_i in the input complex $(val(v_i), Id(v_i))$ with $val(v_i) = \alpha_i$ can be identified with the 1-transitions a_i. Compatibility of the vertices means that they begin a fully asynchronous execution of the processes $Id(v_i)$. This fully asynchronous execution is represented in the HDA model by the N-transition $a_0 \otimes \ldots \otimes a_{N-1}$ and in Herlihy's model by an $N-1$ simplex containing all the v_i. Once more, there is a perfect geometric correspondence between the two models, except the dimension has to be shifted by one. We should think of the HDA model as the "extension in time"[7] of the simplex-based model. In other terms, the input/output complexes are a kind of very useful denotational approximation of the operational behaviour given by HDA. Let us be a bit more precise about that.

The final state $\beta_0 \otimes \ldots \otimes \beta_{N-1}$ contains all output values of the processes. In a similar manner, the part of the output complex with vertices v_i such that $val(v_i) = \beta_i$ can be identified to (up to a shift of dimension one) the "vertical" complex composed of all end boundaries of $a_0 \otimes \ldots \otimes a_{N-1}$.

More generally, let $\mathcal{I}$ and $\mathcal{O}$ be input and output complexes of a protocol $\mathcal{P}$. Let $(s_i)_i$ and $(t_i)_i$ be the maximal simplexes (with respect to the inclusion ordering) in $\mathcal{I}$ and $\mathcal{O}$ respectively with $s_i = (v_i^0, \ldots, v_i^k)$ $(dim\ s_i = k)$ and $t_i = (w_i^0, \ldots, w_i^k)$ $(dim\ t_i = k)$. We suppose we have a semi-regular HDA D (called domain of HDA in [11]) in which we can fire any transition that the distributed system we are considering can execute, from any state of this machine. By what we have seen previously, the simplices s_i are in one-to-one correspondence with $(dim\ s_i+1)$-transitions D with initial state $v_i^0 \otimes \ldots \otimes v_i^{dim\ s_i}$. As a matter of fact suppose that the initial states permitted by the protocol are all in $D_{0,0}$

[7] This explains why we need one more dimension.

Figure 15: Input and output complex for some domain of HDA D, drawn in Herlihy's way at the right hand side.

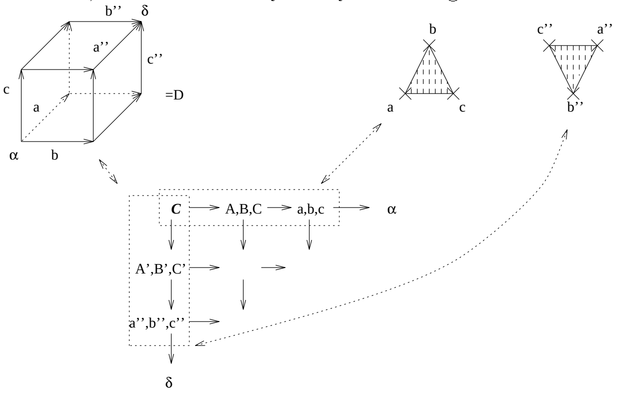
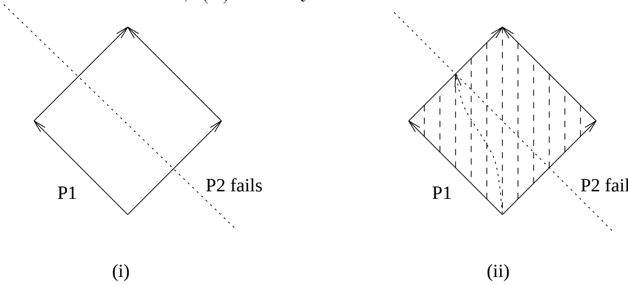


Figure 16: The effect of a failure of one process in (i)-a mutual exclusion, (ii)-a truly concurrent execution.



then there is a sub-semi-simplicial complex of D, isomorphic to the input complex (seen as a semi-simplicial set). This subcomplex is the "horizontal" complex $((D_{n+1,0})_n, (d_i^0)_i)$. The output complex is isomorphic to some subcomplex of D, which we identify now with $((D_{k,n+1-k})_n, (d_i^1)_i)$ if all final states are in $D_{k,-k}$. In between, there is all the paths transforming the input into the output complex (look at Figure 15).

We need now to see what is a wait-free computation in the HDA model, and why this implies that some topological properties are preserved from the input to the output complexes.

Look at Figure 16. In (i), the initial state is an internal nondeterministic choice. If we are not so lucky, the execution will begin by P_2 which fails to terminate: P_1 will never proceed and the computation is certainly not wait-free. In (ii), the execution is asynchronous between P_1 and P_2 and whatever happens to P_2, P_1 will terminate (one of the possible 1-paths is pictured). Hence, intuitively, to go from state α to state β in a wait free manner, we must have $\Pi_1^{\alpha,\beta}$ reduced to one class of paths. In the schedulers' point of view, this is the same as asking for the possible reordering for all executions of the failing processes after the terminating ones. This in turn could be shown to entail that the input and output complexes are homotopic which is precisely what we stated in intuitive language as "the input complex can be continuously stretched and folded to cover its output complex".